made with doc-o-matic

RAD Studio VCL Reference

Copyright(C) 2008 Embarcadero Technologies, Inc. All Rights Reserved.

Table of Contents

ScktComp Namespace 1

Classes **3**

ESocketError Class 3

TAbstractSocket Class 4

TAbstractSocket Members 5

TAbstractSocket Methods 5

TAbstractSocket Properties 11

TClientSocket Class 14

TClientSocket Members 15

TClientSocket.Create Constructor 16

TClientSocket.Destroy Destructor 16

TClientSocket Methods 17

TClientSocket Properties 18

TClientSocket Events 22

TClientWinSocket Class 25

TClientWinSocket Members 26

TClientWinSocket Methods 27

TClientWinSocket Properties 28

TCustomServerSocket Class 29

TCustomServerSocket Members 30

TCustomServerSocket.Destroy Destructor 31

TCustomServerSocket Fields 32

TCustomServerSocket Methods 32

TCustomServerSocket Properties 39

TCustomServerSocket Events 40

TCustomSocket Class 46

TCustomSocket Members 46

TCustomSocket Methods 47

TCustomSocket Events 49

TCustomWinSocket Class 54

TCustomWinSocket Members 54

TCustomWinSocket.Create Constructor 56

TCustomWinSocket.Destroy Destructor 56

TCustomWinSocket Methods 57

TCustomWinSocket Properties 71

TCustomWinSocket Events 78
TServerAcceptThread Class 80
TServerAcceptThread Members 80
TServerAcceptThread.Create Constructor 80
TServerAcceptThread Methods 80
TServerAcceptThread Properties 84
TServerClientThread Class 85
TServerClientThread Members 86
TServerClientThread.Create Constructor 86
TServerClientThread.Destroy Destructor 87
TServerClientThread Methods 87
TServerClientThread Properties 91
TServerClientWinSocket Class 93
TServerClientWinSocket Members 94
TServerClientWinSocket.Create Constructor 95
TServerClientWinSocket.Destroy Destructor 96
TServerClientWinSocket Properties 96
TServerSocket Class 97
TServerSocket Members 97
TServerSocket.Create Constructor 99
TServerSocket Properties 99
TServerSocket Events 103
TServerWinSocket Class 109
TServerWinSocket Members 110
TServerWinSocket.Create Constructor 112
TServerWinSocket.Destroy Destructor 112
TServerWinSocket Methods 113
TServerWinSocket Properties 118
TServerWinSocket Events 121
TWinSocketStream Class 127
TWinSocketStream Members 128
TWinSocketStream.Create Constructor 129
TWinSocketStream.Destroy Destructor 129
TWinSocketStream Methods 130
TWinSocketStream Properties 132

Functions **133**
ScktComp.SetErrorProc Function 133

Structs, Records, Enums **134**

ScktComp.TAsyncStyle Enumeration 134
ScktComp.TClientType Enumeration 135
ScktComp.TCMLookupComplete Record 135
ScktComp.TCMSocketMessage Record 136
ScktComp.TErrorEvent Enumeration 136
ScktComp.TLookupState Enumeration 137
ScktComp.TServerType Enumeration 137
ScktComp.TSocketEvent Enumeration 137

Types **139**
ScktComp.TAsyncStyles Type 139
ScktComp.TGetSocketEvent Type 139
ScktComp.TGetThreadEvent Type 140
ScktComp.TSocketErrorEvent Type 140
ScktComp.TSocketErrorProc Type 141
ScktComp.TSocketEventEvent Type 141
ScktComp.TSocketNotifyEvent Type 141
ScktComp.TThreadNotifyEvent Type 142

Constants **143**
ScktComp.CM_DEFERFREE Constant 143
ScktComp.CM_LOOKUPCOMPLETE Constant 143
ScktComp.CM_SOCKETMESSAGE Constant 143

SConnect Namespace 145

Classes **147**
EInterpreterError Class 147
ESocketConnectionError Class 148
TCustomDataBlockInterpreter Class 148
TCustomDataBlockInterpreter Members 149
TCustomDataBlockInterpreter Methods 149
TDataBlock Class 153
TDataBlock Members 153
TDataBlock.Create Constructor 155
TDataBlock.Destroy Destructor 155
TDataBlock Methods 156
TDataBlock Properties 160
TDataBlockInterpreter Class 163
TDataBlockInterpreter Members 163
TDataBlockInterpreter.Create Constructor 164

TDataBlockInterpreter.Destroy Destructor 164
TDataBlockInterpreter Methods 165
TDataDispatch Class 171
TDataDispatch Members 171
TDataDispatch.Create Constructor 172
TDataDispatch.Destroy Destructor 173
TDataDispatch Methods 173
TDataDispatch Properties 174
TPacketInterceptFactory Class 175
TPacketInterceptFactory Members 175
TPacketInterceptFactory Methods 176
TSocketConnection Class 176
TSocketConnection Members 177
TSocketConnection.Create Constructor 178
TSocketConnection Methods 179
TSocketConnection Properties 180
TSocketTransport Class 185
TSocketTransport Members 185
TSocketTransport.Create Constructor 186
TSocketTransport.Destroy Destructor 187
TSocketTransport Methods 187
TSocketTransport Properties 190
TStreamedConnection Class 191
TStreamedConnection Members 192
TStreamedConnection.Create Constructor 193
TStreamedConnection.Destroy Destructor 193
TStreamedConnection Methods 193
TStreamedConnection Properties 201
TTransportThread Class 204
TTransportThread Members 204
TTransportThread.Create Constructor 204
TTransportThread.Destroy Destructor 204
TTransportThread Methods 205
TTransportThread Properties 209
TWebConnection Class 209
TWebConnection Members 210
TWebConnection.Create Constructor 212
TWebConnection.Destroy Destructor 212
TWebConnection Methods 212

TWebConnection Properties 217

Interfaces 221

IDataBlock Interface 221

IDataBlock Members 222

IDataBlock Methods 222

IDataBlock Properties 227

IDataIntercept Interface 229

IDataIntercept Members 230

IDataIntercept Methods 230

ISendDataBlock Interface 231

ISendDataBlock Members 232

ISendDataBlock Methods 232

ITransport Interface 233

ITransport Members 233

ITransport Methods 234

ITransport Properties 236

Functions 237

SConnect.GetPacketInterceptorList Function 237

SConnect.LoadWinSock2 Function 237

Structs, Records, Enums 238

SConnect.TVarFlag Enumeration 238

Types 239

SConnect.PIntArray Type 239

SConnect.PVariantArray Type 239

SConnect.TIntArray Type 240

SConnect.TVarFlags Type 240

SConnect.TVariantArray Type 240

Variables 241

SConnect.WSACloseEvent Variable 241

SConnect.WSACreateEvent Variable 241

SConnect.WSAEventSelect Variable 242

SConnect.WSAResetEvent Variable 242

Constants 243

SConnect.asCreateObject Constant 243

SConnect.asError Constant 243

SConnect.asFreeObject Constant 244

SConnect.asGetAppServers Constant 244

SConnect.asGetGUID Constant 244
SConnect.asGetID Constant 245
SConnect.asGetServers Constant 245
SConnect.asInvoke Constant 245
SConnect.asMask Constant 246
SConnect.asSoapCommand Constant 246
SConnect.CallSig Constant 246
SConnect.ResultSig Constant 247
SConnect.THREAD_EXCEPTION Constant 247
SConnect.THREAD_RECEIVEDSTREAM Constant 247
SConnect.THREAD_REPLACETRANSPORT Constant 248
SConnect.THREAD_SENDNOTIFY Constant 248
SConnect.THREAD_SENDSTREAM Constant 248

ScreenTips Namespace 249

Classes 251
EAllocatedActionsException Class 251
TAllocatedActions Class 251
TAllocatedActions Members 252
TAllocatedActions.Create Constructor 252
TAllocatedActions.Destroy Destructor 252
TAllocatedActions Methods 253
TAllocatedActions Properties 254
TAssociateItem Class 255
TAssociateItem Members 255
TAssociateItem.Create Constructor 255
TAssociateItem.Destroy Destructor 255
TAssociateItem Properties 256
TAssociateItemList Class 256
TAssociateItemList Members 257
TAssociateItemList.Create Constructor 257
TAssociateItemList.Destroy Destructor 257
TAssociateItemList Methods 257
TCustomScreenTipsManager Class 259
TCustomScreenTipsManager Members 259
TCustomScreenTipsManager.Create Constructor 260
TCustomScreenTipsManager.Destroy Destructor 261
TCustomScreenTipsManager Methods 261

TCustomScreenTipsManager Properties 266
TCustomScreenTipsManager Events 269
TCustomScreenTipsPopup Class 270
TCustomScreenTipsPopup Members 271
TCustomScreenTipsPopup.Create Constructor 271
TCustomScreenTipsPopup.Destroy Destructor 272
TCustomScreenTipsPopup Methods 272
TCustomScreenTipsPopup Properties 275
TDisplayOffset Class 278
TDisplayOffset Members 278
TDisplayOffset.Create Constructor 278
TDisplayOffset Properties 279
TScreenTipBorderStyle Class 280
TScreenTipBorderStyle Members 280
TScreenTipBorderStyle.Create Constructor 280
TScreenTipBorderStyle Properties 281
TScreenTipItem Class 282
TScreenTipItem Members 282
TScreenTipItem.Create Constructor 283
TScreenTipItem.Destroy Destructor 283
TScreenTipItem Methods 283
TScreenTipItem Properties 285
TScreenTipsCollection Class 288
TScreenTipsCollection Members 288
TScreenTipsCollection.Create Constructor 289
TScreenTipsCollection Methods 289
TScreenTipsCollection Properties 291
TScreenTipsManager Class 291
TScreenTipsManager Members 292
TScreenTipsManager Properties 293
TScreenTipsManager Events 295
TScreenTipsPopup Class 297
TScreenTipsPopup Members 297
TScreenTipsPopup Properties 298
TScreenTipsWindow Class 305
TScreenTipsWindow Members 305
TScreenTipsWindow.Create Constructor 306
TScreenTipsWindow.Destroy Destructor 306
TScreenTipsWindow Methods 307

TScreenTipsWindow Properties 311

Interfaces 315

IScreenTipItem Interface 315

IScreenTipItem Members 315

IScreenTipItem Methods 315

Structs, Records, Enums 317

ScreenTips.TDescriptionType Enumeration 317

ScreenTips.TImageAlign Enumeration 317

ScreenTips.TItemImageAlign Enumeration 318

ScreenTips.TScreenTipOption Enumeration 318

ScreenTips.TScreenTipPopupType Enumeration 319

ScreenTips.TTipBorderStyle Enumeration 319

Types 320

ScreenTips.TCreateScreenTipEvent Type 320

ScreenTips.TDuplicateCreateErrorEvent Type 320

ScreenTips.TScreenTipOptions Type 321

ScreenTips.TScreenTipsWindowClass Type 321

ScreenTips.TScreenTipWinClass Type 321

ScreenTips.TShowScreenTipEvent Type 322

Constants 323

ScreenTips.cDefaultBorderColor Constant 323

ScreenTips.cDefaultEndColor Constant 323

ScreenTips.cDefaultStartColor Constant 324

ScreenTips.cScreenTipHeaderSpacing Constant 324

ScreenTips.cScreenTipMinimumHeight Constant 324

ScreenTips.cScreenTipTextAndImageWidth Constant 325

ScreenTips.cScreenTipTextOnlyWidth Constant 325

SessColn Namespace 327

Classes 328

ESessionsException Class 328

TAbstractSessionIDGen Class 328

TAbstractSessionIDGen Members 329

TAbstractSessionIDGen Methods 329

TAbstractSessionStatusChange Class 329

TAbstractSessionStatusChange Members 330

TAbstractSessionStatusChange Methods 330

TSessionIDGen Class 331
TSessionIDGen Members 331
TSessionIDGen.Create Constructor 331
TSessionIDGen Methods 332
TSessionItem Class 332
TSessionItem Members 332
TSessionItem.Create Constructor 333
TSessionItem.Destroy Destructor 333
TSessionItem Methods 334
TSessionItem Properties 335
TSessions Class 337
TSessions Members 338
TSessions.Create Constructor 338
TSessions.Destroy Destructor 339
TSessions Methods 339
TSessions Properties 346

Structs, Records, Enums **348**
SessColn.TSessionStatusFlag Enumeration 348

Types **349**
SessColn.TSessionStatusFlags Type 349

Variables **350**
SessColn.Sessions Variable 350

ShadowWnd Namespace 351

Classes **352**
TShadowWindow Class 352
TShadowWindow Members 352
TShadowWindow.Create Constructor 353
TShadowWindow.CreateShadow Constructor 353
TShadowWindow.Destroy Destructor 353
TShadowWindow Methods 354
TShadowWindow Properties 357

Structs, Records, Enums **358**
ShadowWnd.TControlSide Enumeration 358

ShareMem Namespace 359

Functions **360**
ShareMem.DumpBlocks Function 360
ShareMem.GetAllocMemCount Function 360
ShareMem.GetAllocMemSize Function 361
ShareMem.GetHeapStatus Function 361
ShareMem.HeapAddRef Function 361
ShareMem.HeapRelease Function 362
ShareMem.SysAllocMem Function 362
ShareMem.SysFreeMem Function 362
ShareMem.SysGetMem Function 363
ShareMem.SysReallocMem Function 363
ShareMem.SysRegisterExpectedMemoryLeak Function 363
ShareMem.SysUnregisterExpectedMemoryLeak Function 364

SHDocVw Namespace 365

Classes **371**
CoCScriptErrorList Class 371
CoCScriptErrorList Members 371
CoCScriptErrorList Methods 372
CoInternetExplorer Class 372
CoInternetExplorer Members 373
CoInternetExplorer Methods 373
CoSearchAssistantOC Class 373
CoSearchAssistantOC Members 374
CoSearchAssistantOC Methods 374
CoShellBrowserWindow Class 375
CoShellBrowserWindow Members 375
CoShellBrowserWindow Methods 375
CoShellNameSpace Class 376
CoShellNameSpace Members 376
CoShellNameSpace Methods 376
CoShellUIHelper Class 377
CoShellUIHelper Members 377
CoShellUIHelper Methods 378
CoShellWindows Class 378
CoShellWindows Members 379
CoShellWindows Methods 379
TInternetExplorer Class 379

TInternetExplorer Members 380
TInternetExplorer.Create Constructor 383
TInternetExplorer.Destroy Destructor 383
TInternetExplorer Methods 384
TInternetExplorer Properties 406
TInternetExplorer Events 416
TShellNameSpace Class 424
TShellNameSpace Members 424
TShellNameSpace.Create Constructor 425
TShellNameSpace.Destroy Destructor 426
TShellNameSpace Methods 427
TShellNameSpace Properties 437
TShellNameSpace Events 440
TShellUIHelper Class 441
TShellUIHelper Members 442
TShellUIHelper.Create Constructor 442
TShellUIHelper.Destroy Destructor 443
TShellUIHelper Methods 444
TShellUIHelper Properties 452
TShellWindows Class 452
TShellWindows Members 453
TShellWindows.Create Constructor 453
TShellWindows.Destroy Destructor 454
TShellWindows Methods 455
TShellWindows Properties 460
TShellWindows Events 461
TWebBrowser Class 462
TWebBrowser Members 462
TWebBrowser Methods 465
TWebBrowser Properties 477
TWebBrowser Events 502
TWebBrowser_V1 Class 526
TWebBrowser_V1 Members 526
TWebBrowser_V1 Methods 527
TWebBrowser_V1 Properties 533
TWebBrowser_V1 Events 553

Interfaces **571**
_SearchAssistantEvents Interface 571
_SearchAssistantEvents Members 572

_SearchAssistantEvents Methods 572
DShellNameSpaceEvents Interface 573
DShellNameSpaceEvents Members 573
DShellNameSpaceEvents Methods 573
DShellWindowsEvents Interface 574
DShellWindowsEvents Members 575
DShellWindowsEvents Methods 575
DWebBrowserEvents Interface 576
DWebBrowserEvents Members 576
DWebBrowserEvents Methods 576
DWebBrowserEvents2 Interface 581
DWebBrowserEvents2 Members 581
DWebBrowserEvents2 Methods 582
IScriptErrorList Interface 590
IScriptErrorList Members 590
IScriptErrorList Methods 591
IScriptErrorListDisp Interface 594
IScriptErrorListDisp Members 594
IScriptErrorListDisp Methods 595
ISearch Interface 598
ISearch Members 599
ISearch Methods 599
ISearch Properties 600
ISearchAssistantOC Interface 601
ISearchAssistantOC Members 601
ISearchAssistantOC Methods 602
ISearchAssistantOC Properties 609
ISearchAssistantOC2 Interface 611
ISearchAssistantOC2 Members 611
ISearchAssistantOC2 Methods 612
ISearchAssistantOC2 Properties 613
ISearchAssistantOC2Disp Interface 613
ISearchAssistantOC2Disp Members 614
ISearchAssistantOC2Disp Methods 614
ISearchAssistantOC2Disp Properties 619
ISearchAssistantOC3 Interface 621
ISearchAssistantOC3 Members 621
ISearchAssistantOC3 Methods 622
ISearchAssistantOC3 Properties 623

ISearchAssistantOC3Disp Interface 624
ISearchAssistantOC3Disp Members 624
ISearchAssistantOC3Disp Methods 625
ISearchAssistantOC3Disp Properties 629
ISearchAssistantOCDisp Interface 632
ISearchAssistantOCDisp Members 632
ISearchAssistantOCDisp Methods 633
ISearchAssistantOCDisp Properties 637
ISearchDisp Interface 639
ISearchDisp Members 640
ISearchDisp Properties 640
ISearches Interface 641
ISearches Members 641
ISearches Methods 642
ISearches Properties 643
ISearchesDisp Interface 643
ISearchesDisp Members 644
ISearchesDisp Methods 644
ISearchesDisp Properties 645
IShellFavoritesNameSpace Interface 645
IShellFavoritesNameSpace Members 646
IShellFavoritesNameSpace Methods 646
IShellFavoritesNameSpace Properties 649
IShellFavoritesNameSpaceDisp Interface 650
IShellFavoritesNameSpaceDisp Members 650
IShellFavoritesNameSpaceDisp Methods 651
IShellFavoritesNameSpaceDisp Properties 654
IShellNameSpace Interface 654
IShellNameSpace Members 655
IShellNameSpace Methods 656
IShellNameSpace Properties 661
IShellNameSpaceDisp Interface 663
IShellNameSpaceDisp Members 664
IShellNameSpaceDisp Methods 664
IShellNameSpaceDisp Properties 668
IShellUIHelper Interface 671
IShellUIHelper Members 671
IShellUIHelper Methods 672
IShellUIHelperDisp Interface 675

IShellUIHelperDisp Members 675
IShellUIHelperDisp Methods 676
IShellWindows Interface 679
IShellWindows Members 679
IShellWindows Methods 680
IShellWindows Properties 683
IShellWindowsDisp Interface 683
IShellWindowsDisp Members 683
IShellWindowsDisp Methods 684
IShellWindowsDisp Properties 686
IWebBrowser Interface 687
IWebBrowser Members 687
IWebBrowser Methods 688
IWebBrowser Properties 694
IWebBrowser2 Interface 698
IWebBrowser2 Members 698
IWebBrowser2 Methods 700
IWebBrowser2 Properties 705
IWebBrowser2Disp Interface 707
IWebBrowser2Disp Members 707
IWebBrowser2Disp Methods 708
IWebBrowser2Disp Properties 712
IWebBrowserApp Interface 720
IWebBrowserApp Members 720
IWebBrowserApp Methods 722
IWebBrowserApp Properties 727
IWebBrowserAppDisp Interface 729
IWebBrowserAppDisp Members 730
IWebBrowserAppDisp Methods 730
IWebBrowserAppDisp Properties 733
IWebBrowserDisp Interface 739
IWebBrowserDisp Members 739
IWebBrowserDisp Methods 740
IWebBrowserDisp Properties 742

Types **746**

SHDocVw.BrowserNavConstants Type 748
SHDocVw.CommandStateChangeConstants Type 748
SHDocVw.CScriptErrorList Type 748
SHDocVw.InternetExplorer Type 749

SHDocVw.OLECMDEXECOPT Type 749
SHDocVw.OLECMDF Type 750
SHDocVw.OLECMDID Type 750
SHDocVw.POleVariant1 Type 751
SHDocVw.RefreshConstants Type 751
SHDocVw.SearchAssistantOC Type 751
SHDocVw.SecureLockIconConstants Type 752
SHDocVw.ShellBrowserWindow Type 752
SHDocVw.ShellNameSpace Type 752
SHDocVw.ShellUIHelper Type 753
SHDocVw.ShellWindowFindWindowOptions Type 753
SHDocVw.ShellWindows Type 753
SHDocVw.ShellWindowTypeConstants Type 754
SHDocVw.tagREADYSTATE Type 754
SHDocVw.TInternetExplorerBeforeNavigate2 Type 754
SHDocVw.TInternetExplorerClientToHostWindow Type 755
SHDocVw.TInternetExplorerCommandStateChange Type 755
SHDocVw.TInternetExplorerDocumentComplete Type 755
SHDocVw.TInternetExplorerFileDownload Type 756
SHDocVw.TInternetExplorerNavigateComplete2 Type 756
SHDocVw.TInternetExplorerNavigateError Type 756
SHDocVw.TInternetExplorerNewWindow2 Type 757
SHDocVw.TInternetExplorerOnFullScreen Type 757
SHDocVw.TInternetExplorerOnMenuBar Type 757
SHDocVw.TInternetExplorerOnStatusBar Type 758
SHDocVw.TInternetExplorerOnTheaterMode Type 758
SHDocVw.TInternetExplorerOnToolBar Type 758
SHDocVw.TInternetExplorerOnVisible Type 759
SHDocVw.TInternetExplorerPrintTemplateInstantiation Type 759
SHDocVw.TInternetExplorerPrintTemplateTeardown Type 759
SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type 760
SHDocVw.TInternetExplorerProgressChange Type 760
SHDocVw.TInternetExplorerPropertyChange Type 760
SHDocVw.TInternetExplorerSetSecureLockIcon Type 761
SHDocVw.TInternetExplorerStatusTextChange Type 761
SHDocVw.TInternetExplorerTitleChange Type 761
SHDocVw.TInternetExplorerUpdatePageStatus Type 762
SHDocVw.TInternetExplorerWindowClosing Type 762
SHDocVw.TInternetExplorerWindowSetHeight Type 762

SHDocVw.TInternetExplorerWindowSetLeft Type 763
SHDocVw.TInternetExplorerWindowSetResizable Type 763
SHDocVw.TInternetExplorerWindowSetTop Type 763
SHDocVw.TInternetExplorerWindowSetWidth Type 764
SHDocVw.TShellNameSpaceFavoritesSelectionChange Type 764
SHDocVw.TShellWindowsWindowRegistered Type 764
SHDocVw.TShellWindowsWindowRevoked Type 765
SHDocVw.TWebBrowser_V1BeforeNavigate Type 765
SHDocVw.TWebBrowser_V1CommandStateChange Type 765
SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type 766
SHDocVw.TWebBrowser_V1FrameNavigateComplete Type 766
SHDocVw.TWebBrowser_V1FrameNewWindow Type 766
SHDocVw.TWebBrowser_V1NavigateComplete Type 767
SHDocVw.TWebBrowser_V1NewWindow Type 767
SHDocVw.TWebBrowser_V1ProgressChange Type 768
SHDocVw.TWebBrowser_V1PropertyChange Type 768
SHDocVw.TWebBrowser_V1Quit Type 768
SHDocVw.TWebBrowser_V1StatusTextChange Type 769
SHDocVw.TWebBrowser_V1TitleChange Type 769
SHDocVw.TWebBrowserBeforeNavigate2 Type 769
SHDocVw.TWebBrowserClientToHostWindow Type 770
SHDocVw.TWebBrowserCommandStateChange Type 770
SHDocVw.TWebBrowserDocumentComplete Type 770
SHDocVw.TWebBrowserFileDownload Type 771
SHDocVw.TWebBrowserNavigateComplete2 Type 771
SHDocVw.TWebBrowserNavigateError Type 771
SHDocVw.TWebBrowserNewWindow2 Type 772
SHDocVw.TWebBrowserOnFullScreen Type 772
SHDocVw.TWebBrowserOnMenuBar Type 772
SHDocVw.TWebBrowserOnStatusBar Type 773
SHDocVw.TWebBrowserOnTheaterMode Type 773
SHDocVw.TWebBrowserOnToolBar Type 773
SHDocVw.TWebBrowserOnVisible Type 774
SHDocVw.TWebBrowserPrintTemplateInstantiation Type 774
SHDocVw.TWebBrowserPrintTemplateTeardown Type 774
SHDocVw.TWebBrowserPrivacyImpactedStateChange Type 775
SHDocVw.TWebBrowserProgressChange Type 775
SHDocVw.TWebBrowserPropertyChange Type 775
SHDocVw.TWebBrowserSetSecureLockIcon Type 776

SHDocVw.TWebBrowserStatusTextChange Type 776
SHDocVw.TWebBrowserTitleChange Type 776
SHDocVw.TWebBrowserUpdatePageStatus Type 777
SHDocVw.TWebBrowserWindowClosing Type 777
SHDocVw.TWebBrowserWindowSetHeight Type 777
SHDocVw.TWebBrowserWindowSetLeft Type 778
SHDocVw.TWebBrowserWindowSetResizable Type 778
SHDocVw.TWebBrowserWindowSetTop Type 778
SHDocVw.TWebBrowserWindowSetWidth Type 779
SHDocVw.WebBrowser Type 779
SHDocVw.WebBrowser_V1 Type 779

Constants 780

SHDocVw.CGID_IWebBrowser Constant 782
SHDocVw.CLASS_CScriptErrorList Constant 783
SHDocVw.CLASS_InternetExplorer Constant 783
SHDocVw.CLASS_SearchAssistantOC Constant 783
SHDocVw.CLASS_ShellBrowserWindow Constant 784
SHDocVw.CLASS_ShellNameSpace Constant 784
SHDocVw.CLASS_ShellUIHelper Constant 784
SHDocVw.CLASS_ShellWindows Constant 785
SHDocVw.CLASS_WebBrowser Constant 785
SHDocVw.CLASS_WebBrowser_V1 Constant 785
SHDocVw.CSC_NAVIGATEBACK Constant 786
SHDocVw.CSC_NAVIGATEFORWARD Constant 786
SHDocVw.CSC_UPDATECOMMANDS Constant 786
SHDocVw.DIID__SearchAssistantEvents Constant 787
SHDocVw.DIID_DShellNameSpaceEvents Constant 787
SHDocVw.DIID_DShellWindowsEvents Constant 787
SHDocVw.DIID_DWebBrowserEvents Constant 788
SHDocVw.DIID_DWebBrowserEvents2 Constant 788
SHDocVw.dtlOcxPage Constant 788
SHDocVw.dtlServerPage Constant 789
SHDocVw.HTMLID_FIND Constant 789
SHDocVw.HTMLID_OPTIONS Constant 789
SHDocVw.HTMLID_VIEWSOURCE Constant 790
SHDocVw.IID_IScriptErrorList Constant 790
SHDocVw.IID_ISearch Constant 790
SHDocVw.IID_ISearchAssistantOC Constant 791
SHDocVw.IID_ISearchAssistantOC2 Constant 791

SHDocVw.IID_ISearchAssistantOC3 Constant 791
SHDocVw.IID_ISearches Constant 792
SHDocVw.IID_IShellFavoritesNameSpace Constant 792
SHDocVw.IID_IShellNameSpace Constant 792
SHDocVw.IID_IShellUIHelper Constant 793
SHDocVw.IID_IShellWindows Constant 793
SHDocVw.IID_IWebBrowser Constant 793
SHDocVw.IID_IWebBrowser2 Constant 794
SHDocVw.IID_IWebBrowserApp Constant 794
SHDocVw.LIBID_SHDocVw Constant 794
SHDocVw.navAllowAutosearch Constant 795
SHDocVw.navBrowserBar Constant 795
SHDocVw.navNoHistory Constant 795
SHDocVw.navNoReadFromCache Constant 796
SHDocVw.navNoWriteToCache Constant 796
SHDocVw.navOpenInNewWindow Constant 796
SHDocVw.OLECMDEXECOPT_DODEFAULT Constant 797
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant 797
SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant 797
SHDocVw.OLECMDEXECOPT_SHOWHELP Constant 798
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant 798
SHDocVw.OLECMDF_ENABLED Constant 798
SHDocVw.OLECMDF_INVISIBLE Constant 799
SHDocVw.OLECMDF_LATCHED Constant 799
SHDocVw.OLECMDF_NINCHED Constant 799
SHDocVw.OLECMDF_SUPPORTED Constant 800
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant 800
SHDocVw.OLECMDID_CLEARSELECTION Constant 800
SHDocVw.OLECMDID_CLOSE Constant 801
SHDocVw.OLECMDID_COPY Constant 801
SHDocVw.OLECMDID_CUT Constant 801
SHDocVw.OLECMDID_DELETE Constant 802
SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant 802
SHDocVw.OLECMDID_ENABLE_INTERACTION Constant 802
SHDocVw.OLECMDID_FIND Constant 803
SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant 803
SHDocVw.OLECMDID_GETZOOMRANGE Constant 803
SHDocVw.OLECMDID_HIDETOOLBARS Constant 804
SHDocVw.OLECMDID_HTTPEQUIV Constant 804

SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant 804
SHDocVw.OLECMDID_NEW Constant 805
SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant 805
SHDocVw.OLECMDID_ONUNLOAD Constant 805
SHDocVw.OLECMDID_OPEN Constant 806
SHDocVw.OLECMDID_PAGESETUP Constant 806
SHDocVw.OLECMDID_PASTE Constant 806
SHDocVw.OLECMDID_PASTESPECIAL Constant 807
SHDocVw.OLECMDID_PREREFRESH Constant 807
SHDocVw.OLECMDID_PRINT Constant 807
SHDocVw.OLECMDID_PRINT2 Constant 808
SHDocVw.OLECMDID_PRINTPREVIEW Constant 808
SHDocVw.OLECMDID_PRINTPREVIEW2 Constant 808
SHDocVw.OLECMDID_PROPERTIES Constant 809
SHDocVw.OLECMDID_PROPERTYBAG2 Constant 809
SHDocVw.OLECMDID_REDO Constant 809
SHDocVw.OLECMDID_REFRESH Constant 810
SHDocVw.OLECMDID_SAVE Constant 810
SHDocVw.OLECMDID_SAVEAS Constant 810
SHDocVw.OLECMDID_SAVECOPYAS Constant 811
SHDocVw.OLECMDID_SELECTALL Constant 811
SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant 811
SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant 812
SHDocVw.OLECMDID_SETPROGRESSMAX Constant 812
SHDocVw.OLECMDID_SETPROGRESSPOS Constant 812
SHDocVw.OLECMDID_SETPROGRESSTEXT Constant 813
SHDocVw.OLECMDID_SETTITLE Constant 813
SHDocVw.OLECMDID_SHOWFIND Constant 813
SHDocVw.OLECMDID_SHOWMESSAGE Constant 814
SHDocVw.OLECMDID_SHOWPAGESETUP Constant 814
SHDocVw.OLECMDID_SHOWPRINT Constant 814
SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant 815
SHDocVw.OLECMDID_SPELL Constant 815
SHDocVw.OLECMDID_STOP Constant 815
SHDocVw.OLECMDID_STOPDOWNLOAD Constant 816
SHDocVw.OLECMDID_UNDO Constant 816
SHDocVw.OLECMDID_UPDATECOMMANDS Constant 816
SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant 817
SHDocVw.OLECMDID_ZOOM Constant 817

SHDocVw.READYSTATE_COMPLETE Constant 817
SHDocVw.READYSTATE_INTERACTIVE Constant 818
SHDocVw.READYSTATE_LOADED Constant 818
SHDocVw.READYSTATE_LOADING Constant 818
SHDocVw.READYSTATE_UNINITIALIZED Constant 819
SHDocVw.REFRESH_COMPLETELY Constant 819
SHDocVw.REFRESH_CONTINUE Constant 819
SHDocVw.REFRESH_IFEXPIRED Constant 820
SHDocVw.REFRESH_NORMAL Constant 820
SHDocVw.secureLockIconMixed Constant 820
SHDocVw.secureLockIconSecure128Bit Constant 821
SHDocVw.secureLockIconSecure40Bit Constant 821
SHDocVw.secureLockIconSecure56Bit Constant 821
SHDocVw.secureLockIconSecureFortezza Constant 822
SHDocVw.secureLockIconSecureUnknownBits Constant 822
SHDocVw.secureLockIconUnsecure Constant 822
SHDocVw.SHDocVwMajorVersion Constant 823
SHDocVw.SHDocVwMinorVersion Constant 823
SHDocVw.SWC_3RDPARTY Constant 823
SHDocVw.SWC_BROWSER Constant 824
SHDocVw.SWC_CALLBACK Constant 824
SHDocVw.SWC_EXPLORER Constant 824
SHDocVw.SWFO_COOKIEPASSED Constant 825
SHDocVw.SWFO_INCLUDEPENDING Constant 825
SHDocVw.SWFO_NEEDDISPATCH Constant 825

ShellAnimations Namespace 827

Classes **828**
TShellResources Class 828

SimpleDS Namespace 829

Classes **830**
TInternalSQLDataSet Class 830
TInternalSQLDataSet Members 830
TInternalSQLDataSet Properties 833
TSimpleDataSet Class 838
TSimpleDataSet Members 839
TSimpleDataSet.Create Constructor 841

TSimpleDataSet.Destroy Destructor 842
TSimpleDataSet Methods 842
TSimpleDataSet Properties 845
TSimpleDataSet Events 872

SiteComp Namespace 901

Classes 905
TAbstractDesigntimeWarnings Class 905
TAbstractDesigntimeWarnings Members 905
TAbstractDesigntimeWarnings Methods 906
TAbstractInetFileType Class 907
TAbstractInetFileType Members 907
TAbstractInetFileType Methods 908
TAbstractInetFileType Properties 909
TNotifyList Class 910
TNotifyList Members 910
TNotifyList.Create Constructor 911
TNotifyList.Destroy Destructor 912
TNotifyList Methods 912
TNotifyList Properties 914
TScriptComponent Class 915
TScriptComponent Members 915
TScriptComponent.Create Constructor 917
TScriptComponent Methods 917
TScriptComponent Properties 917
TScriptObject Class 918

Interfaces 919
IActionImageProducer Interface 920
IActionImageProducer Members 921
IActionImageProducer Methods 921
IAdapterAccess Interface 923
IAdapterAccess Members 923
IAdapterAccess Methods 923
IAdapterActionAccess Interface 924
IAdapterActionAccess Members 925
IAdapterActionAccess Methods 925
IAdapterActionAttributes Interface 925
IAdapterActionAttributes Members 926

IAdapterActionAttributes Methods 926
IAdapterActionAttributes Properties 926
IAdapterDispatcher Interface 927
IAdapterDispatcher Members 927
IAdapterDispatcher Methods 928
IAdapterDispatchParams Interface 929
IAdapterDispatchParams Members 930
IAdapterDispatchParams Methods 930
IAdapterDispatchParams Properties 932
IAdapterError Interface 934
IAdapterError Members 934
IAdapterError Methods 934
IAdapterError Properties 935
IAdapterErrors Interface 937
IAdapterErrors Members 937
IAdapterErrors Methods 937
IAdapterErrors Properties 939
IAdapterFieldAccess Interface 940
IAdapterFieldAccess Members 941
IAdapterFieldAccess Methods 941
IAdapterFieldAttributes Interface 941
IAdapterFieldAttributes Members 942
IAdapterFieldAttributes Methods 942
IAdapterFieldAttributes Properties 943
IAdapterHiddenField Interface 943
IAdapterHiddenField Members 944
IAdapterHiddenField Methods 944
IAdapterHiddenField Properties 945
IAdapterHiddenFields Interface 945
IAdapterHiddenFields Members 946
IAdapterHiddenFields Methods 946
IAdapterHiddenFields Properties 946
IAdapterItemRequestParams Interface 947
IAdapterItemRequestParams Members 948
IAdapterItemRequestParams Methods 948
IAdapterItemRequestParams Properties 950
IAdapterMode Interface 951
IAdapterMode Members 952
IAdapterMode Methods 952

IAdapterMode Properties 953
IAdapterRequestHandler Interface 954
IAdapterRequestHandler Members 954
IAdapterRequestHandler Methods 954
IAsFormatted Interface 955
IAsFormatted Members 956
IAsFormatted Methods 956
IDefaultPageFileName Interface 956
IDefaultPageFileName Members 957
IDefaultPageFileName Methods 957
IFieldValuesAdapter Interface 957
IFieldValuesAdapter Members 958
IFieldValuesAdapter Methods 958
IFindIncludeFile Interface 958
IFindIncludeFile Members 959
IFindIncludeFile Methods 959
IGetAdapterActions Interface 959
IGetAdapterActions Members 960
IGetAdapterActions Methods 960
IGetAdapterDisplayCharacteristics Interface 960
IGetAdapterDisplayCharacteristics Members 961
IGetAdapterDisplayCharacteristics Methods 961
IGetAdapterErrors Interface 961
IGetAdapterErrors Members 962
IGetAdapterErrors Methods 962
IGetAdapterFields Interface 962
IGetAdapterFields Members 963
IGetAdapterFields Methods 963
IGetAdapterHiddenFields Interface 963
IGetAdapterHiddenFields Members 963
IGetAdapterHiddenFields Methods 964
IGetAdapterImage Interface 964
IGetAdapterImage Members 965
IGetAdapterImage Methods 965
IGetAdapterItemRequestParams Interface 965
IGetAdapterItemRequestParams Members 966
IGetAdapterItemRequestParams Methods 966
IGetAdapterRequestDefaultResponse Interface 966
IGetAdapterRequestDefaultResponse Members 967

IGetAdapterRequestDefaultResponse Methods 967
IGetAdapterRequestDefaultResponse Properties 968
IGetAdapterValuesList Interface 968
IGetAdapterValuesList Members 969
IGetAdapterValuesList Methods 969
IGetDefaultAction Interface 969
IGetDefaultAction Members 969
IGetDefaultAction Methods 970
IGetDefaultAction Properties 970
IGetDesigntimeWarnings Interface 971
IGetDesigntimeWarnings Members 971
IGetDesigntimeWarnings Methods 971
IGetFieldValuesAdapter Interface 972
IGetFieldValuesAdapter Members 972
IGetFieldValuesAdapter Methods 972
IGetHTMLStyle Interface 973
IGetHTMLStyle Members 973
IGetHTMLStyle Methods 973
IGetProducerComponent Interface 976
IGetProducerComponent Members 976
IGetProducerComponent Methods 976
IGetScriptObject Interface 977
IGetScriptObject Members 977
IGetScriptObject Methods 977
IGetScriptObjectClassKey Interface 978
IGetScriptObjectClassKey Members 978
IGetScriptObjectClassKey Methods 978
IGetWebAppComponents Interface 979
IGetWebAppComponents Members 979
IGetWebAppComponents Methods 979
IGetWebAppComponents Properties 981
IIDEFileManager Interface 982
IIDEFileManager Members 982
IIDEFileManager Methods 982
IImageProducer Interface 983
IImageProducer Members 983
IImageProducer Methods 984
IInternetEnvOptions Interface 984
IInternetEnvOptions Members 984

IInternetEnvOptions Methods 985
IInternetEnvOptions Properties 986
IInternetExecuteScript Interface 987
IInternetExecuteScript Members 987
IInternetExecuteScript Methods 987
IInternetFileTypes Interface 988
IInternetFileTypes Members 988
IInternetFileTypes Methods 989
IInternetScriptDebugger Interface 989
IInternetScriptDebugger Members 989
IInternetScriptDebugger Methods 990
IIsAdapterActionList Interface 990
IIsAdapterActionList Members 991
IIsAdapterActionList Methods 991
IIsAdapterActionList Properties 991
IIterateIntfSupport Interface 992
IIterateIntfSupport Members 992
IIterateIntfSupport Methods 992
IIterateObjectSupport Interface 993
IIterateObjectSupport Members 993
IIterateObjectSupport Methods 993
IIteratorIndex Interface 994
IIteratorIndex Members 994
IIteratorIndex Methods 994
IIteratorIndex Properties 995
IIteratorSupport Interface 996
IIteratorSupport Members 997
IIteratorSupport Methods 997
INotifyAdapterChange Interface 998
INotifyAdapterChange Members 998
INotifyAdapterChange Methods 999
INotifyList Interface 999
INotifyList Members 1000
INotifyList Methods 1000
IPageDispatcher Interface 1001
IPageDispatcher Members 1001
IPageDispatcher Methods 1002
IPageDispatcher Properties 1005
IPageResult Interface 1006

IPageResult Members 1006
IPageResult Methods 1006
ISessionsService Interface 1007
ISessionsService Members 1008
ISessionsService Methods 1008
ISessionsServiceConnectID Interface 1010
ISessionsServiceConnectID Members 1010
ISessionsServiceConnectID Methods 1010
ISessionsServicePersist Interface 1011
ISessionsServicePersist Members 1011
ISessionsServicePersist Methods 1011
ISessionsServiceStoreID Interface 1013
ISessionsServiceStoreID Members 1013
ISessionsServiceStoreID Methods 1013
IValuesListAdapter Interface 1014
IValuesListAdapter Members 1015
IValuesListAdapter Methods 1015
IWebActionsList Interface 1017
IWebActionsList Members 1018
IWebActionsList Methods 1018
IWebActionsList Properties 1019
IWebApplicationInfo Interface 1019
IWebApplicationInfo Members 1020
IWebApplicationInfo Methods 1020
IWebApplicationInfo Properties 1021
IWebContainerSubComponent Interface 1022
IWebDataFields Interface 1023
IWebDataFields Members 1023
IWebDataFields Methods 1023
IWebDataFields Properties 1024
IWebEnabled Interface 1025
IWebEnabled Members 1025
IWebEnabled Methods 1025
IWebFormComponentContainer Interface 1026
IWebFormComponentContainer Members 1026
IWebFormComponentContainer Methods 1026
IWebFormComponentContainer Properties 1027
IWebGetActionName Interface 1028
IWebGetActionName Members 1028

IWebGetActionName Methods 1028
IWebGetActionName Properties 1029
IWebGetDisplayLabel Interface 1029
IWebGetDisplayLabel Members 1030
IWebGetDisplayLabel Methods 1030
IWebGetDisplayWidth Interface 1030
IWebGetDisplayWidth Members 1030
IWebGetDisplayWidth Methods 1031
IWebGetFieldName Interface 1031
IWebGetFieldName Members 1031
IWebGetFieldName Methods 1032
IWebGetFieldName Properties 1032
IWebGetFieldValue Interface 1032
IWebGetFieldValue Members 1033
IWebGetFieldValue Methods 1033
IWebGetFieldValue Properties 1033
IWebGetFieldValues Interface 1034
IWebGetFieldValues Members 1034
IWebGetFieldValues Methods 1034
IWebGetMaxLength Interface 1037
IWebGetMaxLength Members 1037
IWebGetMaxLength Methods 1037
IWebGetSessionIDs Interface 1038
IWebGetSessionIDs Members 1038
IWebGetSessionIDs Methods 1038
IWebImageHREF Interface 1039
IWebImageHREF Members 1039
IWebImageHREF Methods 1039
IWebInputName Interface 1040
IWebInputName Members 1040
IWebInputName Methods 1041
IWebInputName Properties 1041
IWebIsDefaultAction Interface 1041
IWebIsDefaultAction Members 1042
IWebIsDefaultAction Methods 1042
IWebIsDefaultField Interface 1042
IWebIsDefaultField Members 1043
IWebIsDefaultField Methods 1043
IWebSession Interface 1043

IWebSession Members 1044
IWebSession Methods 1044
IWebSessionAttributes Interface 1045
IWebSessionAttributes Members 1045
IWebSessionAttributes Methods 1046
IWebSessionAttributes Properties 1048
IWebSessionID Interface 1049
IWebSessionID Members 1050
IWebSessionID Methods 1050
IWebSessionID Properties 1050
IWebSessionIDs Interface 1051
IWebSessionIDs Members 1051
IWebSessionIDs Methods 1052
IWebSessionIDs Properties 1052
IWebSessionValues Interface 1053
IWebSessionValues Members 1054
IWebSessionValues Methods 1054
IWebSessionValues Properties 1055
IWebSetActionName Interface 1055
IWebSetActionName Members 1056
IWebSetActionName Methods 1056
IWebSetFieldName Interface 1056
IWebSetFieldName Members 1057
IWebSetFieldName Methods 1057
IWebUserList Interface 1057
IWebUserList Members 1058
IWebUserList Methods 1058
IWebValueIsEqual Interface 1059
IWebValueIsEqual Members 1060
IWebValueIsEqual Methods 1060

Functions **1061**
SiteComp.FullyQualifiedFieldName Function 1061
SiteComp.FullyQualifiedName Function 1061

Structs, Records, Enums **1062**
SiteComp.TAdapterActionHTMLElementType Enumeration 1062
SiteComp.TAdapterDisplayCharacteristic Enumeration 1062
SiteComp.TAdapterDisplayHTMLElementType Enumeration 1063
SiteComp.TAdapterDisplayViewModeType Enumeration 1063

SiteComp.TAdapterInputHTMLElementType Enumeration 1064
SiteComp.TAdapterItemRequestParamOption Enumeration 1065
SiteComp.TDispatchPageFlag Enumeration 1065
SiteComp.TQualifyOption Enumeration 1065

Types **1067**
SiteComp.TAdapterDisplayCharacteristics Type 1067
SiteComp.TAdapterItemRequestParamOptions Type 1067
SiteComp.TDispatchPageFlags Type 1068
SiteComp.TSessionID Type 1068

Variables **1069**
SiteComp.ExecuteScript Variable 1069
SiteComp.GetHTMLSampleImage Variable 1069
SiteComp.IDEFileManager Variable 1070
SiteComp.InetFileTypes Variable 1070
SiteComp.InternetEnvOptions Variable 1070
SiteComp.ScriptDebugger Variable 1071

Constants **1072**
SiteComp.AdapterActionHTMLElementTypeNames Constant 1072
SiteComp.AdapterDisplayHTMLElementTypeNames Constant 1072
SiteComp.AdapterDisplayViewModeTypeNames Constant 1073
SiteComp.AdapterInputHTMLElementTypeNames Constant 1073

SiteProd Namespace 1075

Classes **1076**
TAdapterPageProducer Class 1076
TAdapterPageProducer Members 1077
TAdapterPageProducer Properties 1078
TAdapterPageProducer Events 1082
TBaseAdapterPageProducer Class 1083
TBaseAdapterPageProducer Members 1084
TBaseAdapterPageProducer.Create Constructor 1085
TBaseAdapterPageProducer.Destroy Destructor 1085
TBaseAdapterPageProducer Methods 1085
TBaseAdapterPageProducer Properties 1096
TBaseAdapterPageProducer Events 1098
TCustomAdapterPageProducer Class 1099
TCustomAdapterPageProducer Members 1100

TCustomAdapterPageProducer.Create Constructor 1101
TCustomAdapterPageProducer.Destroy Destructor 1102
TCustomAdapterPageProducer Methods 1102
TCustomAdapterPageProducer Properties 1105
TSitePageElements Class 1107
TSitePageElements Members 1107
TSitePageElements Fields 1108

SOAPAttachIntf Namespace 1111

Classes 1112
TSoapDataList Class 1112
TSoapDataList Members 1112
TSoapDataList Methods 1113

Interfaces 1116
IMimeAttachmentHandler Interface 1116
IMimeAttachmentHandler Members 1116
IMimeAttachmentHandler Methods 1117
IMimeAttachmentHandler Properties 1122
IMimeAttachmentHandler Events 1123

Types 1125
SOAPAttachIntf.TOnGetAttachmentEvent Type 1125
SOAPAttachIntf.TOnSendAttachmentEvent Type 1125

SOAPConn Namespace 1127

Classes 1128
TSoapConnection Class 1128
TSoapConnection Members 1129
TSoapConnection.Create Constructor 1130
TSoapConnection.Destroy Destructor 1131
TSoapConnection Methods 1131
TSoapConnection Properties 1135
TSoapConnection Events 1141

SOAPDm Namespace 1145

Classes 1146
TSoapDataModule Class 1146

TSoapDataModule Members 1147
TSoapDataModule.Create Constructor 1148
TSoapDataModule.Destroy Destructor 1148
TSoapDataModule Methods 1149
TSoapDataModule Properties 1167

SOAPDomConv Namespace 1171

Classes **1172**
TSOAPDOMProcessor Class 1172
TSOAPDOMProcessor Members 1172
TSOAPDOMProcessor.Destroy Destructor 1173
TSOAPDOMProcessor Fields 1173
TSOAPDOMProcessor Methods 1174

Interfaces **1179**
IDOMHeaderProcessor Interface 1179
IDOMHeaderProcessor Members 1180
IDOMHeaderProcessor Methods 1180

Structs, Records, Enums **1182**
SOAPDomConv.TDOMHeaderProcessorEntry Record 1182

Types **1183**
SOAPDomConv.TDOMHeaderProcessorArray Type 1183

SOAPHTTPClient Namespace 1185

Classes **1186**
THTTPRIO Class 1186
THTTPRIO Members 1187
THTTPRIO.Create Constructor 1187
THTTPRIO.Destroy Destructor 1188
THTTPRIO Methods 1188
THTTPRIO Properties 1190

SOAPHTTPDisp Namespace 1195

Classes **1196**
THTTPSoapDispatchNode Class 1196
THTTPSoapDispatchNode Members 1196
THTTPSoapDispatchNode Fields 1197

THTTPSoapDispatchNode Methods 1197
THTTPSoapDispatchNode Properties 1198

Interfaces 1200
IHTTPSoapDispatch Interface 1200
IHTTPSoapDispatch Members 1200
IHTTPSoapDispatch Methods 1201

SOAPHTTPPasInv Namespace 1203

Classes 1204
THTTPSoapCppInvoker Class 1204
THTTPSoapPascalInvoker Class 1205
THTTPSoapPascalInvoker Members 1206
THTTPSoapPascalInvoker Methods 1207

SOAPHTTPTrans Namespace 1209

Classes 1210
ESOAPHTTPException Class 1210
ESOAPHTTPException Members 1210
ESOAPHTTPException.Create Constructor 1212
ESOAPHTTPException.CreateFmt Constructor 1212
ESOAPHTTPException Properties 1213
THTTPReqResp Class 1214
THTTPReqResp Members 1215
THTTPReqResp.Create Constructor 1216
THTTPReqResp.Destroy Destructor 1216
THTTPReqResp Methods 1217
THTTPReqResp Properties 1225
THTTPReqResp Events 1232

Interfaces 1234
IHTTPReqResp Interface 1234
IHTTPReqResp Members 1234
IHTTPReqResp Methods 1234

Structs, Records, Enums 1236
SOAPHTTPTrans.SOAPInvokeOptions Enumeration 1236

Types 1237
SOAPHTTPTrans.TBeforePostEvent Type 1237

SOAPHTTPTrans.TPostingDataEvent Type 1237
SOAPHTTPTrans.TReceivingDataEvent Type 1238
SOAPHTTPTrans.TSOAPInvokeOptions Type 1238
SOAPHTTPTrans.TWinInetErrorEvent Type 1238

SOAPLinked Namespace 1239

Classes **1241**
ECustomException Class 1241
ECustomException Members 1241
ECustomException.Create Constructor 1243
ECustomException.Destroy Destructor 1245
ECustomException Methods 1246
TFTCompT Class 1247
TFTCompT Members 1247
TFTCompT Methods 1247
TFTCompT Events 1248
TFTCompTI Class 1249
TFTCompTI Members 1249
TFTCompTI Methods 1251
TLinkedRIO Class 1252
TLinkedRIO Members 1253
TLinkedRIO.Create Constructor 1253
TLinkedRIO.CreateFile Constructor 1254
TLinkedRIO.Destroy Destructor 1254
TLinkedRIO Methods 1255
TLinkedRIO Properties 1256
TLinkedWebNode Class 1257
TLinkedWebNode Members 1257
TLinkedWebNode.Create Constructor 1258
TLinkedWebNode.Destroy Destructor 1258
TLinkedWebNode Methods 1259
TLinkedWebNode Properties 1261
TLogLinkedWebNode Class 1262
TLogLinkedWebNode Members 1263
TLogLinkedWebNode.Create Constructor 1263
TLogLinkedWebNode.Destroy Destructor 1264
TLogLinkedWebNode Methods 1264
TLogLinkedWebNode Properties 1265

TWinHandleUser Class 1266
TWinHandleUser Members 1266
TWinHandleUser Methods 1267

Interfaces 1268
IIntegerTest Interface 1268
IIntegerTest Members 1268
IIntegerTest Methods 1268
IObjectGetter Interface 1269
IObjectGetter Members 1269
IObjectGetter Methods 1270
IStringTest Interface 1270
IStringTest Members 1270
IStringTest Methods 1271

Structs, Records, Enums 1272
SOAPLinked.EventKind Enumeration 1272

Types 1273
SOAPLinked.TCStringEvent Type 1273
SOAPLinked.TCustExCallback Type 1273
SOAPLinked.TStringEvent Type 1274
SOAPLinked.TVStringEvent Type 1274

SOAPMidas Namespace 1275

Interfaces 1276
IAppServerSOAP Interface 1276
IAppServerSOAP Members 1276
IAppServerSOAP Methods 1277

Functions 1281
SOAPMidas.RegDefIAppServerInvClass Function 1281

SOAPPasInv Namespace 1283

Classes 1284
TSoapPascalInvoker Class 1284
TSoapPascalInvoker Members 1284
TSoapPascalInvoker.Create Constructor 1285
TSoapPascalInvoker.Destroy Destructor 1286
TSoapPascalInvoker Fields 1286

TSoapPascalInvoker Methods 1287
TSoapPascalInvoker Properties 1289
TSoapPascalInvoker Events 1290

Interfaces 1293
IAdapterConverter Interface 1293
IAdapterConverter Members 1293
IAdapterConverter Methods 1293
IInvokerConverter Interface 1294
IInvokerConverter Members 1294
IInvokerConverter Methods 1294

Functions 1296
SOAPPasInv.GetSOAPHeaders Function 1296

Types 1297
SOAPPasInv.TAfterDispatchEvent Type 1297
SOAPPasInv.TBeforeDispatchEvent Type 1297
SOAPPasInv.TBeforeDispatchEvent2 Type 1298
SOAPPasInv.TOnExceptionEvent Type 1298

Sockets Namespace 1299

Classes 1301
ESocketError Class 1301
TBaseSocket Class 1302
TBaseSocket Members 1302
TBaseSocket.Create Constructor 1303
TBaseSocket.Destroy Destructor 1304
TBaseSocket Methods 1304
TBaseSocket Properties 1313
TBaseSocket Events 1317
TClientSocketThread Class 1322
TClientSocketThread Members 1322
TClientSocketThread.Create Constructor 1323
TClientSocketThread.Destroy Destructor 1323
TClientSocketThread Methods 1324
TClientSocketThread Properties 1325
TCustomIpClient Class 1326
TCustomIpClient Members 1326
TCustomIpClient.Create Constructor 1328

TCustomIpClient Methods 1328
TCustomIpClient Properties 1331
TCustomIpClient Events 1331
TCustomTcpServer Class 1332
TCustomTcpServer Members 1333
TCustomTcpServer.Create Constructor 1334
TCustomTcpServer.Destroy Destructor 1335
TCustomTcpServer Methods 1335
TCustomTcpServer Properties 1338
TCustomTcpServer Events 1339
TIpSocket Class 1341
TIpSocket Members 1342
TIpSocket.Create Constructor 1343
TIpSocket Methods 1343
TIpSocket Properties 1348
TRawSocket Class 1353
TRawSocket Members 1353
TRawSocket.Create Constructor 1355
TRawSocket Properties 1355
TServerSocketThread Class 1356
TServerSocketThread Members 1356
TServerSocketThread.Create Constructor 1357
TServerSocketThread.Destroy Destructor 1357
TServerSocketThread Methods 1358
TServerSocketThread Properties 1360
TServerSocketThread Events 1361
TTcpClient Class 1362
TTcpClient Members 1363
TTcpClient Properties 1365
TTcpClient Events 1369
TTcpServer Class 1375
TTcpServer Members 1377
TTcpServer Properties 1378
TTcpServer Events 1382
TUdpSocket Class 1387
TUdpSocket Members 1388
TUdpSocket.Create Constructor 1389
TUdpSocket Properties 1390
TUdpSocket Events 1396

Structs, Records, Enums **1403**
Sockets.TIPHeader Record 1403
Sockets.TServerSocketBlockMode Enumeration 1404
Sockets.TSocketDomain Enumeration 1404
Sockets.TSocketType Enumeration 1406

Types **1408**
Sockets.TGetThreadEvent Type 1408
Sockets.TSocketAcceptEvent Type 1408
Sockets.TSocketBlockMode Type 1409
Sockets.TSocketDataEvent Type 1409
Sockets.TSocketErrorEvent Type 1409
Sockets.TSocketHost Type 1410
Sockets.TSocketNotifyEvent Type 1410
Sockets.TSocketPort Type 1410
Sockets.TSocketProtocol Type 1411

Constants **1412**
Sockets.CRLF Constant 1412

SqlExpr Namespace 1413

Classes **1416**
TConnectionData Class 1416
TConnectionData Members 1417
Create Constructor 1417
TConnectionData.Destroy Destructor 1418
TConnectionData Methods 1418
TConnectionData Properties 1420
TCustomSQLDataSet Class 1421
TCustomSQLDataSet Members 1422
TCustomSQLDataSet.Create Constructor 1425
TCustomSQLDataSet.Destroy Destructor 1426
TCustomSQLDataSet Methods 1426
TCustomSQLDataSet Properties 1452
TCustomSQLDataSet Events 1468
TFLDDesc Class 1471
TFLDDesc Members 1471
TFLDDesc Methods 1472
TFLDDesc Properties 1478

TISQLCommand Class 1481
TISQLCommand Members 1481
TISQLCommand.Destroy Destructor 1482
TISQLCommand Methods 1482
TISQLConnection Class 1486
TISQLConnection Members 1487
TISQLConnection.Create Constructor 1487
TISQLConnection Methods 1487
TISQLCursor Class 1492
TISQLCursor Members 1492
TISQLCursor Methods 1493
TISQLMetaData Class 1502
TISQLMetaData Members 1503
TISQLMetaData Methods 1503
TSQLBlobStream Class 1507
TSQLBlobStream Members 1508
TSQLBlobStream.Create Constructor 1508
TSQLBlobStream.Destroy Destructor 1508
TSQLBlobStream Methods 1509
TSQLConnection Class 1510
TSQLConnection Members 1511
TSQLConnection.Create Constructor 1513
TSQLConnection.Destroy Destructor 1513
TSQLConnection Methods 1513
TSQLConnection Properties 1531
TSQLConnection Events 1550
TSQLDataLink Class 1553
TSQLDataLink Members 1553
TSQLDataLink.Create Constructor 1554
TSQLDataLink Methods 1554
TSQLDataSet Class 1556
TSQLDataSet Members 1557
TSQLDataSet.Create Constructor 1560
TSQLDataSet Methods 1560
TSQLDataSet Properties 1561
TSQLMonitor Class 1567
TSQLMonitor Members 1567
TSQLMonitor.Create Constructor 1568
TSQLMonitor.Destroy Destructor 1569

TSQLMonitor Methods 1569
TSQLMonitor Properties 1572
TSQLQuery Class 1577
TSQLQuery Members 1578
TSQLQuery.Create Constructor 1581
TSQLQuery.Destroy Destructor 1581
TSQLQuery Methods 1582
TSQLQuery Properties 1583
TSqlServerMethod Class 1589
TSqlServerMethod Members 1589
TSqlServerMethod.Create Constructor 1592
TSqlServerMethod Methods 1592
TSqlServerMethod Properties 1593
TSQLStoredProc Class 1594
TSQLStoredProc Members 1595
TSQLStoredProc.Create Constructor 1598
TSQLStoredProc Methods 1599
TSQLStoredProc Properties 1600
TSQLTable Class 1604
TSQLTable Members 1605
TSQLTable.Create Constructor 1608
TSQLTable.Destroy Destructor 1608
TSQLTable Methods 1609
TSQLTable Properties 1611
TTransactionItem Class 1618
TTransactionItem Members 1618
TTransactionItem Fields 1618

Functions 1620

SqlExpr.FreeProcParams Function 1620
SqlExpr.GetConnectionNames Function 1620
SqlExpr.GetConnectionRegistryFile Function 1621
SqlExpr.GetDelegateDriverNames Function 1621
SqlExpr.GetDriverNames Function 1621
SqlExpr.GetDriverRegistryFile Function 1622
SqlExpr.LoadParamListItems Function 1622
SqlExpr.RegisterDbXpressLib Function 1623

Structs, Records, Enums 1624

SqlExpr.EConnectFlag Enumeration 1624

SqlExpr.SPParamDesc Record 1625
SqlExpr.SQLTRACEDesc25 Record 1626
SqlExpr.SQLTRACEDesc30 Record 1626
SqlExpr.TConnectionState Enumeration 1627
SqlExpr.TConnectionUserType Enumeration 1627
SqlExpr.TSchemaType Enumeration 1628
SqlExpr.TSQLExceptionType Enumeration 1629
SqlExpr.TSQLSchemaInfo Record 1630
SqlExpr.TTableScope Enumeration 1630
SqlExpr.TTransactionDesc Record 1631
SqlExpr.TTransIsolationLevel Enumeration 1631

Types **1633**

SqlExpr.PSPParamDesc Type 1633
SqlExpr.pSQLTRACEDesc Type 1634
SqlExpr.pSQLTRACEDesc25 Type 1634
SqlExpr.pSQLTRACEDesc30 Type 1634
SqlExpr.SQLResult Type 1635
SqlExpr.SQLSPParamDesc Type 1636
SqlExpr.SQLTRACEDesc Type 1636
SqlExpr.TConnectChangeEvent Type 1636
SqlExpr.TFieldDescList Type 1637
SqlExpr.TFieldList Type 1637
SqlExpr.TFLDDescRef Type 1637
SqlExpr.TGetDriverFunc Type 1638
SqlExpr.TLocale Type 1638
SqlExpr.TLocaleCode Type 1638
SqlExpr.TParseInsertSqlEvent Type 1639
SqlExpr.TParseSqlEvent Type 1639
SqlExpr.TSQLConnectionLoginEvent Type 1639
SqlExpr.TTableScopes Type 1640
SqlExpr.TTraceEvent Type 1640
SqlExpr.TTraceEvent25 Type 1640
SqlExpr.TTraceEvent30 Type 1641
SqlExpr.TTraceLogEvent Type 1641
SqlExpr.TTraceLogEvent25 Type 1641
SqlExpr.TTraceLogEvent30 Type 1642

Variables **1643**

SqlExpr.DllGetClassObject Variable 1643

Constants **1644**
SqlExpr.BlobTypeMap Constant 1644
SqlExpr.DataTypeMap Constant 1645
SqlExpr.DefaultCursor Constant 1645
SqlExpr.DefaultMaxBlobSize Constant 1646
SqlExpr.DefaultRowsetSize Constant 1646
SqlExpr.eSQLAutoIncr Constant 1646
SqlExpr.eSQLDefault Constant 1647
SqlExpr.eSQLFunction Constant 1647
SqlExpr.eSQLLocal Constant 1647
SqlExpr.eSQLNonUnique Constant 1648
SqlExpr.eSQLPackage Constant 1648
SqlExpr.eSQLPrimaryKey Constant 1648
SqlExpr.eSQLProcedure Constant 1649
SqlExpr.eSQLRowId Constant 1649
SqlExpr.eSQLRowVersion Constant 1649
SqlExpr.eSQLSynonym Constant 1650
SqlExpr.eSQLSysProcedure Constant 1650
SqlExpr.eSQLSystemTable Constant 1650
SqlExpr.eSQLTable Constant 1651
SqlExpr.eSQLTempTable Constant 1651
SqlExpr.eSQLUnique Constant 1651
SqlExpr.eSQLView Constant 1652
SqlExpr.FldSubTypeMap Constant 1652
SqlExpr.FldTypeMap Constant 1652
SqlExpr.HourGlassCursor Constant 1653
SqlExpr.SAnd Constant 1653
SqlExpr.SOrderBy Constant 1653
SqlExpr.SParam Constant 1654
SqlExpr.SSelect Constant 1654
SqlExpr.SSelectSpaces Constant 1654
SqlExpr.SSelectStar Constant 1655
SqlExpr.SSelectStarFrom Constant 1655
SqlExpr.SUB_TYPE_MEMO Constant 1655
SqlExpr.SWhere Constant 1656
SqlExpr.TErrorMessageSize Constant 1656

SqlTimSt Namespace **1657**

Classes **1659**
TTimeZone Record 1659
TTimeZone Members 1659
TTimeZone Fields 1659
TTimeZone Methods 1660

Functions **1661**
SqlTimSt.CheckSqlTimeStamp Function 1661
SqlTimSt.DateTimeToSQLTimeStamp Function 1662
SqlTimSt.LocalToUTC Function 1662
SqlTimSt.SQLDayOfWeek Function 1662
SqlTimSt.SQLTimeStampToDateTime Function 1663
SqlTimSt.SQLTimeStampToStr Function 1663
SqlTimSt.StrToSQLTimeStamp Function 1664
SqlTimSt.TryStrToSQLTimeStamp Function 1665
SqlTimSt.UTCToLocal Function 1665
SqlTimSt.VarIsSQLTimeStamp Function 1666
SqlTimSt.VarSQLTimeStamp Function 1666
SqlTimSt.VarSQLTimeStampCreate Function 1667
SqlTimSt.VarToSQLTimeStamp Function 1668

Structs, Records, Enums **1669**
SqlTimSt.TSQLTimeStamp Record 1669

Types **1671**
SqlTimSt.PSQLTimeStamp Type 1671

Constants **1672**
SqlTimSt.NullSQLTimeStamp Constant 1672

StdActnMenus Namespace 1673

Classes **1674**
TStandardAddRemoveItem Class 1674
TStandardAddRemoveItem Members 1674
TStandardAddRemoveItem Methods 1675
TStandardButtonControl Class 1676
TStandardButtonControl Members 1676
TStandardButtonControl Methods 1676
TStandardCustomizePopup Class 1677
TStandardCustomizePopup Members 1677

TStandardCustomizePopup Methods 1678
TStandardDropDownButton Class 1679
TStandardDropDownButton Members 1679
TStandardDropDownButton Methods 1679
TStandardMenuButton Class 1680
TStandardMenuButton Members 1681
TStandardMenuButton Methods 1681
TStandardMenuExpandBtn Class 1682
TStandardMenuExpandBtn Members 1682
TStandardMenuExpandBtn Methods 1683
TStandardMenuItem Class 1683
TStandardMenuItem Members 1684
TStandardMenuItem Methods 1684
TStandardMenuPopup Class 1688
TStandardMenuPopup Members 1688
TStandardMenuPopup Methods 1688
TStandardToolScrollBtn Class 1690
TStandardToolScrollBtn Members 1690
TStandardToolScrollBtn Methods 1690

Functions **1692**
StdActnMenus.RegisterStandardMenus Function 1692

StdActns Namespace 1693

Classes **1695**
TBrowseForFolder Class 1696
TBrowseForFolder Members 1696
TBrowseForFolder Methods 1697
TBrowseForFolder Properties 1698
TBrowseForFolder Events 1704
TColorSelect Class 1706
TColorSelect Members 1706
TColorSelect Methods 1707
TColorSelect Properties 1707
TColorSelect Events 1712
TCommonDialogAction Class 1715
TCommonDialogAction Members 1716
TCommonDialogAction.Create Constructor 1716
TCommonDialogAction.Destroy Destructor 1717

TCommonDialogAction Fields 1717
TCommonDialogAction Methods 1717
TCommonDialogAction Properties 1720
TCommonDialogAction Events 1721
TEditAction Class 1722
TEditAction Members 1723
TEditAction.Destroy Destructor 1723
TEditAction Methods 1723
TEditAction Properties 1725
TEditCopy Class 1726
TEditCopy Members 1727
TEditCopy Methods 1727
TEditCut Class 1728
TEditCut Members 1729
TEditCut Methods 1729
TEditDelete Class 1730
TEditDelete Members 1730
TEditDelete Methods 1731
TEditPaste Class 1732
TEditPaste Members 1733
TEditPaste Methods 1733
TEditSelectAll Class 1734
TEditSelectAll Members 1735
TEditSelectAll Methods 1735
TEditUndo Class 1736
TEditUndo Members 1737
TEditUndo Methods 1738
TFileAction Class 1739
TFileAction Members 1739
TFileAction Methods 1740
TFileAction Properties 1740
TFileExit Class 1741
TFileExit Members 1742
TFileExit Methods 1742
TFileExit Properties 1743
TFileExit Events 1747
TFileOpen Class 1749
TFileOpen Members 1750
TFileOpen.Create Constructor 1751

TFileOpen Methods 1751
TFileOpen Properties 1752
TFileOpen Events 1757
TFileOpenWith Class 1760
TFileOpenWith Members 1761
TFileOpenWith Methods 1762
TFileOpenWith Properties 1763
TFileOpenWith Events 1765
TFilePageSetup Class 1767
TFilePageSetup Members 1767
TFilePageSetup Methods 1768
TFilePageSetup Properties 1769
TFilePageSetup Events 1773
TFilePrintSetup Class 1776
TFilePrintSetup Members 1777
TFilePrintSetup Methods 1778
TFilePrintSetup Properties 1778
TFilePrintSetup Events 1783
TFileSaveAs Class 1785
TFileSaveAs Members 1786
TFileSaveAs.Create Constructor 1787
TFileSaveAs Methods 1787
TFileSaveAs Properties 1788
TFileSaveAs Events 1791
TFontEdit Class 1794
TFontEdit Members 1794
TFontEdit Methods 1795
TFontEdit Properties 1796
TFontEdit Events 1800
THelpAction Class 1803
THelpAction Members 1804
THelpAction.Create Constructor 1804
THelpAction Methods 1804
THelpContents Class 1805
THelpContents Members 1806
THelpContents Methods 1806
THelpContextAction Class 1806
THelpContextAction Members 1807
THelpContextAction Methods 1807

THelpOnHelp Class 1808
THelpOnHelp Members 1809
THelpOnHelp Methods 1809
THelpTopicSearch Class 1810
THelpTopicSearch Members 1810
THelpTopicSearch Methods 1811
THintAction Class 1811
THintAction Members 1812
THintAction.Create Constructor 1812
THintAction Properties 1813
TPrintDlg Class 1814
TPrintDlg Members 1814
TPrintDlg Methods 1815
TPrintDlg Properties 1815
TPrintDlg Events 1820
TSearchAction Class 1823
TSearchAction Members 1824
TSearchAction.Create Constructor 1824
TSearchAction.Destroy Destructor 1825
TSearchAction Fields 1825
TSearchAction Methods 1826
TSearchFind Class 1828
TSearchFind Members 1829
TSearchFind Methods 1830
TSearchFind Properties 1831
TSearchFind Events 1835
TSearchFindFirst Class 1838
TSearchFindFirst Members 1839
TSearchFindFirst.Create Constructor 1840
TSearchFindNext Class 1840
TSearchFindNext Members 1841
TSearchFindNext.Create Constructor 1842
TSearchFindNext Methods 1842
TSearchFindNext Properties 1843
TSearchFindNext Events 1848
TSearchReplace Class 1850
TSearchReplace Members 1850
TSearchReplace Methods 1852
TSearchReplace Properties 1853

TSearchReplace Events 1857
TWindowAction Class 1860
TWindowAction Members 1861
TWindowAction.Create Constructor 1861
TWindowAction Methods 1861
TWindowAction Properties 1863
TWindowArrange Class 1864
TWindowArrange Members 1864
TWindowArrange Methods 1865
TWindowCascade Class 1865
TWindowCascade Members 1866
TWindowCascade Methods 1866
TWindowClose Class 1867
TWindowClose Members 1868
TWindowClose Methods 1868
TWindowMinimizeAll Class 1869
TWindowMinimizeAll Members 1870
TWindowMinimizeAll Methods 1870
TWindowTileHorizontal Class 1871
TWindowTileHorizontal Members 1871
TWindowTileHorizontal Methods 1872
TWindowTileVertical Class 1873
TWindowTileVertical Members 1873
TWindowTileVertical Methods 1874

Structs, Records, Enums **1875**
StdActns.TBrowseForFolderOption Enumeration 1875

Types **1877**
StdActns.TBrowseForFolderCallbackEvent Type 1877
StdActns.TBrowseOptions Type 1877
StdActns.TCommonDialogClass Type 1878

StdConvs Namespace 1879

Functions **1884**
StdConvs.CelsiusToFahrenheit Function 1884
StdConvs.FahrenheitToCelsius Function 1884
StdConvs.InitStdConvs Function 1885

Variables **1886**

StdConvs.auAcres Variable 1888
StdConvs.auAres Variable 1889
StdConvs.auCentares Variable 1889
StdConvs.auHectares Variable 1889
StdConvs.auSquareCentimeters Variable 1890
StdConvs.auSquareDecameters Variable 1890
StdConvs.auSquareDecimeters Variable 1890
StdConvs.auSquareFeet Variable 1891
StdConvs.auSquareHectometers Variable 1891
StdConvs.auSquareInches Variable 1891
StdConvs.auSquareKilometers Variable 1892
StdConvs.auSquareMeters Variable 1892
StdConvs.auSquareMiles Variable 1892
StdConvs.auSquareMillimeters Variable 1893
StdConvs.auSquareRods Variable 1893
StdConvs.auSquareYards Variable 1893
StdConvs.cbArea Variable 1894
StdConvs.cbDistance Variable 1894
StdConvs.cbMass Variable 1895
StdConvs.cbTemperature Variable 1895
StdConvs.cbTime Variable 1895
StdConvs.cbVolume Variable 1896
StdConvs.duAngstroms Variable 1897
StdConvs.duAstronomicalUnits Variable 1897
StdConvs.duCentimeters Variable 1898
StdConvs.duChains Variable 1898
StdConvs.duCubits Variable 1898
StdConvs.duDecameters Variable 1899
StdConvs.duDecimeters Variable 1899
StdConvs.duFathoms Variable 1899
StdConvs.duFeet Variable 1900
StdConvs.duFurlongs Variable 1900
StdConvs.duGigameters Variable 1900
StdConvs.duHands Variable 1901
StdConvs.duHectometers Variable 1901
StdConvs.duInches Variable 1901
StdConvs.duKilometers Variable 1902
StdConvs.duLightYears Variable 1902
StdConvs.duLinks Variable 1902

StdConvs.duMegameters Variable 1903
StdConvs.duMeters Variable 1903
StdConvs.duMicromicrons Variable 1903
StdConvs.duMicrons Variable 1904
StdConvs.duMiles Variable 1904
StdConvs.duMillimeters Variable 1904
StdConvs.duMillimicrons Variable 1905
StdConvs.duNauticalMiles Variable 1905
StdConvs.duPaces Variable 1905
StdConvs.duParsecs Variable 1906
StdConvs.duPicas Variable 1906
StdConvs.duPoints Variable 1906
StdConvs.duRods Variable 1907
StdConvs.duYards Variable 1907
StdConvs.muCentigrams Variable 1907
StdConvs.muDecagrams Variable 1908
StdConvs.muDecigrams Variable 1908
StdConvs.muDrams Variable 1908
StdConvs.muGrains Variable 1909
StdConvs.muGrams Variable 1909
StdConvs.muHectograms Variable 1909
StdConvs.muKilograms Variable 1910
StdConvs.muLongTons Variable 1910
StdConvs.muMetricTons Variable 1910
StdConvs.muMicrograms Variable 1911
StdConvs.muMilligrams Variable 1911
StdConvs.muNanograms Variable 1911
StdConvs.muOunces Variable 1912
StdConvs.muPounds Variable 1912
StdConvs.muStones Variable 1912
StdConvs.muTons Variable 1913
StdConvs.tuCelsius Variable 1913
StdConvs.tuCenturies Variable 1913
StdConvs.tuDateTime Variable 1914
StdConvs.tuDays Variable 1914
StdConvs.tuDecades Variable 1914
StdConvs.tuFahrenheit Variable 1915
StdConvs.tuFortnights Variable 1915
StdConvs.tuHours Variable 1915

StdConvs.tuJulianDate Variable 1916
StdConvs.tuKelvin Variable 1916
StdConvs.tuMillennia Variable 1916
StdConvs.tuMilliSeconds Variable 1917
StdConvs.tuMinutes Variable 1917
StdConvs.tuModifiedJulianDate Variable 1917
StdConvs.tuMonths Variable 1918
StdConvs.tuRankine Variable 1918
StdConvs.tuReaumur Variable 1918
StdConvs.tuSeconds Variable 1919
StdConvs.tuWeeks Variable 1919
StdConvs.tuYears Variable 1919
StdConvs.vuAcreFeet Variable 1920
StdConvs.vuAcreInches Variable 1920
StdConvs.vuCentiLiters Variable 1920
StdConvs.vuCordFeet Variable 1921
StdConvs.vuCords Variable 1921
StdConvs.vuCubicCentimeters Variable 1921
StdConvs.vuCubicDecameters Variable 1922
StdConvs.vuCubicDecimeters Variable 1922
StdConvs.vuCubicFeet Variable 1922
StdConvs.vuCubicHectometers Variable 1923
StdConvs.vuCubicInches Variable 1923
StdConvs.vuCubicKilometers Variable 1923
StdConvs.vuCubicMeters Variable 1924
StdConvs.vuCubicMiles Variable 1924
StdConvs.vuCubicMillimeters Variable 1924
StdConvs.vuCubicYards Variable 1925
StdConvs.vuDecaLiters Variable 1925
StdConvs.vuDecasteres Variable 1925
StdConvs.vuDeciLiters Variable 1926
StdConvs.vuDecisteres Variable 1926
StdConvs.vuDryBuckets Variable 1926
StdConvs.vuDryBushels Variable 1927
StdConvs.vuDryGallons Variable 1927
StdConvs.vuDryPecks Variable 1927
StdConvs.vuDryPints Variable 1928
StdConvs.vuDryQuarts Variable 1928
StdConvs.vuFluidCups Variable 1928

StdConvs.vuFluidGallons Variable 1929
StdConvs.vuFluidGills Variable 1929
StdConvs.vuFluidOunces Variable 1929
StdConvs.vuFluidPints Variable 1930
StdConvs.vuFluidQuarts Variable 1930
StdConvs.vuFluidTablespoons Variable 1930
StdConvs.vuFluidTeaspoons Variable 1931
StdConvs.vuHectoLiters Variable 1931
StdConvs.vuKiloLiters Variable 1931
StdConvs.vuLiters Variable 1932
StdConvs.vuMilliLiters Variable 1932
StdConvs.vuSteres Variable 1932
StdConvs.vuUKBuckets Variable 1933
StdConvs.vuUKBushels Variable 1933
StdConvs.vuUKGallons Variable 1933
StdConvs.vuUKGills Variable 1934
StdConvs.vuUKOunces Variable 1934
StdConvs.vuUKPecks Variable 1934
StdConvs.vuUKPints Variable 1935
StdConvs.vuUKPottles Variable 1935
StdConvs.vuUKQuarts Variable 1935

Constants 1936

StdConvs.CubicMetersPerAcreFoot Constant 1937
StdConvs.CubicMetersPerAcreInch Constant 1937
StdConvs.CubicMetersPerCord Constant 1938
StdConvs.CubicMetersPerCordFoot Constant 1938
StdConvs.CubicMetersPerCubicFoot Constant 1938
StdConvs.CubicMetersPerCubicInch Constant 1939
StdConvs.CubicMetersPerCubicMile Constant 1939
StdConvs.CubicMetersPerCubicYard Constant 1939
StdConvs.CubicMetersPerUKBucket Constant 1940
StdConvs.CubicMetersPerUKBushel Constant 1940
StdConvs.CubicMetersPerUKGallon Constant 1940
StdConvs.CubicMetersPerUKGill Constant 1941
StdConvs.CubicMetersPerUKOunce Constant 1941
StdConvs.CubicMetersPerUKPeck Constant 1941
StdConvs.CubicMetersPerUKPint Constant 1942
StdConvs.CubicMetersPerUKPottle Constant 1942
StdConvs.CubicMetersPerUKQuart Constant 1942

StdConvs.CubicMetersPerUSDryBucket Constant 1943
StdConvs.CubicMetersPerUSDryBushel Constant 1943
StdConvs.CubicMetersPerUSDryGallon Constant 1943
StdConvs.CubicMetersPerUSDryPeck Constant 1944
StdConvs.CubicMetersPerUSDryPint Constant 1944
StdConvs.CubicMetersPerUSDryQuart Constant 1944
StdConvs.CubicMetersPerUSFluidCup Constant 1945
StdConvs.CubicMetersPerUSFluidGallon Constant 1945
StdConvs.CubicMetersPerUSFluidGill Constant 1945
StdConvs.CubicMetersPerUSFluidOunce Constant 1946
StdConvs.CubicMetersPerUSFluidPint Constant 1946
StdConvs.CubicMetersPerUSFluidQuart Constant 1946
StdConvs.CubicMetersPerUSFluidTablespoon Constant 1947
StdConvs.CubicMetersPerUSFluidTeaspoon Constant 1947
StdConvs.GramsPerDrams Constant 1947
StdConvs.GramsPerGrains Constant 1948
StdConvs.GramsPerLongTons Constant 1948
StdConvs.GramsPerOunces Constant 1948
StdConvs.GramsPerPound Constant 1949
StdConvs.GramsPerStones Constant 1949
StdConvs.GramsPerTons Constant 1949
StdConvs.MetersPerAstronomicalUnit Constant 1950
StdConvs.MetersPerChain Constant 1950
StdConvs.MetersPerCubit Constant 1950
StdConvs.MetersPerFathom Constant 1951
StdConvs.MetersPerFoot Constant 1951
StdConvs.MetersPerFurlong Constant 1951
StdConvs.MetersPerHand Constant 1952
StdConvs.MetersPerInch Constant 1952
StdConvs.MetersPerLightSecond Constant 1952
StdConvs.MetersPerLightYear Constant 1953
StdConvs.MetersPerLink Constant 1953
StdConvs.MetersPerMile Constant 1953
StdConvs.MetersPerNauticalMiles Constant 1954
StdConvs.MetersPerPace Constant 1954
StdConvs.MetersPerParsec Constant 1954
StdConvs.MetersPerPica Constant 1955
StdConvs.MetersPerPoint Constant 1955
StdConvs.MetersPerRod Constant 1955

StdConvs.MetersPerYard Constant 1956
StdConvs.SquareMetersPerAcre Constant 1956
StdConvs.SquareMetersPerSquareFoot Constant 1956
StdConvs.SquareMetersPerSquareInch Constant 1957
StdConvs.SquareMetersPerSquareMile Constant 1957
StdConvs.SquareMetersPerSquareRod Constant 1957
StdConvs.SquareMetersPerSquareYard Constant 1958

StdCtrls Namespace 1959

Classes 1961
TButton Class 1961
TButton Members 1962
TButton Properties 1964
TButton Events 1998
TButtonActionLink Class 2030
TButtonActionLink Members 2031
TButtonActionLink Fields 2031
TButtonActionLink Methods 2031
TButtonControl Class 2033
TButtonControl Members 2033
TButtonControl.Create Constructor 2034
TButtonControl Methods 2034
TButtonControl Properties 2036
TCheckBox Class 2040
TCheckBox Members 2040
TCheckBox Properties 2042
TCheckBox Events 2074
TComboBox Class 2106
TComboBox Members 2108
TComboBox Properties 2111
TComboBox Events 2149
TCustomButton Class 2179
TCustomButton Members 2179
TCustomButton.Create Constructor 2180
TCustomButton.Destroy Destructor 2181
TCustomButton Enumerations 2181
TCustomButton Methods 2182
TCustomButton Properties 2186

TCustomButton Events 2195
TCustomCheckBox Class 2196
TCustomCheckBox Members 2196
TCustomCheckBox.Create Constructor 2197
TCustomCheckBox Methods 2197
TCustomCheckBox Properties 2200
TCustomCombo Class 2203
TCustomCombo Members 2203
TCustomCombo.Create Constructor 2205
TCustomCombo.Destroy Destructor 2205
TCustomCombo Fields 2206
TCustomCombo Methods 2208
TCustomCombo Properties 2223
TCustomCombo Events 2236
TCustomComboBox Class 2241
TCustomComboBox Members 2242
TCustomComboBox.Create Constructor 2244
TCustomComboBox.Destroy Destructor 2244
TCustomComboBox Methods 2245
TCustomComboBox Properties 2250
TCustomComboBox Events 2256
TCustomComboBoxStrings Class 2257
TCustomComboBoxStrings Members 2258
TCustomComboBoxStrings.Create Constructor 2258
TCustomComboBoxStrings.Destroy Destructor 2259
TCustomComboBoxStrings Methods 2259
TCustomComboBoxStrings Properties 2263
TCustomEdit Class 2263
TCustomEdit Members 2264
TCustomEdit.Create Constructor 2265
TCustomEdit Methods 2266
TCustomEdit Properties 2282
TCustomEdit Events 2299
TCustomGroupBox Class 2299
TCustomGroupBox Members 2300
TCustomGroupBox.Create Constructor 2300
TCustomGroupBox Methods 2301
TCustomLabel Class 2302
TCustomLabel Members 2303

TCustomLabel.Create Constructor 2303
TCustomLabel Methods 2304
TCustomLabel Properties 2306
TCustomListBox Class 2316
TCustomListBox Members 2317
TCustomListBox.Create Constructor 2318
TCustomListBox.Destroy Destructor 2319
TCustomListBox Fields 2319
TCustomListBox Methods 2320
TCustomListBox Properties 2340
TCustomListBox Events 2355
TCustomMemo Class 2361
TCustomMemo Members 2361
TCustomMemo.Create Constructor 2363
TCustomMemo.Destroy Destructor 2364
TCustomMemo Methods 2364
TCustomMemo Properties 2368
TCustomStaticText Class 2375
TCustomStaticText Members 2375
TCustomStaticText.Create Constructor 2376
TCustomStaticText Methods 2376
TCustomStaticText Properties 2378
TEdit Class 2380
TEdit Members 2381
TEdit Properties 2384
TEdit Events 2418
TGroupBox Class 2451
TGroupBox Members 2451
TGroupBox Properties 2453
TGroupBox Events 2481
TImageMargins Class 2517
TImageMargins Members 2518
TImageMargins Methods 2518
TImageMargins Properties 2519
TImageMargins Events 2521
TLabel Class 2522
TLabel Members 2522
TLabel Properties 2524
TLabel Events 2551

TListBox Class 2572
TListBox Members 2572
TListBox Properties 2575
TListBox Events 2612
TMemo Class 2649
TMemo Members 2650
TMemo Properties 2653
TMemo Events 2689
TPushButtonActionLink Class 2722
TPushButtonActionLink Members 2723
TPushButtonActionLink Methods 2723
TRadioButton Class 2724
TRadioButton Members 2725
TRadioButton.Create Constructor 2726
TRadioButton Methods 2727
TRadioButton Properties 2729
TRadioButton Events 2758
TScrollBar Class 2790
TScrollBar Members 2791
TScrollBar.Create Constructor 2792
TScrollBar Methods 2792
TScrollBar Properties 2794
TScrollBar Events 2828
TStaticText Class 2853
TStaticText Members 2853
TStaticText Properties 2855
TStaticText Events 2884

Structs, Records, Enums 2906

StdCtrls.TCheckBoxState Enumeration 2906
StdCtrls.TComboBoxStyle Enumeration 2907
StdCtrls.TEditCharCase Enumeration 2907
StdCtrls.TEllipsisPosition Enumeration 2908
StdCtrls.TImageAlignment Enumeration 2909
StdCtrls.TListBoxStyle Enumeration 2910
StdCtrls.TScrollCode Enumeration 2911
StdCtrls.TScrollStyle Enumeration 2912
StdCtrls.TSelection Record 2913
StdCtrls.TStaticBorderStyle Enumeration 2913
StdCtrls.TTextLayout Enumeration 2914

Types **2915**
StdCtrls.TButtonActionLinkClass Type 2915
StdCtrls.TCustomComboBoxStringsClass Type 2915
StdCtrls.TDrawItemEvent Type 2916
StdCtrls.TFNDrawText Type 2916
StdCtrls.TLBFindDataEvent Type 2917
StdCtrls.TLBGetDataEvent Type 2917
StdCtrls.TLBGetDataObjectEvent Type 2918
StdCtrls.TMeasureItemEvent Type 2918
StdCtrls.TOwnerDrawState Type 2918
StdCtrls.TPushButtonActionLinkClass Type 2919
StdCtrls.TScrollEvent Type 2920

StdStyleActnCtrls Namespace 2921

Classes **2922**
TStandardStyleActionBars Class 2922
TStandardStyleActionBars Members 2922
TStandardStyleActionBars Methods 2922

Variables **2925**
StdStyleActnCtrls.StandardStyle Variable 2925

StdVCL Namespace 2927

Interfaces **2928**
IDataBroker Interface 2928
IDataBroker Members 2928
IDataBroker Methods 2928
IDataBrokerDisp Interface 2929
IDataBrokerDisp Members 2929
IDataBrokerDisp Methods 2929
IProvider Interface 2930
IProvider Members 2930
IProvider Methods 2931
IProvider Properties 2933
IProviderDisp Interface 2934
IProviderDisp Members 2934
IProviderDisp Methods 2934
IProviderDisp Properties 2936

IStrings Interface 2937
IStrings Members 2937
IStrings Methods 2937
IStrings Properties 2940
IStringsDisp Interface 2940
IStringsDisp Members 2941
IStringsDisp Methods 2941
IStringsDisp Properties 2942

Constants 2944
StdVCL.IID_IDataBroker Constant 2944
StdVCL.IID_IProvider Constant 2944
StdVCL.IID_IStrings Constant 2945
StdVCL.LIBID_StdVCL Constant 2945

StrHlpr Namespace 2947

Functions 2949
StrHlpr.AnsiAppend Function 2950
StrHlpr.AnsiAssign Function 2950
StrHlpr.AnsiCat Function 2950
StrHlpr.AnsiCopy Function 2951
StrHlpr.AnsiDelete Function 2951
StrHlpr.AnsiEqual Function 2951
StrHlpr.AnsiFree Function 2952
StrHlpr.AnsiFromPChar Function 2952
StrHlpr.AnsiFromPWideChar Function 2952
StrHlpr.AnsiFromUnicode Function 2953
StrHlpr.AnsiFromWide Function 2953
StrHlpr.AnsiInsert Function 2953
StrHlpr.AnsiPos Function 2954
StrHlpr.AnsiSetLength Function 2954
StrHlpr.UnicodeAppend Function 2954
StrHlpr.UnicodeAssign Function 2955
StrHlpr.UnicodeCat Function 2955
StrHlpr.UnicodeCopy Function 2955
StrHlpr.UnicodeDelete Function 2956
StrHlpr.UnicodeEqual Function 2956
StrHlpr.UnicodeFree Function 2956
StrHlpr.UnicodeFromAnsi Function 2957

StrHlpr.UnicodeFromPChar Function 2957
StrHlpr.UnicodeFromPWideChar Function 2957
StrHlpr.UnicodeFromWide Function 2958
StrHlpr.UnicodeGreater Function 2958
StrHlpr.UnicodeInsert Function 2958
StrHlpr.UnicodeLess Function 2959
StrHlpr.UnicodePos Function 2959
StrHlpr.UnicodeSetLength Function 2959
StrHlpr.WideAppend Function 2960
StrHlpr.WideAssign Function 2960
StrHlpr.WideCat Function 2960
StrHlpr.WideCopy Function 2961
StrHlpr.WideDelete Function 2961
StrHlpr.WideEqual Function 2961
StrHlpr.WideFree Function 2962
StrHlpr.WideFromAnsi Function 2962
StrHlpr.WideFromPChar Function 2962
StrHlpr.WideFromUnicode Function 2963
StrHlpr.WideGreater Function 2963
StrHlpr.WideInsert Function 2963
StrHlpr.WideLength Function 2964
StrHlpr.WideLess Function 2964
StrHlpr.WidePos Function 2964
StrHlpr.WideSetLength Function 2965

StrUtils Namespace 2967

Functions 2970

StrUtils.AnsiContainsStr Function 2971
StrUtils.AnsiContainsText Function 2972
StrUtils.AnsiEndsStr Function 2972
StrUtils.AnsiEndsText Function 2973
StrUtils.AnsiIndexStr Function 2973
StrUtils.AnsiIndexText Function 2974
StrUtils.AnsiLeftStr Function 2974
StrUtils.AnsiMatchStr Function 2975
StrUtils.AnsiMatchText Function 2975
StrUtils.AnsiMidStr Function 2976
StrUtils.AnsiReplaceStr Function 2977

StrUtils.AnsiReplaceText Function 2977
StrUtils.AnsiResemblesText Function 2978
StrUtils.AnsiReverseString Function 2978
StrUtils.AnsiRightStr Function 2979
StrUtils.AnsiStartsStr Function 2979
StrUtils.AnsiStartsText Function 2980
StrUtils.ContainsStr Function 2980
StrUtils.ContainsText Function 2981
StrUtils.DecodeSoundexInt Function 2981
StrUtils.DecodeSoundexWord Function 2982
StrUtils.DupeString Function 2982
StrUtils.EndsStr Function 2983
StrUtils.EndsText Function 2983
StrUtils.IfThen Function 2984
StrUtils.IndexStr Function 2984
StrUtils.IndexText Function 2985
StrUtils.LeftBStr Function 2985
StrUtils.LeftStr Function 2986
StrUtils.MatchStr Function 2987
StrUtils.MatchText Function 2987
StrUtils.MidBStr Function 2988
StrUtils.MidStr Function 2988
StrUtils.PosEx Function 2989
StrUtils.RandomFrom Function 2989
StrUtils.ReplaceStr Function 2990
StrUtils.ReplaceText Function 2990
StrUtils.ResemblesText Function 2991
StrUtils.ReverseString Function 2991
StrUtils.RightBStr Function 2992
StrUtils.RightStr Function 2992
StrUtils.SearchBuf Function 2993
StrUtils.Soundex Function 2994
StrUtils.SoundexCompare Function 2994
StrUtils.SoundexInt Function 2995
StrUtils.SoundexProc Function 2996
StrUtils.SoundexSimilar Function 2996
StrUtils.SoundexWord Function 2997
StrUtils.StartsStr Function 2998
StrUtils.StartsText Function 2998

StrUtils.StuffString Function 2999

Structs, Records, Enums **3000**

StrUtils.TStringSeachOption Enumeration 3000

Types **3001**

StrUtils.TCompareTextProc Type 3001

StrUtils.TSoundexIntLength Type 3001

StrUtils.TSoundexLength Type 3002

StrUtils.TStringSearchOptions Type 3002

Variables **3003**

StrUtils.AnsiResemblesProc Variable 3003

StrUtils.ResemblesProc Variable 3003

Constants **3005**

StrUtils.WordDelimiters Constant 3005

SvcMgr Namespace 3007

Classes **3009**

TDependencies Class 3009

TDependencies Members 3009

TDependencies.Create Constructor 3010

TDependencies Methods 3010

TDependencies Properties 3011

TDependency Class 3012

TDependency Members 3012

TDependency Methods 3013

TDependency Properties 3013

TEventLogger Class 3014

TEventLogger Members 3015

TEventLogger.Create Constructor 3015

TEventLogger.Destroy Destructor 3016

TEventLogger Methods 3016

TService Class 3017

TService Members 3017

TService.CreateNew Constructor 3019

TService.Destroy Destructor 3019

TService Methods 3019

TService Properties 3023

TService Events 3031

TServiceApplication Class 3035
TServiceApplication Members 3036
TServiceApplication.Create Constructor 3037
TServiceApplication.Destroy Destructor 3037
TServiceApplication Methods 3037
TServiceApplication Properties 3040
TServiceThread Class 3041
TServiceThread Members 3042
TServiceThread.Create Constructor 3042
TServiceThread Methods 3042

Structs, Records, Enums 3048
SvcMgr.TCurrentStatus Enumeration 3048
SvcMgr.TErrorSeverity Enumeration 3049
SvcMgr.TServiceType Enumeration 3049
SvcMgr.TStartType Enumeration 3050

Types 3052
SvcMgr.TContinueEvent Type 3052
SvcMgr.TPauseEvent Type 3052
SvcMgr.TServiceController Type 3053
SvcMgr.TServiceEvent Type 3053
SvcMgr.TStartEvent Type 3053
SvcMgr.TStopEvent Type 3054

Variables 3055
SvcMgr.Application Variable 3055

Constants 3056
SvcMgr.CM_SERVICE_CONTROL_CODE Constant 3056

SyncObjs Namespace 3057

Classes 3058
TConditionVariableCS Class 3058
TConditionVariableCS Members 3058
TConditionVariableCS.Create Constructor 3059
TConditionVariableCS Fields 3060
TConditionVariableCS Methods 3060
TConditionVariableHelper Class Helper 3062
TConditionVariableHelper Members 3062
TConditionVariableHelper Methods 3063

TConditionVariableMutex Class 3064
TConditionVariableMutex Members 3064
TConditionVariableMutex.Create Constructor 3065
TConditionVariableMutex.Destroy Destructor 3066
TConditionVariableMutex Methods 3066
TCriticalSection Class 3068
TCriticalSection Members 3068
TCriticalSection.Create Constructor 3069
TCriticalSection.Destroy Destructor 3069
TCriticalSection Fields 3070
TCriticalSection Methods 3070
TCriticalSectionHelper Class Helper 3072
TCriticalSectionHelper Members 3073
TCriticalSectionHelper Methods 3073
TEvent Class 3074
TEvent Members 3075
Create Constructor 3076
TEvent Methods 3077
THandleObject Class 3078
THandleObject Members 3079
THandleObject.Create Constructor 3080
THandleObject.Destroy Destructor 3080
THandleObject Fields 3081
THandleObject Methods 3081
THandleObject Properties 3082
TMutex Class 3083
TMutex Members 3083
Create Constructor 3084
TMutex Methods 3085
TSemaphore Class 3086
TSemaphore Members 3087
Create Constructor 3088
TSemaphore Methods 3089
TSimpleEvent Class 3090
TSynchroObject Class 3091
TSynchroObject Members 3091
TSynchroObject Methods 3092

Structs, Records, Enums 3094

SyncObjs.TWaitResult Enumeration 3094

Types **3095**
SyncObjs.THandleObjectArray Type 3095

SysConst Namespace 3097

Constants **3101**
SysConst.SAbstractError Constant 3104
SysConst.SAccessDenied Constant 3104
SysConst.SAccessViolationArg3 Constant 3104
SysConst.SAccessViolationNoArg Constant 3105
SysConst.SArgumentMissing Constant 3105
SysConst.SAssertError Constant 3105
SysConst.SAssertionFailed Constant 3106
SysConst.SByteIndexOutOfBounds Constant 3106
SysConst.SCannotCreateDir Constant 3106
SysConst.SCannotReadPackageInfo Constant 3107
SysConst.SCharIndexOutOfBounds Constant 3107
SysConst.SCodesetConversionError Constant 3107
SysConst.SControlC Constant 3108
SysConst.SDateEncodeError Constant 3108
SysConst.SDiskFull Constant 3108
SysConst.SDispatchError Constant 3109
SysConst.SDivByZero Constant 3109
SysConst.SDuplicatePackageUnit Constant 3109
SysConst.SEndOfFile Constant 3110
SysConst.sErrorLoadingPackage Constant 3110
SysConst.SException Constant 3110
SysConst.SExceptTitle Constant 3111
SysConst.SExternalException Constant 3111
SysConst.SFileNotFound Constant 3111
SysConst.SFormatTooLong Constant 3112
SysConst.SInOutError Constant 3112
SysConst.SIntfCastError Constant 3112
SysConst.SIntOverflow Constant 3113
SysConst.SInvalidBoolean Constant 3113
SysConst.SInvalidCast Constant 3113
SysConst.SInvalidCharCount Constant 3114
SysConst.SInvalidCodePage Constant 3114
SysConst.SInvalidCurrency Constant 3114

SysConst.SInvalidDate Constant 3115
SysConst.SInvalidDateTime Constant 3115
SysConst.SInvalidDateTimeFloat Constant 3115
SysConst.SInvalidDestinationArray Constant 3116
SysConst.SInvalidDestinationIndex Constant 3116
SysConst.SInvalidFilename Constant 3116
SysConst.SInvalidFloat Constant 3117
SysConst.SInvalidFormat Constant 3117
SysConst.SInvalidGUID Constant 3117
SysConst.SInvalidInput Constant 3118
SysConst.SInvalidInteger Constant 3118
SysConst.SInvalidOp Constant 3118
SysConst.SInvalidPackageFile Constant 3119
SysConst.SInvalidPackageHandle Constant 3119
SysConst.SInvalidPointer Constant 3119
SysConst.SInvalidSourceArray Constant 3120
SysConst.SInvalidTime Constant 3120
SysConst.SInvalidTimeStamp Constant 3120
SysConst.SInvalidVarCast Constant 3121
SysConst.SInvalidVarNullOp Constant 3121
SysConst.SInvalidVarOp Constant 3121
SysConst.SInvalidVarOpWithHResult Constant 3122
SysConst.SInvalidVarOpWithHResultWithPrefix Constant 3122
SysConst.SLongDayNameFri Constant 3122
SysConst.SLongDayNameMon Constant 3123
SysConst.SLongDayNameSat Constant 3123
SysConst.SLongDayNameSun Constant 3123
SysConst.SLongDayNameThu Constant 3124
SysConst.SLongDayNameTue Constant 3124
SysConst.SLongDayNameWed Constant 3124
SysConst.SLongMonthNameApr Constant 3125
SysConst.SLongMonthNameAug Constant 3125
SysConst.SLongMonthNameDec Constant 3125
SysConst.SLongMonthNameFeb Constant 3126
SysConst.SLongMonthNameJan Constant 3126
SysConst.SLongMonthNameJul Constant 3126
SysConst.SLongMonthNameJun Constant 3127
SysConst.SLongMonthNameMar Constant 3127
SysConst.SLongMonthNameMay Constant 3127

SysConst.SLongMonthNameNov Constant 3128
SysConst.SLongMonthNameOct Constant 3128
SysConst.SLongMonthNameSep Constant 3128
SysConst.SModuleAccessViolation Constant 3129
SysConst.SMonitorLockException Constant 3129
SysConst.SNL Constant 3129
SysConst.SNoMonitorSupportException Constant 3130
SysConst.SOperationAborted Constant 3130
SysConst.SOSError Constant 3130
SysConst.SOutOfMemory Constant 3131
SysConst.SOverflow Constant 3131
SysConst.SPrivilege Constant 3131
SysConst.SQuit Constant 3132
SysConst.SRangeError Constant 3132
SysConst.SReadAccess Constant 3132
SysConst.SResultTooLong Constant 3133
SysConst.SSafecallException Constant 3133
SysConst.SShortDayNameFri Constant 3133
SysConst.SShortDayNameMon Constant 3134
SysConst.SShortDayNameSat Constant 3134
SysConst.SShortDayNameSun Constant 3134
SysConst.SShortDayNameThu Constant 3135
SysConst.SShortDayNameTue Constant 3135
SysConst.SShortDayNameWed Constant 3135
SysConst.SShortMonthNameApr Constant 3136
SysConst.SShortMonthNameAug Constant 3136
SysConst.SShortMonthNameDec Constant 3136
SysConst.SShortMonthNameFeb Constant 3137
SysConst.SShortMonthNameJan Constant 3137
SysConst.SShortMonthNameJul Constant 3137
SysConst.SShortMonthNameJun Constant 3138
SysConst.SShortMonthNameMar Constant 3138
SysConst.SShortMonthNameMay Constant 3138
SysConst.SShortMonthNameNov Constant 3139
SysConst.SShortMonthNameOct Constant 3139
SysConst.SShortMonthNameSep Constant 3139
SysConst.SStackOverflow Constant 3140
SysConst.STimeEncodeError Constant 3140
SysConst.STooManyOpenFiles Constant 3140

SysConst.SUnderflow Constant 3141
SysConst.SUnknown Constant 3141
SysConst.sUnknownFromType Constant 3141
SysConst.sUnknownToType Constant 3142
SysConst.SUnkOSError Constant 3142
SysConst.SUnkWin32Error Constant 3142
SysConst.SVarArrayBounds Constant 3143
SysConst.SVarArrayCreate Constant 3143
SysConst.SVarArrayLocked Constant 3143
SysConst.SVarArrayWithHResult Constant 3144
SysConst.SVarBadType Constant 3144
SysConst.SVarDataCastToRecursing Constant 3144
SysConst.SVarDataClearRecursing Constant 3145
SysConst.SVarDataCopyNoIndRecursing Constant 3145
SysConst.SVarDataCopyRecursing Constant 3145
SysConst.SVarDataInitRecursing Constant 3146
SysConst.SVarInvalid Constant 3146
SysConst.SVarIsEmpty Constant 3146
SysConst.SVarNotArray Constant 3147
SysConst.SVarNotImplemented Constant 3147
SysConst.SVarOutOfMemory Constant 3147
SysConst.SVarOverflow Constant 3148
SysConst.SVarTypeAlreadyUsed Constant 3148
SysConst.SVarTypeAlreadyUsedWithPrefix Constant 3148
SysConst.SVarTypeConvertOverflow Constant 3149
SysConst.SVarTypeCouldNotConvert Constant 3149
SysConst.SVarTypeNotUsable Constant 3149
SysConst.SVarTypeNotUsableWithPrefix Constant 3150
SysConst.SVarTypeOutOfRange Constant 3150
SysConst.SVarTypeOutOfRangeWithPrefix Constant 3150
SysConst.SVarTypeRangeCheck1 Constant 3151
SysConst.SVarTypeRangeCheck2 Constant 3151
SysConst.SVarTypeTooManyCustom Constant 3151
SysConst.SVarTypeUnknown Constant 3152
SysConst.SVarUnexpected Constant 3152
SysConst.SWin32Error Constant 3152
SysConst.SWriteAccess Constant 3153
SysConst.SZeroDivide Constant 3153

SysInit Namespace 3155

Functions **3156**

SysInit._GetTls Function 3156
SysInit._InitExe Function 3156
SysInit._InitLib Function 3157
SysInit._InitPkg Function 3157
SysInit._PackageLoad Function 3157
SysInit._PackageUnload Function 3158
SysInit.VclExit Function 3158
SysInit.VclInit Function 3158

Variables **3159**

SysInit.CoverageLibraryName Variable 3159
SysInit.DataMark Variable 3159
SysInit.DllProc Variable 3160
SysInit.DllProcEx Variable 3160
SysInit.HInstance Variable 3160
SysInit.ModuleIsCpp Variable 3161
SysInit.ModuleIsLib Variable 3161
SysInit.ModuleIsPackage Variable 3161
SysInit.TlsIndex Variable 3162
SysInit.TlsLast Variable 3162

Constants **3163**

SysInit.PtrToNil Constant 3163

System Namespace 3165

Classes **3185**

AnsiStringBase Class 3185
AnsiStringBase Members 3187
AnsiStringBase::~AnsiStringBase Destructor 3189
AnsiStringBase Constructor 3189
AnsiStringBase Structures 3192
AnsiStringBase Enumerations 3193
AnsiStringBase Data Members 3194
AnsiStringBase Methods 3194
AnsiStringBase Operators 3215
AnsiStringT Template 3219

AnsiStringT Members 3220
AnsiStringT Constructor 3222
AnsiStringT Methods 3228
AnsiStringT Operators 3239
Currency Class 3242
Currency Members 3242
Currency Constructor 3244
Currency Operators 3245
Currency Friends 3260
CurrencyBase Class 3262
CurrencyBase Members 3263
CurrencyBase Data Members 3263
DelphiInterface Template 3263
DelphiInterface Members 3264
DelphiInterface::~DelphiInterface Destructor 3264
DelphiInterface Constructor 3265
DelphiInterface Methods 3265
DelphiInterface Operators 3266
DynamicArray Template 3268
DynamicArray Members 3270
DynamicArray::~DynamicArray Destructor 3271
DynamicArray Constructor 3271
DynamicArray Methods 3272
DynamicArray Operators 3276
DynamicArray Properties 3277
DynArrayException Class 3279
DynArrayNullData Class 3279
DynArrayOutOfRange Class 3279
DynArrayOutOfRange Members 3280
DynArrayOutOfRange::DynArrayOutOfRange Constructor 3280
DynArrayOutOfRange Data Members 3280
OleVariant Class 3281
OleVariant Members 3282
OleVariant::~OleVariant Destructor 3286
OleVariant Constructor 3286
OleVariant Data Members 3296
OleVariant Operators 3315
SmallString Template 3322
SmallString Members 3322

SmallString Constructor 3322
SmallString Operators 3323
SmallStringBase Template 3324
SmallStringBase Members 3325
SmallStringBase Data Members 3325
sPointer Template 3326
sPointer Members 3326
sPointer::~sPointer Destructor 3326
sPointer Constructor 3326
sPointer Operators 3327
TAggregatedObject Class 3328
TAggregatedObject Members 3328
TAggregatedObject::Create Constructor 3329
TAggregatedObject Methods 3330
TAggregatedObject Properties 3330
TClassHelperBase Class 3331
TClassHelperBase Members 3331
TClassHelperBase._Create Constructor 3333
TClassHelperBase Fields 3333
TContainedObject Class 3333
TContainedObject Members 3334
TContainedObject Methods 3335
TDateTime Class 3335
TDateTime Members 3336
TDateTime Constructor 3338
TDateTime Enumerations 3339
TDateTime Methods 3340
TDateTime Operators 3346
TDateTimeBase Class 3354
TDateTimeBase Members 3354
TDateTimeBase Data Members 3354
TInterfacedObject Class 3355
TInterfacedObject Members 3355
TInterfacedObject Data Members 3356
TInterfacedObject Methods 3356
TInterfacedObject Properties 3359
TMetaClass Class 3360
TMonitor Record 3361
TMonitor Members 3362

TMonitor Methods 3362
TObject Class 3364
TObject Members 3365
TObject::Create Constructor 3366
TObject::Destroy Destructor 3366
TObject Methods 3367
UnicodeString Class 3393
UnicodeString Members 3394
UnicodeString::~UnicodeString Destructor 3396
UnicodeString Constructor 3396
UnicodeString Structures 3398
UnicodeString Enumerations 3399
UnicodeString Methods 3400
UnicodeString Operators 3423
UnicodeString Friends 3429
Variant Class 3430
Variant Members 3431
Variant::~Variant Destructor 3433
Variant Constructor 3434
Variant Methods 3437
Variant Operators 3445
WideString Class 3476
WideString Members 3476
WideString::~WideString Destructor 3478
WideString Constructor 3478
WideString Enumerations 3480
WideString Methods 3481
WideString Operators 3491

Interfaces 3497

IComparable Interface 3497
IComparable Members 3497
IComparable Methods 3497
IDispatch Interface 3498
IDispatch Members 3498
IDispatch Methods 3498
IEnumerable Interface 3500
IEnumerable Members 3500
IEnumerable Methods 3500
IEnumerator Interface 3501

IEnumerator Members 3501
IEnumerator Methods 3501
IEnumerator Properties 3502
IInterface Interface 3503
IInterface Members 3503
IInterface Methods 3504
IInvokable Interface 3505

Functions **3507**

System.__ArrayToPasStr Function 3518
System.__CLenToPasStr Function 3518
System.__CToPasStr Function 3518
System::__currMul Function 3519
System.__IOTest Function 3519
System.__lldiv Function 3519
System.__lldivo Function 3520
System.__llmod Function 3520
System.__llmodo Function 3520
System.__llmul Function 3521
System.__llmulo Function 3521
System.__llshl Function 3521
System.__lludiv Function 3522
System.__llumod Function 3522
System.__llushr Function 3522
System.__PasToCStr Function 3523
System._AbstractError Function 3523
System._AddRef Function 3523
System._AddRefArray Function 3524
System._AddRefRecord Function 3524
System._AfterConstruction Function 3524
System._Append Function 3525
System._AsClass Function 3525
System._Assert Function 3525
System._Assign Function 3526
System._AStrCmp Function 3526
System._BeforeDestruction Function 3526
System._BlockRead Function 3527
System._BlockWrite Function 3527
System._BoundErr Function 3527
System._CallDynaClass Function 3528

System._CallDynaInst Function 3528
System._CheckAutoResult Function 3528
System._ClassCreate Function 3529
System._ClassDestroy Function 3529
System._Close Function 3529
System._Copy Function 3530
System._CopyArray Function 3530
System._CopyObject Function 3530
System._CopyRecord Function 3531
System._CVR_PROBE Function 3531
System._CVR_STMTPROBE Function 3531
System._Delete Function 3532
System._Dispose Function 3532
System._DoneExcept Function 3532
System._DynArrayAddRef Function 3533
System._DynArrayAsg Function 3533
System._DynArrayClear Function 3533
System._DynArrayCopy Function 3534
System._DynArrayCopyRange Function 3534
System._DynArrayHigh Function 3534
System._DynArrayLength Function 3535
System._DynArraySetLength Function 3535
System._EnsureAnsiString Function 3535
System._EnsureUnicodeString Function 3536
System._EofFile Function 3536
System._EofText Function 3536
System._Eoln Function 3537
System._Erase Function 3537
System._Ext2Real Function 3537
System._Fail Function 3538
System._FilePos Function 3538
System._FileSize Function 3538
System._FillChar Function 3539
System._Finalize Function 3539
System._FinalizeArray Function 3539
System._FinalizeRecord Function 3540
System._FindDynaClass Function 3540
System._FindDynaInst Function 3540
System._Flush Function 3541

System._FreeMem Function 3541
System._FSafeDivide Function 3541
System._FSafeDivideR Function 3542
System._GetHelperIntf Function 3542
System._GetMem Function 3542
System._Halt Function 3543
System._Halt0 Function 3543
System._HandleAnyException Function 3543
System._HandleAutoException Function 3544
System._HandleFinally Function 3544
System._HandleOnException Function 3544
System._Initialize Function 3545
System._InitializeArray Function 3545
System._InitializeRecord Function 3545
System._InitImports Function 3546
System._InitResStringImports Function 3546
System._InitResStrings Function 3546
System._InitUnicodeStrings Function 3547
System._InitWideStrings Function 3547
System._Insert Function 3547
System._InternalLStrFromUStr Function 3548
System._InternalUStrFromLStr Function 3548
System._IntfAddRef Function 3548
System._IntfCast Function 3549
System._IntfClear Function 3549
System._IntfCopy Function 3549
System._IntfDispCall Function 3550
System._IntfVarCall Function 3550
System._IntOver Function 3550
System._IsClass Function 3551
System._LGetDir Function 3551
System._LStrAddRef Function 3551
System._LStrArrayClr Function 3552
System._LStrAsg Function 3552
System._LStrCat Function 3552
System._LStrCat3 Function 3553
System._LStrCatN Function 3553
System._LStrClr Function 3553
System._LStrCmp Function 3554

System._LStrCopy Function 3554
System._LStrDelete Function 3554
System._LStrEqual Function 3555
System._LStrFromArray Function 3555
System._LStrFromChar Function 3555
System._LStrFromPChar Function 3556
System._LStrFromPCharLen Function 3556
System._LStrFromPWChar Function 3556
System._LStrFromPWCharLen Function 3557
System._LStrFromString Function 3557
System._LStrFromUStr Function 3557
System._LStrFromWArray Function 3558
System._LStrFromWChar Function 3558
System._LStrFromWStr Function 3558
System._LStrInsert Function 3559
System._LStrLAsg Function 3559
System._LStrLen Function 3559
System._LStrPos Function 3560
System._LStrSetLength Function 3560
System._LStrToPChar Function 3560
System._LStrToString Function 3561
System._New Function 3561
System._NewAnsiString Function 3561
System._NewUnicodeString Function 3562
System._NewWideString Function 3562
System._ObjCopy Function 3562
System._ObjSetup Function 3563
System._PackageLoad Function 3563
System._PackageUnload Function 3563
System._PCharLen Function 3564
System._Pos Function 3564
System._Pow10 Function 3564
System._PStrCat Function 3565
System._PStrCmp Function 3565
System._PStrCpy Function 3565
System._PStrNCat Function 3566
System._PStrNCpy Function 3566
System._PWCharLen Function 3566
System._RaiseAgain Function 3567

System._RaiseExcept Function 3567
System._ReadChar Function 3567
System._ReadCString Function 3568
System._ReadExt Function 3568
System._ReadInt64 Function 3568
System._ReadLn Function 3569
System._ReadLong Function 3569
System._ReadLString Function 3569
System._ReadRec Function 3570
System._ReadString Function 3570
System._ReadUString Function 3570
System._ReadWChar Function 3571
System._ReadWCString Function 3571
System._ReadWString Function 3571
System._Real2Ext Function 3572
System._ReallocMem Function 3572
System._Rename Function 3572
System._ResetFile Function 3573
System._ResetText Function 3573
System._RewritFile Function 3573
System._RewritText Function 3574
System._ROUND Function 3574
System::_roundToInt64 Function 3574
System._Run0Error Function 3575
System._RunError Function 3575
System._Seek Function 3575
System._SeekEof Function 3576
System._SeekEoln Function 3576
System._SetElem Function 3576
System._SetEq Function 3577
System._SetExpand Function 3577
System._SetIntersect Function 3577
System._SetIntersect3 Function 3578
System._SetLe Function 3578
System._SetLength Function 3578
System._SetRange Function 3579
System._SetString Function 3579
System._SetSub Function 3579
System._SetSub3 Function 3580

System._SetTextBuf Function 3580
System._SetUnion Function 3580
System._SetUnion3 Function 3581
System._SGetDir Function 3581
System._StartExe Function 3581
System._StartLib Function 3582
System._Str0Ext Function 3582
System._Str0Int64 Function 3582
System._Str0Long Function 3583
System._Str0UInt64 Function 3583
System._Str1Ext Function 3583
System._Str2Ext Function 3584
System._StrInt64 Function 3584
System._StrLong Function 3584
System._StrUInt64 Function 3585
System._TRUNC Function 3585
System._Truncate Function 3585
System._TryFinallyExit Function 3586
System._UGetDir Function 3586
System._UniqueStringA Function 3586
System._UniqueStringU Function 3587
System._UniqueStringW Function 3587
System._UStrAddRef Function 3587
System._UStrArrayClr Function 3588
System._UStrAsg Function 3588
System._UStrCat Function 3588
System._UStrCat3 Function 3589
System._UStrCatN Function 3589
System._UStrClr Function 3589
System._UStrCmp Function 3590
System._UStrCopy Function 3590
System._UStrDelete Function 3590
System._UStrEqual Function 3591
System._UStrFromArray Function 3591
System._UStrFromChar Function 3591
System._UStrFromLStr Function 3592
System._UStrFromPChar Function 3592
System._UStrFromPCharLen Function 3592
System._UStrFromPWChar Function 3593

System._UStrFromPWCharLen Function 3593
System._UStrFromString Function 3593
System._UStrFromWArray Function 3594
System._UStrFromWChar Function 3594
System._UStrFromWStr Function 3594
System._UStrInsert Function 3595
System._UStrLAsg Function 3595
System._UStrLen Function 3595
System._UStrSetLength Function 3596
System._UStrToPWChar Function 3596
System._UStrToString Function 3596
System._ValExt Function 3597
System._ValInt64 Function 3597
System._ValLong Function 3597
System._WCharToString Function 3598
System._WGetDir Function 3598
System._Write0Bool Function 3598
System._Write0Char Function 3599
System._Write0CString Function 3599
System._Write0Ext Function 3599
System._Write0Int64 Function 3600
System._Write0Long Function 3600
System._Write0LString Function 3600
System._Write0String Function 3601
System._Write0UInt64 Function 3601
System._Write0UString Function 3601
System._Write0Variant Function 3602
System._Write0WChar Function 3602
System._Write0WCString Function 3602
System._Write0WString Function 3603
System._Write1Ext Function 3603
System._Write2Ext Function 3603
System._WriteBool Function 3604
System._WriteChar Function 3604
System._WriteCString Function 3604
System._WriteInt64 Function 3605
System._WriteLn Function 3605
System._WriteLong Function 3605
System._WriteLString Function 3606

System._WriteRec Function 3606
System._WriteString Function 3606
System._WriteUInt64 Function 3607
System._WriteUString Function 3607
System._WriteVariant Function 3607
System._WriteWChar Function 3608
System._WriteWCString Function 3608
System._WriteWString Function 3608
System._WStrAddRef Function 3609
System._WStrArrayClr Function 3609
System._WStrAsg Function 3609
System._WStrCat Function 3610
System._WStrCat3 Function 3610
System._WStrCatN Function 3610
System._WStrClr Function 3611
System._WStrCmp Function 3611
System._WStrCopy Function 3611
System._WStrDelete Function 3612
System._WStrEqual Function 3612
System._WStrFromArray Function 3612
System._WStrFromChar Function 3613
System._WStrFromLStr Function 3613
System._WStrFromPChar Function 3613
System._WStrFromPCharLen Function 3614
System._WStrFromPWChar Function 3614
System._WStrFromPWCharLen Function 3614
System._WStrFromString Function 3615
System._WStrFromUStr Function 3615
System._WStrFromWArray Function 3615
System._WStrFromWChar Function 3616
System._WStrInsert Function 3616
System._WStrLAsg Function 3616
System._WStrLCmp Function 3617
System._WStrLen Function 3617
System._WStrSetLength Function 3617
System._WStrToPWChar Function 3618
System._WStrToString Function 3618
System::+ Function 3618
System::+ Function 3619

System.Abs Function 3619
System.AcquireExceptionObject Function 3619
System.AddModuleUnloadProc Function 3620
System.Addr Function 3621
System.AllocMem Function 3622
System.AnsiToUtf8 Function 3624
System.Append Function 3624
System.ArcTan Function 3625
System.Assert Function 3627
System.Assigned Function 3628
System.AssignFile Function 3629
System.AttemptToUseSharedMemoryManager Function 3631
System.BeginThread Function 3632
System.BlockRead Function 3633
System.BlockWrite Function 3634
System.Break Function 3636
System.ChDir Function 3638
System::CheckSafecallResult Function 3640
System.Chr Function 3640
System.Close Function 3643
System.CloseFile Function 3643
System.CompareTo Function 3645
System.CompToCurrency Function 3645
System.CompToDouble Function 3646
System.Concat Function 3646
System.Continue Function 3647
System.Copy Function 3648
System.Cos Function 3650
System.CurrencyToComp Function 3651
System.Dec Function 3652
System.Delete Function 3653
System.Dispose Function 3654
System.DoubleToComp Function 3656
System.DynArrayClear Function 3656
System.DynArrayDim Function 3657
System.DynArraySetLength Function 3657
System.EndThread Function 3657
System.EnumModules Function 3658
System.EnumResourceModules Function 3658

System.Eof Function 3659
System.Eoln Function 3660
System.Equals Function 3661
System.Erase Function 3661
System.Error Function 3662
System.ExceptAddr Function 3663
System.ExceptObject Function 3664
System.Exclude Function 3664
System.Exit Function 3665
System.Exp Function 3667
System.FilePos Function 3668
System.FileSize Function 3669
System.FillChar Function 3670
System.Finalize Function 3672
System.FindClassHInstance Function 3672
System::FindDynaInst Function 3673
System.FindHInstance Function 3673
System.FindResourceHInstance Function 3674
System.Flush Function 3674
System.FPower10 Function 3675
System.Frac Function 3675
System.FreeMem Function 3676
System.FreeMemory Function 3678
System.Get8087CW Function 3679
System.GetCurrent Function 3679
System.GetDir Function 3679
System.GetEnumerator Function 3681
System.GetHeapStatus Function 3681
System.GetLastError Function 3681
System.GetMem Function 3683
System.GetMemory Function 3684
System.GetMemoryManager Function 3686
System.GetMemoryManagerState Function 3690
System.GetMemoryMap Function 3691
System.GetMinimumBlockAlignment Function 3691
System.GetModuleFileName Function 3692
System.GetVariantManager Function 3692
System.Halt Function 3693
System.Hi Function 3694

System.High Function 3694
System.Inc Function 3696
System.Include Function 3697
System.Initialize Function 3698
System.Insert Function 3698
System.Int Function 3699
System.IOResult Function 3700
System.IsMemoryManagerSet Function 3701
System.IsVariantManagerSet Function 3701
System.Length Function 3702
System.Ln Function 3703
System.Lo Function 3704
System.LoadResourceModule Function 3705
System::LoadResourceString Function 3705
System.LoadResString Function 3705
System.Low Function 3706
System.Mark Function 3707
System.MemoryBarrier Function 3707
System.MkDir Function 3708
System.MonitorEnter Function 3709
System.MonitorExit Function 3709
System.MonitorPulse Function 3710
System.MonitorPulseAll Function 3710
System.MonitorTryEnter Function 3710
System.MonitorWait Function 3711
System.Move Function 3711
System.MoveChars Function 3712
System.New Function 3712
System.Odd Function 3714
System.OleStrToString Function 3715
System.OleStrToStrVar Function 3715
System.Ord Function 3716
System.ParamCount Function 3716
System.ParamStr Function 3718
System.Pi Function 3719
System.Pos Function 3720
System.Pred Function 3721
System.Ptr Function 3723
System.PUCS4Chars Function 3724

System.RaiseList Function 3724
System.Random Function 3725
System.Randomize Function 3726
System.Read Function 3727
System.ReadLn Function 3728
System.ReallocMem Function 3729
System.ReallocMemory Function 3730
System.RegisterExpectedMemoryLeak Function 3731
System.RegisterModule Function 3731
System.Release Function 3732
System.ReleaseExceptionObject Function 3732
System.RemoveModuleUnloadProc Function 3732
System.Rename Function 3733
System.Reset Function 3734
System.Rewrite Function 3736
System.RmDir Function 3738
System.Round Function 3739
System.RunError Function 3740
System.Seek Function 3741
System.SeekEof Function 3742
System.SeekEoln Function 3743
System.Set8087CW Function 3744
System.SetCodePage Function 3746
System.SetInOutRes Function 3746
System.SetLastError Function 3747
System.SetLength Function 3748
System.SetLineBreakStyle Function 3750
System.SetMemoryManager Function 3751
System.SetMinimumBlockAlignment Function 3756
System.SetMultiByteConversionCodePage Function 3756
System.SetRaiseList Function 3757
System.SetString Function 3757
System.SetTextBuf Function 3758
System.SetVariantManager Function 3759
System.ShareMemoryManager Function 3759
System.Sin Function 3760
System.SizeOf Function 3761
System.Slice Function 3762
System.Sqr Function 3762

System.Sqrt Function 3763
System.Str Function 3764
System.StringCodePage Function 3765
System.StringCodePage Function 3765
System.StringElementSize Function 3765
System.StringElementSize Function 3766
System.StringOfChar Function 3766
System.StringRefCount Function 3766
System.StringRefCount Function 3767
System.StringToOleStr Function 3767
System.StringToWideChar Function 3768
System.Succ Function 3768
System.Swap Function 3769
System.SysAllocMem Function 3769
System.SysFreeMem Function 3770
System.SysGetMem Function 3770
System.SysReallocMem Function 3771
System.SysRegisterExpectedMemoryLeak Function 3772
System.SysUnregisterExpectedMemoryLeak Function 3772
System.TextStart Function 3772
System.Trunc Function 3773
System.Truncate Function 3774
System.UCS4StringToUnicodeString Function 3775
System.UCS4StringToWideString Function 3775
System.UnicodeStringToUCS4String Function 3775
System.UnicodeToUtf8 Function 3776
System.UniqueString Function 3777
System.UnregisterExpectedMemoryLeak Function 3777
System.UnregisterModule Function 3778
System.UpCase Function 3778
System.UTF8Decode Function 3779
System.UTF8Encode Function 3780
System.UTF8EncodeToShortString Function 3780
System.UTF8EncodeToShortString Function 3780
System.UTF8EncodeToShortString Function 3781
System.Utf8ToAnsi Function 3781
System.UTF8ToString Function 3782
System.UTF8ToString Function 3782
System.UTF8ToString Function 3782

System.UTF8ToString Function 3783
System.Utf8ToUnicode Function 3783
System.UTF8ToUnicodeString Function 3784
System.UTF8ToUnicodeString Function 3784
System.UTF8ToUnicodeString Function 3784
System.UTF8ToWideString Function 3785
System.Val Function 3785
System.VarArrayRedim Function 3786
System.VarClear Function 3786
System.WideCharLenToString Function 3787
System.WideCharLenToStrVar Function 3788
System.WideCharToString Function 3788
System.WideCharToStrVar Function 3788
System.WideStringToUCS4String Function 3789
System.Write Function 3789
System.WriteLn Function 3792
System.YieldProcessor Function 3793

Structs, Records, Enums 3794

System.PackageInfoTable Record 3795
System.PackageUnitEntry Record 3795
System.TCallDesc Record 3795
System.TChunkStatus Enumeration 3796
System.TCVModInfo Record 3797
System.TDispatchMessage Record 3797
System.TDispDesc Record 3798
System.TDynArrayTypeInfo Record 3798
System.TFileRec Record 3799
System.TGUID Record 3800
System::THeapBlock Structure 3802
System::THeapStatus Record 3802
System.TInitContext Record 3803
System.TInterfaceEntry Record 3803
System.TInterfaceTable Record 3804
System::TLibModule Record 3804
System::TMemoryManager Record 3805
System::TMemoryManagerEx Record 3805
System::TMemoryManagerState Record 3806
System.TMethod Record 3806
System.TMinimumBlockAlignment Enumeration 3807

System::TModuleUnloadRec Record 3808
System.TMonitorSupport Record 3808
System::TResStringRec Record 3809
System.TRuntimeError Enumeration 3809
System::TSmallBlockTypeState Record 3810
System.TTextLineBreakStyle Enumeration 3811
System.TTextRec Record 3811
System::TVarArray Record 3812
System::TVarArrayBound Record 3813
System.TVarData Record 3813
System::TVariantManager Record 3817
System::TVarRec Record 3819
System::TCallDesc Structure 3820
System::TDispDesc Structure 3820

Types **3822**

System.CppLongInt Type 3824
System.CppULongInt Type 3825
System.GetPackageInfoTable Type 3825
System.HGLOBAL Type 3825
System.HINST Type 3826
System.HMODULE Type 3826
System.HRESULT Type 3826
System.HRSRC Type 3827
System.Int16 Type 3827
System.Int32 Type 3827
System.Int8 Type 3828
System.IntegerArray Type 3828
System.IUnknown Type 3828
System.PackageInfo Type 3829
System.PAnsiString Type 3829
System.PBoolean Type 3829
System.PByte Type 3830
System::PCallDesc Type 3830
System.PCardinal Type 3831
System.PComp Type 3831
System.PCurrency Type 3831
System.PCVModInfo Type 3832
System.PDate Type 3832
System.PDateTime Type 3833

System.PDispatch Type 3833
System::PDispDesc Type 3833
System.PDouble Type 3834
System.PDynArrayTypeInfo Type 3834
System.PError Type 3834
System.PExtended Type 3835
System.PGUID Type 3835
System.PInitContext Type 3837
System.PInt64 Type 3837
System.PInteger Type 3837
System.PIntegerArray Type 3838
System.PInterfaceEntry Type 3838
System.PInterfaceTable Type 3839
System.PLibModule Type 3839
System.PLongint Type 3839
System.PLongWord Type 3840
System::PMemoryManager Type 3840
System::PMemoryManagerEx Type 3840
System::PModuleUnloadRec Type 3841
System.PMonitor Type 3841
System.PMonitorSupport Type 3841
System.PointerArray Type 3842
System.POleVariant Type 3842
System.PPAnsiChar Type 3842
System.PPCharArray Type 3843
System.PPDispatch Type 3843
System.PPMonitor Type 3843
System.PPointer Type 3844
System.PPointerArray Type 3844
System.PPUnknown Type 3844
System.PPWideChar Type 3845
System.PRawByteString Type 3845
System::PResStringRec Type 3845
System.PShortInt Type 3846
System.PShortString Type 3846
System.PSingle Type 3846
System::PSmallBlockTypeState Type 3847
System.PSmallInt Type 3847
System.PTextBuf Type 3848

System.PUCS4Char Type 3848
System.PUCS4CharArray Type 3848
System.PUInt64 Type 3849
System.PUnicodeString Type 3849
System.PUnitEntryTable Type 3849
System.PUnknown Type 3850
System.PUTF8String Type 3850
System.PVarArray Type 3850
System.PVarArrayBoundArray Type 3851
System.PVarArrayCoorArray Type 3851
System.PVarData Type 3851
System.PVariant Type 3852
System::PVariantManager Type 3852
System::PVarRec Type 3852
System.PWideString Type 3853
System.PWord Type 3853
System.PWordBool Type 3853
System.RawByteString Type 3854
System::ResourceString Type 3854
System.TAssertErrorProc Type 3855
System::TBooleanDynArray Type 3855
System.TBoundArray Type 3855
System::TByteDynArray Type 3856
System::TCardinalDynArray Type 3856
System.TClass Type 3856
System.TClassHelperBaseClass Type 3857
System.TDate Type 3857
System.TDateTime Type 3858
System.TDLLProc Type 3859
System.TDLLProcEx Type 3859
System::TDoubleDynArray Type 3859
System.TEnumModuleFunc Type 3860
System.TEnumModuleFuncLW Type 3860
System.TFileIOFunc Type 3860
System::TInt64DynArray Type 3861
System::TIntegerDynArray Type 3861
System.TInterfacedClass Type 3861
System::TLongWordDynArray Type 3862
System.TMemoryMap Type 3862

System.TModuleUnloadProc Type 3863
System.TModuleUnloadProcLW Type 3863
System.TPCharArray Type 3863
System.TResourceHandle Type 3864
System.TSafeCallErrorProc Type 3864
System::TShortIntDynArray Type 3864
System::TSingleDynArray Type 3865
System::TSmallBlockTypeStates Type 3865
System::TSmallIntDynArray Type 3865
System::TStringDynArray Type 3866
System.TSystemThreadEndProc Type 3866
System.TSystemThreadFuncProc Type 3866
System.TTextBuf Type 3867
System.TTextIOFunc Type 3867
System::TThreadFunc Type 3867
System.TTime Type 3868
System.TUCS4CharArray Type 3868
System.TVarArrayBoundArray Type 3868
System.TVarArrayCoorArray Type 3869
System.TVarOp Type 3869
System::TWideStringDynArray Type 3870
System::TWordDynArray Type 3870
System.UCS4Char Type 3871
System.UCS4String Type 3871
System.UInt16 Type 3872
System.UInt32 Type 3872
System.UInt8 Type 3872
System.UnitEntryTable Type 3873
System.UTF8String Type 3873

Variables **3874**

System.AbstractErrorProc Variable 3875
System.AllocMemCount Variable 3875
System.AllocMemSize Variable 3877
System.AssertErrorProc Variable 3878
System.CmdLine Variable 3880
System.CmdShow Variable 3880
System.CPUCount Variable 3881
System.DebugHook Variable 3881
System.Default8087CW Variable 3881

System.DefaultSystemCodePage Variable 3883
System.DefaultTextLineBreakStyle Variable 3883
System.DefaultUnicodeCodePage Variable 3884
System.DispCallByIDProc Variable 3884
System.DLLProc Variable 3884
System.envp Variable 3885
System.ErrorAddr Variable 3885
System.ErrorProc Variable 3886
System.ErrOutput Variable 3888
System.ExceptClsProc Variable 3888
System.ExceptionAcquired Variable 3889
System.ExceptionClass Variable 3889
System.ExceptObjProc Variable 3889
System.ExceptProc Variable 3890
System.ExitCode Variable 3891
System.ExitProc Variable 3891
System.ExitProcessProc Variable 3892
System.FileMode Variable 3892
System.HeapAllocFlags Variable 3893
System.HPrevInst Variable 3894
System.InitProc Variable 3894
System.Input Variable 3895
System.IsConsole Variable 3896
System.IsLibrary Variable 3896
System.IsMultiThread Variable 3897
System.JITEnable Variable 3897
System.LibModuleList Variable 3898
System.MainInstance Variable 3898
System.MainThreadID Variable 3899
System.ModuleUnloadList Variable 3899
System.MonitorSupport Variable 3900
System.NeverSleepOnMMThreadContention Variable 3900
System.NoErrMsg Variable 3901
System.Output Variable 3901
System.RaiseExceptionProc Variable 3902
System.RaiseExceptObjProc Variable 3902
System.RandSeed Variable 3902
System.ReportMemoryLeaksOnShutdown Variable 3903
System.RTLUnwindProc Variable 3903

System.SafeCallErrorProc Variable 3904
System.SystemThreadEndProc Variable 3904
System.SystemThreadFuncProc Variable 3905
System.Test8086 Variable 3905
System.Test8087 Variable 3905
System.TestFDIV Variable 3906
System.UTF8CompareLocale Variable 3906
System.VarAddRefProc Variable 3906
System.VarClearProc Variable 3907
System.VarCopyProc Variable 3907
System.VarToLStrProc Variable 3907
System.VarToWStrProc Variable 3908

Constants 3909

System.CompilerVersion Constant 3911
System.CPUi386 Constant 3911
System.CPUi486 Constant 3912
System.CPUPentium Constant 3912
System.E_NOINTERFACE Constant 3912
System.E_NOTIMPL Constant 3913
System.E_UNEXPECTED Constant 3913
System.fmClosed Constant 3913
System.fmInOut Constant 3914
System.fmInput Constant 3914
System.fmOutput Constant 3914
System.hfFieldSize Constant 3915
System.hfMonitorOffset Constant 3915
System.INFINITE Constant 3915
System.MaxInt Constant 3916
System.MaxLongint Constant 3916
System.NumSmallBlockTypes Constant 3916
System.opAdd Constant 3917
System.opAnd Constant 3917
System.opCmpEQ Constant 3918
System.opCmpGE Constant 3918
System.opCmpGT Constant 3919
System.opCmpLE Constant 3919
System.opCmpLT Constant 3920
System.opCmpNE Constant 3920
System.opCompare Constant 3921

System.opDivide Constant 3921
System.opIntDivide Constant 3922
System.opModulus Constant 3922
System.opMultiply Constant 3923
System.opNegate Constant 3923
System.opNot Constant 3924
System.opOr Constant 3924
System.opShiftLeft Constant 3925
System.opShiftRight Constant 3925
System.opSubtract Constant 3926
System.opXor Constant 3926
System.RTLVersion Constant 3927
System.S_FALSE Constant 3927
System.S_OK Constant 3927
System.sLineBreak Constant 3928
System.tfCRLF Constant 3928
System.varAny Constant 3928
System.varArray Constant 3929
System.varBoolean Constant 3930
System.varByRef Constant 3930
System.varByte Constant 3931
System.varCurrency Constant 3931
System.varDate Constant 3932
System.varDispatch Constant 3932
System.varDouble Constant 3933
System.varEmpty Constant 3933
System.varError Constant 3934
System.varInt64 Constant 3934
System.varInteger Constant 3935
System.varLongWord Constant 3935
System.varNull Constant 3936
System.varOleStr Constant 3936
System.varShortInt Constant 3937
System.varSingle Constant 3937
System.varSmallint Constant 3938
System.varStrArg Constant 3938
System.varString Constant 3938
System.varTypeMask Constant 3939
System.varUInt64 Constant 3940

System.varUnknown Constant 3940
System.varUString Constant 3940
System.varVariant Constant 3941
System.varWord Constant 3941
System.vmtAddRef Constant 3942
System.vmtAfterConstruction Constant 3942
System.vmtAutoTable Constant 3942
System.vmtBeforeDestruction Constant 3943
System.vmtClassName Constant 3943
System.vmtCreateObject Constant 3943
System.vmtDefaultHandler Constant 3944
System.vmtDestroy Constant 3944
System.vmtDispatch Constant 3944
System.vmtDynamicTable Constant 3945
System.vmtEquals Constant 3945
System.vmtFieldTable Constant 3945
System.vmtFreeInstance Constant 3946
System.vmtGetHashCode Constant 3946
System.vmtInitTable Constant 3946
System.vmtInstanceSize Constant 3947
System.vmtIntfTable Constant 3947
System.vmtMethodTable Constant 3947
System.vmtNewInstance Constant 3948
System.vmtParent Constant 3948
System.vmtQueryInterface Constant 3948
System.vmtRelease Constant 3949
System.vmtSafeCallException Constant 3949
System.vmtSelfPtr Constant 3949
System.vmtToString Constant 3950
System.vmtTypeInfo Constant 3950
System.vtAnsiString Constant 3950
System.vtBoolean Constant 3951
System.vtChar Constant 3951
System.vtClass Constant 3952
System.vtCurrency Constant 3952
System.vtExtended Constant 3953
System.vtInt64 Constant 3953
System.vtInteger Constant 3954
System.vtInterface Constant 3954

System.vtObject Constant 3955
System.vtPChar Constant 3955
System.vtPointer Constant 3956
System.vtPWideChar Constant 3956
System.vtString Constant 3957
System.vtUnicodeString Constant 3957
System.vtVariant Constant 3957
System.vtWideChar Constant 3958
System.vtWideString Constant 3958

Macros **3960**
System::USE Macro 3960
System::USEADDITIONALFILES Macro 3960
System::USEASM Macro 3961
System::USEDATAMODULE Macro 3961
System::USEDATAMODULENS Macro 3961
System::USEDEF Macro 3962
System::USEFILE Macro 3962
System::USEFORM Macro 3963
System::USEFORMNS Macro 3963
System::USEFORMRES Macro 3963
System::USEIDL Macro 3964
System::USELIB Macro 3964
System::USEOBJ Macro 3964
System::USEPACKAGE Macro 3965
System::USERC Macro 3965
System::USERES Macro 3965
System::USETLB Macro 3966
System::USEUNIT Macro 3966

SysUtils Namespace 3967

Classes **3981**
EAbort Class 3982
EAbstractError Class 3982
EAccessViolation Class 3983
EArgumentException Class 3984
EArgumentOutOfRangeException Class 3984
EAssertionFailed Class 3984
EControlC Class 3985

EConvertError Class 3985
EDivByZero Class 3987
EEncodingError Class 3988
EExternal Class 3988
EExternal Members 3989
EExternal Fields 3991
EExternalException Class 3991
EHeapException Class 3992
EHeapException Members 3993
EHeapException Methods 3994
EInOutError Class 3995
EInOutError Members 3997
EInOutError Fields 3998
EIntError Class 3999
EIntfCastError Class 3999
EIntOverflow Class 4000
EInvalidCast Class 4000
EInvalidContainer Class 4001
EInvalidInsert Class 4001
EInvalidOp Class 4002
EInvalidPointer Class 4002
EMathError Class 4003
EMonitor Class 4004
EMonitorLockException Class 4004
ENoConstructException Class 4004
ENoMonitorSupportException Class 4005
EOSError Class 4005
EOSError Members 4006
EOSError Fields 4007
EOutOfMemory Class 4008
EOverflow Class 4008
EPackageError Class 4009
EPrivilege Class 4009
EProgrammerNotFound Class 4010
EPropReadOnly Class 4010
EPropWriteOnly Class 4011
ERangeError Class 4011
ESafecallException Class 4012
EStackOverflow Class 4013

EUnderflow Class 4013
EVariantError Class 4014
EWin32Error Class 4015
Exception Class 4015
Exception Members 4016
Exception.Create Constructor 4018
Exception.CreateFmt Constructor 4020
Exception.CreateFmtHelp Constructor 4021
Exception.CreateHelp Constructor 4021
CreateRes Constructor 4022
CreateResFmt Constructor 4022
CreateResFmtHelp Constructor 4023
CreateResHelp Constructor 4025
Exception.Destroy Destructor 4026
Exception Fields 4026
Exception Methods 4027
Exception Properties 4030
EZeroDivide Class 4032
TBigEndianUnicodeEncoding Class 4033
TBigEndianUnicodeEncoding Members 4033
TBigEndianUnicodeEncoding Methods 4034
TEncoding Class 4035
TEncoding Members 4038
TEncoding Fields 4039
TEncoding Methods 4040
TEncoding Properties 4048
TLanguages Class 4051
TLanguages Members 4051
TLanguages.Create Constructor 4052
TLanguages Methods 4052
TLanguages Properties 4053
TMBCSEncoding Class 4056
TMBCSEncoding Members 4056
Create Constructor 4057
TMBCSEncoding Methods 4058
TMultiReadExclusiveWriteSynchronizer Class 4061
TMultiReadExclusiveWriteSynchronizer Members 4062
TMultiReadExclusiveWriteSynchronizer.Create Constructor 4063
TMultiReadExclusiveWriteSynchronizer.Destroy Destructor 4063

TMultiReadExclusiveWriteSynchronizer Methods 4064
TMultiReadExclusiveWriteSynchronizer Properties 4066
TSimpleRWSync Class 4066
TSimpleRWSync Members 4067
TSimpleRWSync.Create Constructor 4068
TSimpleRWSync.Destroy Destructor 4068
TSimpleRWSync Methods 4069
TStringBuilder Class 4070
TStringBuilder Members 4073
Create Constructor 4074
TStringBuilder Fields 4075
TStringBuilder Methods 4076
TStringBuilder Properties 4085
TThreadLocalCounter Class 4087
TThreadLocalCounter Members 4087
TThreadLocalCounter.Destroy Destructor 4087
TThreadLocalCounter Methods 4087
TUnicodeEncoding Class 4088
TUnicodeEncoding Members 4089
TUnicodeEncoding.Create Constructor 4090
TUnicodeEncoding Methods 4090
TUTF7Encoding Class 4093
TUTF7Encoding Members 4093
TUTF7Encoding.Create Constructor 4094
TUTF7Encoding Methods 4095
TUTF8Encoding Class 4098
TUTF8Encoding Members 4098
TUTF8Encoding.Create Constructor 4099
TUTF8Encoding Methods 4099

Interfaces **4101**
IReadWriteSync Interface 4101
IReadWriteSync Members 4101
IReadWriteSync Methods 4101

Functions **4103**
SysUtils.Abort Function 4111
SysUtils.AddExitProc Function 4112
SysUtils.AddTerminateProc Function 4112
SysUtils.AdjustLineBreaks Function 4113

SysUtils.AnsiCompareFileName Function 4113
SysUtils.AnsiCompareStr Function 4114
SysUtils.AnsiCompareText Function 4115
SysUtils.AnsiDequotedStr Function 4115
SysUtils.AnsiExtractQuotedStr Function 4116
SysUtils.AnsiLastChar Function 4116
SysUtils.AnsiLowerCase Function 4117
SysUtils.AnsiLowerCaseFileName Function 4117
SysUtils.AnsiPos Function 4118
SysUtils.AnsiQuotedStr Function 4118
SysUtils.AnsiSameStr Function 4119
SysUtils.AnsiSameText Function 4119
SysUtils.AnsiStrAlloc Function 4120
SysUtils.AnsiStrComp Function 4120
SysUtils.AnsiStrIComp Function 4121
SysUtils.AnsiStrLastChar Function 4121
SysUtils.AnsiStrLComp Function 4122
SysUtils.AnsiStrLIComp Function 4122
SysUtils.AnsiStrLower Function 4123
SysUtils.AnsiStrPos Function 4124
SysUtils.AnsiStrRScan Function 4124
SysUtils.AnsiStrScan Function 4125
SysUtils.AnsiStrUpper Function 4125
SysUtils.AnsiUpperCase Function 4126
SysUtils.AnsiUpperCaseFileName Function 4126
SysUtils.AppendStr Function 4127
SysUtils.AssignStr Function 4127
SysUtils.Beep Function 4127
SysUtils.BoolToStr Function 4130
SysUtils.ByteLength Function 4132
SysUtils.BytesOf Function 4132
SysUtils.BytesOf Function 4133
SysUtils.BytesOf Function 4133
SysUtils.BytesOf Function 4133
SysUtils.ByteToCharIndex Function 4134
SysUtils.ByteToCharLen Function 4134
SysUtils.ByteType Function 4135
SysUtils.CallTerminateProcs Function 4136
SysUtils.ChangeFileExt Function 4136

SysUtils.ChangeFilePath Function 4137
SysUtils.CharInSet Function 4138
SysUtils.CharInSet Function 4138
SysUtils.CharLength Function 4139
SysUtils.CharToByteIndex Function 4139
SysUtils.CharToByteLen Function 4140
SysUtils.CharToElementIndex Function 4140
SysUtils.CharToElementIndex Function 4140
SysUtils.CharToElementLen Function 4141
SysUtils.CharToElementLen Function 4141
SysUtils.CheckWin32Version Function 4141
SysUtils.CompareMem Function 4142
SysUtils.CompareStr Function 4142
SysUtils.CompareText Function 4143
SysUtils.CreateDir Function 4147
SysUtils.CreateGUID Function 4148
SysUtils.CurrentYear Function 4150
SysUtils.CurrToStr Function 4150
SysUtils.CurrToStrF Function 4151
SysUtils.Date Function 4151
SysUtils.DateTimeToFileDate Function 4152
SysUtils.DateTimeToStr Function 4153
SysUtils.DateTimeToString Function 4153
SysUtils.DateTimeToSystemTime Function 4155
SysUtils.DateTimeToTimeStamp Function 4156
SysUtils.DateToStr Function 4156
SysUtils.DayOfWeek Function 4158
SysUtils.DecodeDate Function 4159
SysUtils.DecodeDateFully Function 4160
SysUtils.DecodeTime Function 4161
SysUtils.DelegatesEqual Function 4162
SysUtils.DeleteFile Function 4162
SysUtils.DirectoryExists Function 4164
SysUtils.DiskFree Function 4165
SysUtils.DiskSize Function 4166
SysUtils.DisposeStr Function 4167
SysUtils.ElementToCharIndex Function 4168
SysUtils.ElementToCharIndex Function 4168
SysUtils.ElementToCharLen Function 4168

SysUtils.ElementToCharLen Function 4169
SysUtils.EncodeDate Function 4169
SysUtils.EncodeTime Function 4170
SysUtils.ExceptionErrorMessage Function 4171
SysUtils.ExcludeTrailingBackslash Function 4172
SysUtils.ExcludeTrailingPathDelimiter Function 4173
SysUtils.ExpandFileName Function 4173
SysUtils.ExpandFileNameCase Function 4174
SysUtils.ExpandUNCFileName Function 4175
SysUtils.ExtractFileDir Function 4176
SysUtils.ExtractFileDrive Function 4176
SysUtils.ExtractFileExt Function 4177
SysUtils.ExtractFileName Function 4177
SysUtils.ExtractFilePath Function 4179
SysUtils.ExtractRelativePath Function 4181
SysUtils.ExtractShortPathName Function 4182
SysUtils.FileAge Function 4183
SysUtils.FileClose Function 4185
SysUtils.FileCreate Function 4188
SysUtils.FileDateToDateTime Function 4191
SysUtils.FileExists Function 4193
SysUtils.FileGetAttr Function 4197
SysUtils.FileGetDate Function 4199
SysUtils.FileIsReadOnly Function 4200
SysUtils.FileOpen Function 4200
SysUtils.FileRead Function 4204
SysUtils.FileSearch Function 4206
SysUtils.FileSeek Function 4207
SysUtils.FileSetAttr Function 4209
SysUtils.FileSetDate Function 4212
SysUtils.FileSetReadOnly Function 4212
SysUtils.FileWrite Function 4213
SysUtils.FinalizePackage Function 4216
SysUtils.FindClose Function 4216
SysUtils.FindCmdLineSwitch Function 4218
SysUtils.FindFirst Function 4218
SysUtils.FindNext Function 4221
SysUtils.FloatToCurr Function 4223
SysUtils.FloatToDateTime Function 4223

SysUtils.FloatToDecimal Function 4224
SysUtils.FloatToStr Function 4224
SysUtils.FloatToStrF Function 4226
SysUtils.FloatToText Function 4227
SysUtils.FloatToTextFmt Function 4228
SysUtils.FmtLoadStr Function 4229
SysUtils.FmtStr Function 4229
SysUtils.ForceDirectories Function 4230
SysUtils.Format Function 4232
SysUtils.FormatBuf Function 4237
SysUtils.FormatCurr Function 4238
SysUtils.FormatDateTime Function 4239
SysUtils.FormatFloat Function 4241
SysUtils.FreeAndNil Function 4245
SysUtils.GDAL Function 4246
SysUtils.GetCurrentDir Function 4246
SysUtils.GetEnvironmentVariable Function 4248
SysUtils.GetFileVersion Function 4249
SysUtils.GetFormatSettings Function 4250
SysUtils.GetLocaleChar Function 4250
SysUtils.GetLocaleFormatSettings Function 4250
SysUtils.GetLocaleStr Function 4251
SysUtils.GetModuleName Function 4251
SysUtils.GetPackageDescription Function 4252
SysUtils.GetPackageInfo Function 4253
SysUtils.GetTime Function 4253
SysUtils.GUIDToString Function 4254
SysUtils.HashName Function 4256
SysUtils.IncAMonth Function 4256
SysUtils.IncludeTrailingBackslash Function 4257
SysUtils.IncludeTrailingPathDelimiter Function 4257
SysUtils.IncMonth Function 4258
SysUtils.InitializePackage Function 4259
SysUtils.IntToHex Function 4259
SysUtils.IntToStr Function 4260
SysUtils.IsAssembly Function 4261
SysUtils.IsDelimiter Function 4262
SysUtils.IsEqualGUID Function 4262
SysUtils.IsLeadChar Function 4264

SysUtils.IsLeadChar Function 4264
SysUtils.IsLeapYear Function 4264
SysUtils.IsPathDelimiter Function 4266
SysUtils.IsValidIdent Function 4266
SysUtils.Languages Function 4267
SysUtils.LastDelimiter Function 4267
SysUtils.LoadPackage Function 4268
SysUtils.LoadStr Function 4268
SysUtils.LowerCase Function 4269
SysUtils.MSecsToTimeStamp Function 4270
SysUtils.NewStr Function 4271
SysUtils.NextCharIndex Function 4271
SysUtils.Now Function 4271
SysUtils.OutOfMemoryError Function 4272
SysUtils.PlatformBytesOf Function 4273
SysUtils.PlatformStringOf Function 4273
SysUtils.QuotedStr Function 4273
SysUtils.RaiseLastOSError Function 4274
SysUtils.RaiseLastWin32Error Function 4275
SysUtils.RCS Function 4276
SysUtils.RemoveDir Function 4276
SysUtils.RenameFile Function 4277
SysUtils.ReplaceDate Function 4279
SysUtils.ReplaceTime Function 4280
SysUtils.RPR Function 4281
SysUtils.SafeLoadLibrary Function 4281
SysUtils.SameFileName Function 4282
SysUtils.SameStr Function 4282
SysUtils.SameText Function 4283
SysUtils.SetCurrentDir Function 4283
SysUtils.ShowException Function 4284
SysUtils.Sleep Function 4284
SysUtils.StrAlloc Function 4285
SysUtils.StrBufSize Function 4285
SysUtils.StrByteType Function 4286
SysUtils.StrCat Function 4286
SysUtils.StrCharLength Function 4288
SysUtils.StrComp Function 4288
SysUtils.StrCopy Function 4289

SysUtils.StrDispose Function 4291
SysUtils.StrECopy Function 4292
SysUtils.StrEnd Function 4293
SysUtils.StrFmt Function 4294
SysUtils.StrIComp Function 4295
SysUtils.StringOf Function 4296
SysUtils.StringReplace Function 4297
SysUtils.StringToGUID Function 4297
SysUtils.StrLCat Function 4299
SysUtils.StrLComp Function 4300
SysUtils.StrLCopy Function 4302
SysUtils.StrLen Function 4303
SysUtils.StrLFmt Function 4304
SysUtils.StrLIComp Function 4305
SysUtils.StrLower Function 4307
SysUtils.StrMove Function 4308
SysUtils.StrNew Function 4310
SysUtils.StrNextChar Function 4311
SysUtils.StrPas Function 4311
SysUtils.StrPCopy Function 4313
SysUtils.StrPLCopy Function 4314
SysUtils.StrPos Function 4314
SysUtils.StrRScan Function 4315
SysUtils.StrScan Function 4317
SysUtils.StrToBool Function 4318
SysUtils.StrToBoolDef Function 4318
SysUtils.StrToCurr Function 4319
SysUtils.StrToCurrDef Function 4320
SysUtils.StrToDate Function 4320
SysUtils.StrToDateDef Function 4322
SysUtils.StrToDateTime Function 4322
SysUtils.StrToDateTimeDef Function 4323
SysUtils.StrToFloat Function 4324
SysUtils.StrToFloatDef Function 4327
SysUtils.StrToInt Function 4328
SysUtils.StrToInt64 Function 4329
SysUtils.StrToInt64Def Function 4329
SysUtils.StrToIntDef Function 4330
SysUtils.StrToTime Function 4332

SysUtils.StrToTimeDef Function 4333
SysUtils.StrUpper Function 4334
SysUtils.Supports Function 4335
SysUtils.SysErrorMessage Function 4336
SysUtils.SystemTimeToDateTime Function 4336
SysUtils.TextPos Function 4337
SysUtils.TextPos Function 4337
SysUtils.TextToFloat Function 4338
SysUtils.Time Function 4338
SysUtils.TimeStampToDateTime Function 4339
SysUtils.TimeStampToMSecs Function 4340
SysUtils.TimeToStr Function 4340
SysUtils.Trim Function 4342
SysUtils.TrimLeft Function 4343
SysUtils.TrimRight Function 4343
SysUtils.TryEncodeDate Function 4343
SysUtils.TryEncodeTime Function 4344
SysUtils.TryFloatToCurr Function 4345
SysUtils.TryFloatToDateTime Function 4345
SysUtils.TryStrToBool Function 4346
SysUtils.TryStrToCurr Function 4347
SysUtils.TryStrToDate Function 4347
SysUtils.TryStrToDateTime Function 4348
SysUtils.TryStrToFloat Function 4349
SysUtils.TryStrToInt Function 4349
SysUtils.TryStrToInt64 Function 4350
SysUtils.TryStrToTime Function 4351
SysUtils.UIntToStr Function 4351
SysUtils.UIntToStr Function 4352
SysUtils.UnloadPackage Function 4352
SysUtils.UpperCase Function 4352
SysUtils.WideBytesOf Function 4354
SysUtils.WideCompareStr Function 4354
SysUtils.WideCompareText Function 4355
SysUtils.WideFmtStr Function 4355
SysUtils.WideFormat Function 4356
SysUtils.WideFormatBuf Function 4356
SysUtils.WideLowerCase Function 4357
SysUtils.WideSameStr Function 4357

SysUtils.WideSameText Function 4358
SysUtils.WideStrAlloc Function 4358
SysUtils.WideStringOf Function 4359
SysUtils.WideUpperCase Function 4359
SysUtils.Win32Check Function 4359
SysUtils.WrapText Function 4360

Structs, Records, Enums **4361**
SysUtils.Int64Rec Record 4361
SysUtils.LongRec Record 4362
SysUtils.TCharSearch Record 4362
SysUtils.TExceptionRecord Record 4363
SysUtils.TFilenameCaseMatch Enumeration 4364
SysUtils.TFloatFormat Enumeration 4364
SysUtils.TFloatRec Record 4366
SysUtils.TFloatValue Enumeration 4366
SysUtils.TFormatSettings Record 4367
SysUtils.TLangRec Record 4368
SysUtils.TLocaleOptions Enumeration 4369
SysUtils.TMbcsByteType Enumeration 4369
SysUtils.TNameType Enumeration 4370
SysUtils.TSearchRec Record 4371
SysUtils.TSysLocale Record 4374
SysUtils.TThreadInfo Record 4375
SysUtils.TTimeStamp Record 4375
SysUtils.WordRec Record 4376

Types **4377**
SysUtils.ExceptClass Type 4377
SysUtils.PByteArray Type 4378
SysUtils.PDayTable Type 4378
SysUtils.PExceptionRecord Type 4379
SysUtils.PThreadInfo Type 4379
SysUtils.PWordArray Type 4379
SysUtils.TByteArray Type 4380
SysUtils.TBytes Type 4380
SysUtils.TCharArray Type 4381
SysUtils.TDayTable Type 4381
SysUtils.TFileName Type 4381
SysUtils.TIntegerSet Type 4382

SysUtils.TMREWSync Type 4382
SysUtils.TPackageInfoProc Type 4383
SysUtils.TProc Type 4383
SysUtils.TProcedure Type 4384
SysUtils.TReplaceFlags Type 4384
SysUtils.TSysCharSet Type 4384
SysUtils.TTerminateProc Type 4385
SysUtils.TValidatePackageProc Type 4385
SysUtils.TWordArray Type 4386

Variables **4387**

SysUtils.CurrencyDecimals Variable 4388
SysUtils.CurrencyFormat Variable 4388
SysUtils.CurrencyString Variable 4389
SysUtils.DateSeparator Variable 4389
SysUtils.DecimalSeparator Variable 4390
SysUtils.EmptyAnsiStr Variable 4391
SysUtils.EmptyStr Variable 4391
SysUtils.EmptyWideStr Variable 4391
SysUtils.EraNames Variable 4392
SysUtils.EraYearOffsets Variable 4392
SysUtils.FalseBoolStrs Variable 4392
SysUtils.GetDiskFreeSpaceEx Variable 4393
SysUtils.HexDisplayPrefix Variable 4393
SysUtils.LeadBytes Variable 4394
SysUtils.ListSeparator Variable 4394
SysUtils.LongDateFormat Variable 4394
SysUtils.LongDayNames Variable 4395
SysUtils.LongMonthNames Variable 4395
SysUtils.LongTimeFormat Variable 4396
SysUtils.NegCurrFormat Variable 4396
SysUtils.NullAnsiStr Variable 4397
SysUtils.NullStr Variable 4397
SysUtils.NullWideStr Variable 4398
SysUtils.ShortDateFormat Variable 4398
SysUtils.ShortDayNames Variable 4399
SysUtils.ShortMonthNames Variable 4399
SysUtils.ShortTimeFormat Variable 4400
SysUtils.SysLocale Variable 4400
SysUtils.ThousandSeparator Variable 4402

SysUtils.TimeAMString Variable 4402
SysUtils.TimePMString Variable 4402
SysUtils.TimeSeparator Variable 4403
SysUtils.TrueBoolStrs Variable 4403
SysUtils.TwoDigitYearCenturyWindow Variable 4404
SysUtils.Win32BuildNumber Variable 4404
SysUtils.Win32CSDVersion Variable 4405
SysUtils.Win32MajorVersion Variable 4405
SysUtils.Win32MinorVersion Variable 4406
SysUtils.Win32Platform Variable 4406

Constants **4408**

SysUtils.DateDelta Constant 4409
SysUtils.DefaultFalseBoolStr Constant 4410
SysUtils.DefaultTrueBoolStr Constant 4410
SysUtils.DriveDelim Constant 4411
SysUtils.faAnyFile Constant 4411
SysUtils.faArchive Constant 4412
SysUtils.faDirectory Constant 4412
SysUtils.faHidden Constant 4413
SysUtils.faNormal Constant 4414
SysUtils.faReadOnly Constant 4414
SysUtils.faSymLink Constant 4415
SysUtils.faSysFile Constant 4415
SysUtils.faTemporary Constant 4416
SysUtils.faVolumeID Constant 4416
SysUtils.fmOpenRead Constant 4417
SysUtils.fmOpenReadWrite Constant 4417
SysUtils.fmOpenWrite Constant 4418
SysUtils.fmShareCompat Constant 4419
SysUtils.fmShareDenyNone Constant 4419
SysUtils.fmShareDenyRead Constant 4419
SysUtils.fmShareDenyWrite Constant 4420
SysUtils.fmShareExclusive Constant 4420
SysUtils.HoursPerDay Constant 4420
SysUtils.MaxCurrency Constant 4421
SysUtils.MaxDateTime Constant 4421
SysUtils.MaxEraCount Constant 4422
SysUtils.MinCurrency Constant 4422
SysUtils.MinDateTime Constant 4423

SysUtils.MinsPerDay Constant 4423
SysUtils.MinsPerHour Constant 4424
SysUtils.MonthDays Constant 4424
SysUtils.MSecsPerDay Constant 4425
SysUtils.MSecsPerSec Constant 4425
SysUtils.PathDelim Constant 4426
SysUtils.PathSep Constant 4426
SysUtils.pfBCB4Produced Constant 4427
SysUtils.pfConsumerBCB Constant 4427
SysUtils.pfConsumerCompat Constant 4428
SysUtils.pfConsumerDelphi Constant 4428
SysUtils.pfConsumerMask Constant 4428
SysUtils.pfDelphi4Produced Constant 4429
SysUtils.pfDesignOnly Constant 4429
SysUtils.pfExeModule Constant 4430
SysUtils.pfIgnoreDupUnits Constant 4430
SysUtils.pfLibraryModule Constant 4431
SysUtils.pfModuleTypeMask Constant 4431
SysUtils.pfNeverBuild Constant 4432
SysUtils.pfPackageModule Constant 4432
SysUtils.pfProducerMask Constant 4433
SysUtils.pfProducerUndefined Constant 4433
SysUtils.pfRunOnly Constant 4434
SysUtils.pfV3Produced Constant 4434
SysUtils.SecsPerDay Constant 4435
SysUtils.SecsPerHour Constant 4435
SysUtils.SecsPerMin Constant 4436
SysUtils.SwitchChars Constant 4436
SysUtils.ufImplicitUnit Constant 4437
SysUtils.ufMainUnit Constant 4437
SysUtils.ufOrgWeakUnit Constant 4438
SysUtils.ufPackageUnit Constant 4438
SysUtils.ufWeakPackageUnit Constant 4439
SysUtils.ufWeakUnit Constant 4439
SysUtils.UnixDateDelta Constant 4439

TabNotBk Namespace 4441

Classes **4442**

TTabbedNotebook Class 4442
TTabbedNotebook Members 4442
TTabbedNotebook.Create Constructor 4443
TTabbedNotebook.Destroy Destructor 4444
TTabbedNotebook Methods 4444
TTabbedNotebook Properties 4451
TTabbedNotebook Events 4473
TTabPage Class 4480
TTabPage Members 4480
TTabPage.Create Constructor 4481
TTabPage Methods 4481
TTabPage Properties 4482

Types **4491**
TabNotBk.TPageChangeEvent Type 4491

Constants **4492**
TabNotBk.CM_TABFONTCHANGED Constant 4492

Tabs Namespace 4493

Classes **4494**
TScroller Class 4494
TScroller Members 4494
TScroller.Create Constructor 4495
TScroller.Destroy Destructor 4496
TScroller Methods 4496
TScroller Properties 4496
TScroller Events 4499
TTabList Class 4499
TTabList Members 4500
TTabList.Create Constructor 4500
TTabList Methods 4500
TTabSet Class 4506
TTabSet Members 4507
TTabSet.Create Constructor 4509
TTabSet.Destroy Destructor 4509
TTabSet Methods 4510
TTabSet Properties 4516
TTabSet Events 4545

Structs, Records, Enums **4573**
Tabs.TEdgePart Enumeration 4573
Tabs.TEdgeType Enumeration 4573
Tabs.TScrollBtn Enumeration 4574
Tabs.TScrollOrientation Enumeration 4574
Tabs.TTabSetTabStyle Enumeration 4575

Types **4576**
Tabs.TDrawTabEvent Type 4576
Tabs.TMeasureTabEvent Type 4576
Tabs.TTabChangeEvent Type 4577
Tabs.TTabSetDrawTabEvent Type 4577
Tabs.TTabStyle Type 4578

TConnect Namespace 4579

Classes **4580**
TLocalConnection Class 4580
TLocalConnection Members 4581
TLocalConnection.Create Constructor 4581
TLocalConnection.Destroy Destructor 4582
TLocalConnection Methods 4582
TLocalConnection Properties 4591

ThemedActnCtrls Namespace 4593

Classes **4594**
TThemedAddRemoveItem Class 4594
TThemedAddRemoveItem Members 4594
TThemedAddRemoveItem Methods 4595
TThemedButtonControl Class 4597
TThemedButtonControl Members 4597
TThemedButtonControl Methods 4598
TThemedCustomizePopup Class 4599
TThemedCustomizePopup Members 4600
TThemedCustomizePopup.Create Constructor 4600
TThemedCustomizePopup Methods 4600
TThemedDropDownButton Class 4601
TThemedDropDownButton Members 4601
TThemedDropDownButton Methods 4602

TThemedExpandBtn Class 4604
TThemedExpandBtn Members 4604
TThemedExpandBtn Methods 4604
TThemedMenuButton Class 4605
TThemedMenuButton Members 4605
TThemedMenuButton Methods 4606
TThemedMenuItem Class 4607
TThemedMenuItem Members 4607
TThemedMenuItem Methods 4608
TThemedPopupMenu Class 4611
TThemedPopupMenu Members 4611
TThemedPopupMenu.Create Constructor 4611
TThemedPopupMenu Methods 4612
TThemedToolScrollBtn Class 4613
TThemedToolScrollBtn Members 4613
TThemedToolScrollBtn Methods 4614

Themes Namespace 4615

Classes 4617
TThemeServices Class 4617
TThemeServices Members 4617
TThemeServices.Create Constructor 4618
TThemeServices.Destroy Destructor 4619
TThemeServices Methods 4619
TThemeServices Properties 4632
TThemeServices Events 4633

Functions 4634
Themes.ThemeControl Function 4634
Themes.ThemeServices Function 4634
Themes.UnthemedDesigner Function 4634

Structs, Records, Enums 4636
Themes.TThemedButton Enumeration 4636
Themes.TThemedClock Enumeration 4638
Themes.TThemedComboBox Enumeration 4638
Themes.TThemedEdit Enumeration 4639
Themes.TThemedElement Enumeration 4639
Themes.TThemedElementDetails Record 4640
Themes.TThemedExplorerBar Enumeration 4641

Themes.TThemedHeader Enumeration 4642
Themes.TThemedListview Enumeration 4643
Themes.TThemedMenu Enumeration 4643
Themes.TThemedPage Enumeration 4644
Themes.TThemedProgress Enumeration 4645
Themes.TThemedRebar Enumeration 4646
Themes.TThemedScrollBar Enumeration 4647
Themes.TThemedSpin Enumeration 4649
Themes.TThemedStartPanel Enumeration 4650
Themes.TThemedStatus Enumeration 4650
Themes.TThemedTab Enumeration 4651
Themes.TThemedTaskBand Enumeration 4653
Themes.TThemedTaskBar Enumeration 4653
Themes.TThemedToolBar Enumeration 4654
Themes.TThemedToolTip Enumeration 4655
Themes.TThemedTrackBar Enumeration 4656
Themes.TThemedTrayNotify Enumeration 4657
Themes.TThemedTreeview Enumeration 4658
Themes.TThemedWindow Enumeration 4658

Types **4663**

Themes.PThemedElementDetails Type 4663
Themes.TThemeData Type 4663
Themes.TThemeServicesClass Type 4663

Variables **4665**

Themes.ThemeServicesClass Variable 4665

ToolWin Namespace 4667

Classes **4668**

TToolDockForm Class 4668
TToolDockForm Members 4668
TToolDockForm.Create Constructor 4669
TToolDockForm Methods 4669
TToolDockObject Class 4671
TToolDockObject Members 4672
TToolDockObject.Create Constructor 4672
TToolDockObject Methods 4672
TToolWindow Class 4673
TToolWindow Members 4674

TToolWindow.Create Constructor 4674
TToolWindow Methods 4674
TToolWindow Properties 4676

Structs, Records, Enums **4678**
ToolWin.TEdgeBorder Enumeration 4678
ToolWin.TEdgeStyle Enumeration 4679
ToolWin.TSizingOrientation Enumeration 4679

Types **4681**
ToolWin.TEdgeBorders Type 4681

Types Namespace 4683

Functions **4685**
Types.Bounds Function 4685
Types.CenteredRect Function 4686
Types.CenterPoint Function 4686
Types.EqualRect Function 4686
Types.IntersectRect Function 4687
Types.IsRectEmpty Function 4687
Types.OffsetRect Function 4688
Types.Point Function 4688
Types.PtInRect Function 4689
Types.Rect Function 4689
Types.RectHeight Function 4690
Types.RectWidth Function 4690
Types.SmallPoint Function 4690
Types.SplitRect Function 4691
Types.SplitRect Function 4691
Types.UnionRect Function 4692

Structs, Records, Enums **4693**
Types.tagSIZE Record 4693
Types.TPoint Record 4694
Types.TRect Record 4694
Types.TSmallPoint Record 4695
Types.TSplitRectType Enumeration 4696

Types **4697**
Types.DWORD Type 4697
Types.PByte Type 4698

Types.PDouble Type 4698
Types.PInteger Type 4699
Types.PLongint Type 4699
Types.PPoint Type 4699
Types.PRect Type 4700
Types.PSize Type 4700
Types.PSmallInt Type 4701
Types.PSmallPoint Type 4701
Types.SIZE Type 4701
Types.tagPOINT Type 4702
Types.TBooleanDynArray Type 4702
Types.TByteDynArray Type 4702
Types.TCardinalDynArray Type 4703
Types.TDoubleDynArray Type 4703
Types.TInt64DynArray Type 4704
Types.TIntegerDynArray Type 4704
Types.TLongWordDynArray Type 4704
Types.TShortIntDynArray Type 4705
Types.TSingleDynArray Type 4705
Types.TSize Type 4706
Types.TSmallIntDynArray Type 4706
Types.TStringDynArray Type 4706
Types.TValueRelationship Type 4707
Types.TWideStringDynArray Type 4707
Types.TWordDynArray Type 4708

Constants **4709**
Types.EqualsValue Constant 4709
Types.GreaterThanValue Constant 4709
Types.LessThanValue Constant 4710
Types.RT_RCDATA Constant 4710

TypeTrans Namespace 4713

Classes **4714**
ETypeTransException Class 4714
TTypeTranslator Class 4714
TTypeTranslator Members 4715
TTypeTranslator.Create Constructor 4715
TTypeTranslator.Destroy Destructor 4715

TTypeTranslator Methods 4715

Functions **4718**

TypeTrans.FloatToStrEx Function 4718
TypeTrans.GetEnumValueEx Function 4718
TypeTrans.GetEnumValueExW Function 4719
TypeTrans.SetEnumPropEx Function 4719
TypeTrans.SetEnumPropExW Function 4719
TypeTrans.StrToFloatEx Function 4720

Variables **4721**

TypeTrans.TypeTranslator Variable 4721

TypInfo Namespace 4723

Classes **4727**

EPropertyConvertError Class 4727
EPropertyError Class 4727
TPublishableVariantType Class 4728
TPublishableVariantType Members 4729
TPublishableVariantType Methods 4731

Functions **4735**

TypInfo.FreeAndNilProperties Function 4737
TypInfo.GetAnsiStrProp Function 4737
TypInfo.GetAnsiStrProp Function 4738
TypInfo.GetDynArrayProp Function 4738
TypInfo.GetEnumName Function 4739
TypInfo.GetEnumProp Function 4742
TypInfo.GetEnumValue Function 4744
TypInfo.GetFloatProp Function 4745
TypInfo.GetInt64Prop Function 4746
TypInfo.GetInterfaceProp Function 4746
TypInfo.GetMethodProp Function 4747
TypInfo.GetObjectProp Function 4748
TypInfo.GetObjectPropClass Function 4749
TypInfo.GetOrdProp Function 4750
TypInfo.GetPropInfo Function 4751
TypInfo.GetPropInfos Function 4751
TypInfo.GetPropList Function 4752
TypInfo.GetPropName Function 4753

TypInfo.GetPropValue Function 4753
TypInfo.GetSetElementName Function 4754
TypInfo.GetSetElementValue Function 4754
TypInfo.GetSetProp Function 4755
TypInfo.GetStrProp Function 4756
TypInfo.GetTypeData Function 4756
TypInfo.GetTypeName Function 4757
TypInfo.GetUnicodeStrProp Function 4757
TypInfo.GetUnicodeStrProp Function 4758
TypInfo.GetVariantProp Function 4758
TypInfo.GetWideStrProp Function 4759
TypInfo.IsPublishedProp Function 4759
TypInfo.IsStoredProp Function 4760
TypInfo.PropIsType Function 4761
TypInfo.PropType Function 4761
TypInfo.SamePropTypeName Function 4762
TypInfo.SetAnsiStrProp Function 4762
TypInfo.SetAnsiStrProp Function 4763
TypInfo.SetDynArrayProp Function 4763
TypInfo.SetEnumProp Function 4764
TypInfo.SetFloatProp Function 4766
TypInfo.SetInt64Prop Function 4767
TypInfo.SetInterfaceProp Function 4768
TypInfo.SetMethodProp Function 4769
TypInfo.SetObjectProp Function 4770
TypInfo.SetOrdProp Function 4770
TypInfo.SetPropValue Function 4771
TypInfo.SetSetProp Function 4772
TypInfo.SetStrProp Function 4773
TypInfo.SetToString Function 4774
TypInfo.SetUnicodeStrProp Function 4775
TypInfo.SetUnicodeStrProp Function 4775
TypInfo.SetVariantProp Function 4775
TypInfo.SetWideStrProp Function 4776
TypInfo.SortPropList Function 4777
TypInfo.StringToSet Function 4778

Structs, Records, Enums 4779

TypInfo.TFloatType Enumeration 4779
TypInfo.TIntfFlag Enumeration 4780

TypInfo.TManagedField Record 4780
TypInfo.TMethodKind Enumeration 4780
TypInfo.TOrdType Enumeration 4782
TypInfo.TParamFlag Enumeration 4783
TypInfo.TPropData Record 4783
TypInfo.TPropInfo Record 4784
TypInfo.TTypeData Record 4785
TypInfo.TTypeInfo Record 4786
TypInfo.TTypeKind Enumeration 4787

Types **4789**
TypInfo.PPropInfo Type 4789
TypInfo.PPropList Type 4790
TypInfo.PPTypeInfo Type 4790
TypInfo.PTypeData Type 4791
TypInfo.PTypeInfo Type 4794
TypInfo.ShortStringBase Type 4797
TypInfo.TIntfFlags Type 4797
TypInfo.TIntfFlagsBase Type 4798
TypInfo.TParamFlags Type 4799
TypInfo.TParamFlagsBase Type 4799
TypInfo.TPropInfoProc Type 4799
TypInfo.TPropList Type 4800
TypInfo.TTypeKinds Type 4800

Variables **4801**
TypInfo.BooleanIdents Variable 4801
TypInfo.DotSep Variable 4801

Constants **4802**
TypInfo.tkAny Constant 4802
TypInfo.tkMethods Constant 4802
TypInfo.tkProperties Constant 4802

UDDIHelper Namespace 4805

Functions **4806**
UDDIHelper.GetBindingkeyAccessPoint Function 4806

ValEdit Namespace 4807

Classes **4808**

TItemProp Class 4808
TItemProp Members 4808
TItemProp.Create Constructor 4809
TItemProp.Destroy Destructor 4809
TItemProp Methods 4810
TItemProp Properties 4811
TValueListEditor Class 4814
TValueListEditor Members 4815
TValueListEditor.Create Constructor 4817
TValueListEditor.Destroy Destructor 4818
TValueListEditor Methods 4818
TValueListEditor Properties 4830
TValueListEditor Events 4868
TValueListStrings Class 4911
TValueListStrings Members 4911
TValueListStrings.Create Constructor 4912
TValueListStrings.Destroy Destructor 4912
TValueListStrings Methods 4913
TValueListStrings Properties 4919

Structs, Records, Enums **4921**
ValEdit.TDisplayOption Enumeration 4921
ValEdit.TKeyOption Enumeration 4922

Types **4923**
ValEdit.TDisplayOptions Type 4923
ValEdit.TGetPickListEvent Type 4923
ValEdit.TItemProps Type 4924
ValEdit.TKeyOptions Type 4924
ValEdit.TOnValidateEvent Type 4924

VarCmplx Namespace 4925

Functions **4927**
VarCmplx.VarAsComplex Function 4928
VarCmplx.VarComplex Function 4929
VarCmplx.VarComplexAbs Function 4929
VarCmplx.VarComplexAbsSqr Function 4930
VarCmplx.VarComplexAngle Function 4931
VarCmplx.VarComplexArcCos Function 4931
VarCmplx.VarComplexArcCosH Function 4932

VarCmplx.VarComplexArcCot Function 4933
VarCmplx.VarComplexArcCotH Function 4934
VarCmplx.VarComplexArcCsc Function 4935
VarCmplx.VarComplexArcCscH Function 4937
VarCmplx.VarComplexArcSec Function 4938
VarCmplx.VarComplexArcSecH Function 4939
VarCmplx.VarComplexArcSin Function 4940
VarCmplx.VarComplexArcSinH Function 4941
VarCmplx.VarComplexArcTan Function 4942
VarCmplx.VarComplexArcTanH Function 4943
VarCmplx.VarComplexConjugate Function 4944
VarCmplx.VarComplexCos Function 4945
VarCmplx.VarComplexCosH Function 4946
VarCmplx.VarComplexCot Function 4947
VarCmplx.VarComplexCotH Function 4948
VarCmplx.VarComplexCreate Function 4949
VarCmplx.VarComplexCsc Function 4950
VarCmplx.VarComplexCscH Function 4951
VarCmplx.VarComplexExp Function 4952
VarCmplx.VarComplexFromPolar Function 4953
VarCmplx.VarComplexInverse Function 4954
VarCmplx.VarComplexLn Function 4954
VarCmplx.VarComplexLog10 Function 4955
VarCmplx.VarComplexLog2 Function 4956
VarCmplx.VarComplexLogN Function 4956
VarCmplx.VarComplexPower Function 4957
VarCmplx.VarComplexSec Function 4958
VarCmplx.VarComplexSecH Function 4959
VarCmplx.VarComplexSign Function 4960
VarCmplx.VarComplexSimplify Function 4960
VarCmplx.VarComplexSin Function 4961
VarCmplx.VarComplexSinH Function 4962
VarCmplx.VarComplexSqr Function 4963
VarCmplx.VarComplexSqrt Function 4964
VarCmplx.VarComplexTan Function 4964
VarCmplx.VarComplexTanH Function 4965
VarCmplx.VarComplexTimesImaginary Function 4967
VarCmplx.VarComplexTimesNegI Function 4967
VarCmplx.VarComplexTimesPosI Function 4968

VarCmplx.VarComplexTimesReal Function 4968
VarCmplx.VarComplexToPolar Function 4969
VarCmplx.VarIsComplex Function 4969

Variables **4971**
VarCmplx.ComplexNumberDefuzzAtZero Variable 4971
VarCmplx.ComplexNumberSymbol Variable 4971
VarCmplx.ComplexNumberSymbolBeforeImaginary Variable 4972

VarConv Namespace 4973

Functions **4974**
VarConv.VarAsConvert Function 4974
VarConv.VarConvert Function 4975
VarConv.VarConvertCreate Function 4976
VarConv.VarIsConvert Function 4977

VarHlpr Namespace 4979

Functions **4981**
VarHlpr.VariantAdd Function 4981
VarHlpr.VariantAdd2 Function 4982
VarHlpr.VariantAnd Function 4982
VarHlpr.VariantAnd2 Function 4983
VarHlpr.VariantArrayRedim Function 4983
VarHlpr.VariantCast Function 4984
VarHlpr.VariantClear Function 4984
VarHlpr.VariantCpy Function 4985
VarHlpr.VariantDiv Function 4985
VarHlpr.VariantDiv2 Function 4986
VarHlpr.VariantFromUnicodeString Function 4986
VarHlpr.VariantGetElement Function 4987
VarHlpr.VariantMod Function 4988
VarHlpr.VariantMod2 Function 4988
VarHlpr.VariantMul Function 4989
VarHlpr.VariantMul2 Function 4989
VarHlpr.VariantNeg Function 4990
VarHlpr.VariantNot Function 4990
VarHlpr.VariantOr Function 4990
VarHlpr.VariantOr2 Function 4991

VarHlpr.VariantPutElement Function 4992
VarHlpr.VariantShl Function 4992
VarHlpr.VariantShl2 Function 4993
VarHlpr.VariantShr Function 4993
VarHlpr.VariantShr2 Function 4994
VarHlpr.VariantSub Function 4994
VarHlpr.VariantSub2 Function 4995
VarHlpr.VariantToUnicodeString Function 4995
VarHlpr.VariantXor Function 4996
VarHlpr.VariantXor2 Function 4996

Variants Namespace 4997

Classes 5002

EVariantArrayCreateError Class 5002
EVariantArrayLockedError Class 5003
EVariantBadIndexError Class 5004
EVariantBadVarTypeError Class 5004
EVariantDispatchError Class 5005
EVariantInvalidArgError Class 5005
EVariantInvalidNullOpError Class 5006
EVariantInvalidOpError Class 5007
EVariantNotAnArrayError Class 5007
EVariantNotImplError Class 5008
EVariantOutOfMemoryError Class 5008
EVariantOverflowError Class 5009
EVariantRangeCheckError Class 5010
EVariantTypeCastError Class 5010
EVariantUnexpectedError Class 5011
TCustomVariantType Class 5011
TCustomVariantType Members 5012
Create Constructor 5014
TCustomVariantType.Destroy Destructor 5015
TCustomVariantType Methods 5016
TCustomVariantType Properties 5036
TInvokeableVariantType Class 5037
TInvokeableVariantType Members 5038
TInvokeableVariantType Methods 5040

Interfaces 5045

IVarInstanceReference Interface 5045
IVarInstanceReference Members 5045
IVarInstanceReference Methods 5045
IVarInvokeable Interface 5046
IVarInvokeable Members 5046
IVarInvokeable Methods 5046

Functions **5048**
Variants._DispInvoke Function 5050
Variants._OleVarFromInt Function 5051
Variants._OleVarFromInt64 Function 5051
Variants._OleVarFromLStr Function 5051
Variants._OleVarFromPStr Function 5052
Variants._OleVarFromUInt64 Function 5052
Variants._OleVarFromUStr Function 5052
Variants._OleVarFromVar Function 5053
Variants._VarAdd Function 5053
Variants._VarAddRef Function 5053
Variants._VarAnd Function 5054
Variants._VarArrayGet Function 5054
Variants._VarArrayPut Function 5054
Variants._VarArrayRedim Function 5055
Variants._VarCast Function 5055
Variants._VarCastOle Function 5055
Variants._VarClear Function 5056
Variants._VarClr Function 5056
Variants._VarCmpEQ Function 5056
Variants._VarCmpGE Function 5057
Variants._VarCmpGT Function 5057
Variants._VarCmpLE Function 5057
Variants._VarCmpLT Function 5058
Variants._VarCmpNE Function 5058
Variants._VarCopy Function 5058
Variants._VarCopyNoInd Function 5059
Variants._VarDiv Function 5059
Variants._VarFromBool Function 5059
Variants._VarFromCurr Function 5060
Variants._VarFromDisp Function 5060
Variants._VarFromDynArray Function 5060
Variants._VarFromInt Function 5061

Variants._VarFromInt64 Function 5061
Variants._VarFromIntf Function 5061
Variants._VarFromLStr Function 5062
Variants._VarFromPStr Function 5062
Variants._VarFromReal Function 5062
Variants._VarFromTDateTime Function 5063
Variants._VarFromUInt64 Function 5063
Variants._VarFromUStr Function 5063
Variants._VarFromWChar Function 5064
Variants._VarFromWStr Function 5064
Variants._VarMod Function 5064
Variants._VarMul Function 5065
Variants._VarNeg Function 5065
Variants._VarNot Function 5065
Variants._VarOr Function 5066
Variants._VarRDiv Function 5066
Variants._VarShl Function 5066
Variants._VarShr Function 5067
Variants._VarSub Function 5067
Variants._VarToBool Function 5067
Variants._VarToCurrency Function 5068
Variants._VarToDisp Function 5068
Variants._VarToDynArray Function 5068
Variants._VarToInt64 Function 5069
Variants._VarToInteger Function 5069
Variants._VarToIntf Function 5069
Variants._VarToLStr Function 5070
Variants._VarToPStr Function 5070
Variants._VarToReal Function 5070
Variants._VarToUInt64 Function 5071
Variants._VarToUStr Function 5071
Variants._VarToWStr Function 5071
Variants._VarXor Function 5072
Variants.DynArrayFromVariant Function 5072
Variants.DynArrayToVariant Function 5072
Variants.FindCustomVariantType Function 5073
Variants.FindVarData Function 5074
Variants.HandleConversionException Function 5074
Variants.Null Function 5075

Variants.Unassigned Function 5076
Variants.VarArrayAsPSafeArray Function 5077
Variants.VarArrayCreate Function 5077
Variants.VarArrayCreateError Function 5078
Variants.VarArrayDimCount Function 5078
Variants.VarArrayGet Function 5079
Variants.VarArrayHighBound Function 5079
Variants.VarArrayLock Function 5080
Variants.VarArrayLowBound Function 5080
Variants.VarArrayOf Function 5081
Variants.VarArrayPut Function 5081
Variants.VarArrayRef Function 5082
Variants.VarArrayUnlock Function 5083
Variants.VarAsError Function 5083
Variants.VarAsType Function 5083
Variants.VarCastError Function 5084
Variants.VarCheckEmpty Function 5084
Variants.VarCompareValue Function 5085
Variants.VarCopyNoInd Function 5086
Variants.VarEnsureRange Function 5086
Variants.VarFromDateTime Function 5087
Variants.VarInRange Function 5087
Variants.VarInvalidNullOp Function 5088
Variants.VarInvalidOp Function 5088
Variants.VarIsArray Function 5089
Variants.VarIsByRef Function 5089
Variants.VarIsClear Function 5090
Variants.VarIsCustom Function 5090
Variants.VarIsEmpty Function 5091
Variants.VarIsEmptyParam Function 5092
Variants.VarIsError Function 5092
Variants.VarIsFloat Function 5093
Variants.VarIsNull Function 5093
Variants.VarIsNumeric Function 5094
Variants.VarIsOrdinal Function 5094
Variants.VarIsStr Function 5095
Variants.VarIsType Function 5095
Variants.VarOverflowError Function 5096
Variants.VarRangeCheckError Function 5096

Variants.VarResultCheck Function 5097
Variants.VarSameValue Function 5097
Variants.VarSupports Function 5098
Variants.VarToDateTime Function 5098
Variants.VarToStr Function 5099
Variants.VarToStrDef Function 5099
Variants.VarToWideStr Function 5100
Variants.VarToWideStrDef Function 5100
Variants.VarType Function 5101
Variants.VarTypeAsText Function 5102
Variants.VarTypeIsValidArrayType Function 5103
Variants.VarTypeIsValidElementType Function 5103

Structs, Records, Enums **5105**
Variants.TBooleanToStringRule Enumeration 5105
Variants.TNullCompareRule Enumeration 5105
Variants.TVarCompareResult Enumeration 5106
Variants.TVariantRelationship Enumeration 5106

Types **5108**
Variants.TAnyProc Type 5108
Variants.TCustomVariantTypeClass Type 5108
Variants.TVarDataArray Type 5109
Variants.TVarDispProc Type 5109

Variables **5110**
Variants.BooleanToStringRule Variable 5110
Variants.ChangeAnyProc Variable 5111
Variants.ClearAnyProc Variable 5111
Variants.EmptyParam Variable 5111
Variants.NullAsStringValue Variable 5112
Variants.NullEqualityRule Variable 5112
Variants.NullMagnitudeRule Variable 5113
Variants.NullStrictConvert Variable 5114
Variants.OleVariantInt64AsDouble Variable 5114
Variants.PackVarCreation Variable 5115
Variants.RefAnyProc Variable 5115
Variants.VarDispProc Variable 5115

VarUtils Namespace 5117

Classes **5120**
ESafeArrayBoundsError Class 5120
ESafeArrayCreateError Class 5120
ESafeArrayError Class 5121
ESafeArrayError Members 5121
ESafeArrayError.CreateHResult Constructor 5122
ESafeArrayError Properties 5123
ESafeArrayLockedError Class 5123

Functions **5124**
VarUtils.SafeArrayAccessData Function 5124
VarUtils.SafeArrayAllocData Function 5125
VarUtils.SafeArrayAllocDescriptor Function 5125
VarUtils.SafeArrayCheck Function 5125
VarUtils.SafeArrayCopy Function 5126
VarUtils.SafeArrayCopyData Function 5126
VarUtils.SafeArrayCreate Function 5126
VarUtils.SafeArrayDestroy Function 5127
VarUtils.SafeArrayDestroyData Function 5127
VarUtils.SafeArrayDestroyDescriptor Function 5127
VarUtils.SafeArrayError Function 5128
VarUtils.SafeArrayGetDim Function 5128
VarUtils.SafeArrayGetElement Function 5128
VarUtils.SafeArrayGetElemSize Function 5129
VarUtils.SafeArrayGetLBound Function 5129
VarUtils.SafeArrayGetUBound Function 5129
VarUtils.SafeArrayLock Function 5130
VarUtils.SafeArrayPtrOfIndex Function 5130
VarUtils.SafeArrayPutElement Function 5130
VarUtils.SafeArrayRedim Function 5131
VarUtils.SafeArrayUnaccessData Function 5131
VarUtils.SafeArrayUnlock Function 5131
VarUtils.VarExceptionToResult Function 5132
VarUtils.VariantChangeType Function 5132
VarUtils.VariantClear Function 5132
VarUtils.VariantCopy Function 5133
VarUtils.VariantCopyInd Function 5133
VarUtils.VariantInit Function 5133

Structs, Records, Enums **5134**

VarUtils.TVarTypeToElementInfo Record 5134

Variables **5135**

VarUtils.VarAdd Variable 5135
VarUtils.VarAnd Variable 5136
VarUtils.VarBoolFromStr Variable 5136
VarUtils.VarBstrFromBool Variable 5136
VarUtils.VarBstrFromCy Variable 5137
VarUtils.VarBstrFromDate Variable 5137
VarUtils.VarCmp Variable 5137
VarUtils.VarCyFromStr Variable 5138
VarUtils.VarDateFromStr Variable 5138
VarUtils.VarDiv Variable 5138
VarUtils.VarI4FromStr Variable 5139
VarUtils.VariantChangeTypeEx Variable 5139
VarUtils.VarIDiv Variable 5139
VarUtils.VarMod Variable 5140
VarUtils.VarMul Variable 5140
VarUtils.VarNeg Variable 5140
VarUtils.VarNot Variable 5141
VarUtils.VarOr Variable 5141
VarUtils.VarR4FromStr Variable 5141
VarUtils.VarR8FromStr Variable 5142
VarUtils.VarSub Variable 5142
VarUtils.VarXor Variable 5142

Constants **5143**

VarUtils.ARR_DISPATCH Constant 5143
VarUtils.ARR_FIXEDSIZE Constant 5144
VarUtils.ARR_NONE Constant 5144
VarUtils.ARR_OLESTR Constant 5144
VarUtils.ARR_UNKNOWN Constant 5145
VarUtils.ARR_VARIANT Constant 5145
VarUtils.CMaxArrayVarType Constant 5145
VarUtils.CMinArrayVarType Constant 5146
VarUtils.CVarTypeToElementInfo Constant 5146
VarUtils.GenericOperations Constant 5147
VarUtils.GenericSafeArrays Constant 5147
VarUtils.GenericSafeArrayUsesLibC Constant 5147
VarUtils.GenericVariants Constant 5148

VarUtils.GenericVarUtils Constant 5148
VarUtils.VAR_ARRAYISLOCKED Constant 5148
VarUtils.VAR_BADINDEX Constant 5149
VarUtils.VAR_BADVARTYPE Constant 5149
VarUtils.VAR_CMP_EQ Constant 5149
VarUtils.VAR_CMP_GT Constant 5150
VarUtils.VAR_CMP_LT Constant 5150
VarUtils.VAR_CMP_NULL Constant 5150
VarUtils.VAR_EXCEPTION Constant 5151
VarUtils.VAR_INVALIDARG Constant 5151
VarUtils.VAR_LOCALE_USER_DEFAULT Constant 5151
VarUtils.VAR_NOTIMPL Constant 5152
VarUtils.VAR_OK Constant 5152
VarUtils.VAR_OUTOFMEMORY Constant 5152
VarUtils.VAR_OVERFLOW Constant 5153
VarUtils.VAR_PARAMNOTFOUND Constant 5153
VarUtils.VAR_TYPEMISMATCH Constant 5153
VarUtils.VAR_UNEXPECTED Constant 5154

VCLCom Namespace 5155

Classes 5156
TComponentFactory Class 5156
TComponentFactory Members 5156
TComponentFactory.Create Constructor 5157
TComponentFactory Methods 5157

WebAdapt Namespace 5159

Classes 5164
EAdapterException Class 5166
EAdapterException Members 5166
EAdapterException.CreateID Constructor 5167
EAdapterException Properties 5167
EAdapterFieldException Class 5167
EAdapterFieldException Members 5168
EAdapterFieldException.Create Constructor 5168
EAdapterFieldException Properties 5169
TAbstractFieldValuesAdapter Class 5169
TAbstractFieldValuesAdapter Members 5171

TAbstractFieldValuesAdapter Methods 5174
TAdapter Class 5179
TAdapter Members 5181
TAdapter Properties 5184
TAdapter Events 5186
TAdapterAction Class 5190
TAdapterAction Members 5191
TAdapterAction Properties 5194
TAdapterAction Events 5195
TAdapterActions Class 5198
TAdapterActions Members 5199
TAdapterActions Methods 5201
TAdapterActions Properties 5203
TAdapterApplicationField Class 5203
TAdapterApplicationField Members 5204
TAdapterApplicationField Properties 5207
TAdapterApplicationTitleField Class 5207
TAdapterApplicationTitleField Members 5208
TAdapterApplicationTitleField Methods 5211
TAdapterBooleanField Class 5212
TAdapterBooleanField Members 5213
TAdapterBooleanField Properties 5216
TAdapterBooleanField Events 5219
TAdapterDefaultActions Class 5220
TAdapterDefaultActions Members 5221
TAdapterDefaultActions.Destroy Destructor 5222
TAdapterDefaultActions Methods 5223
TAdapterDefaultActions Properties 5226
TAdapterDefaultFields Class 5226
TAdapterDefaultFields Members 5227
TAdapterDefaultFields.Destroy Destructor 5229
TAdapterDefaultFields Methods 5229
TAdapterDefaultFields Properties 5232
TAdapterEndUserDisplayNameField Class 5233
TAdapterEndUserDisplayNameField Members 5233
TAdapterEndUserDisplayNameField Methods 5236
TAdapterEndUserField Class 5237
TAdapterEndUserField Members 5238
TAdapterEndUserField Properties 5240

TAdapterEndUserLoggedInField Class 5241
TAdapterEndUserLoggedInField Members 5241
TAdapterEndUserLoggedInField Methods 5244
TAdapterError Class 5245
TAdapterError Members 5245
TAdapterError.Create Constructor 5247
TAdapterError Methods 5247
TAdapterErrorsList Class 5248
TAdapterErrorsList Members 5249
TAdapterErrorsList.Create Constructor 5251
TAdapterErrorsList.Destroy Destructor 5251
TAdapterErrorsList Methods 5252
TAdapterErrorsList Properties 5254
TAdapterField Class 5256
TAdapterField Members 5257
TAdapterField Properties 5260
TAdapterField Events 5264
TAdapterFields Class 5265
TAdapterFields Members 5266
TAdapterFields Methods 5267
TAdapterFields Properties 5269
TAdapterFileField Class 5270
TAdapterFileField Members 5272
TAdapterFileField Properties 5276
TAdapterFileField Events 5278
TAdapterGotoPageAction Class 5280
TAdapterGotoPageAction Members 5281
TAdapterGotoPageAction Properties 5284
TAdapterGotoPageAction Events 5285
TAdapterHiddenField Class 5288
TAdapterHiddenField Members 5288
TAdapterHiddenField.Create Constructor 5290
TAdapterHiddenField Methods 5290
TAdapterHiddenFieldsList Class 5290
TAdapterHiddenFieldsList Members 5291
TAdapterHiddenFieldsList.Create Constructor 5293
TAdapterHiddenFieldsList.Destroy Destructor 5293
TAdapterHiddenFieldsList Methods 5293
TAdapterHiddenFieldsList Properties 5296

TAdapterImageField Class 5297
TAdapterImageField Members 5298
TAdapterImageField Properties 5302
TAdapterImageField Events 5305
TAdapterLoginField Class 5308
TAdapterLoginField Members 5309
TAdapterLoginField Properties 5311
TAdapterMemoField Class 5312
TAdapterMemoField Members 5313
TAdapterMemoField Properties 5316
TAdapterMemoField Events 5319
TAdapterMultiValueField Class 5320
TAdapterMultiValueField Members 5322
TAdapterMultiValueField Properties 5326
TAdapterMultiValueField Events 5329
TAdapterNamedDisplayField Class 5331
TAdapterNamedDisplayField Members 5332
TAdapterNamedDisplayField.Create Constructor 5334
TAdapterNamedDisplayField Methods 5334
TAdapterNamedDisplayField Properties 5338
TAdapterNamedField Class 5339
TAdapterNamedField Members 5340
TAdapterNamedField Methods 5342
TAdapterNamedField Properties 5355
TAdapterNextPageAction Class 5358
TAdapterNextPageAction Members 5359
TAdapterNextPageAction Methods 5362
TAdapterNextPageAction Properties 5364
TAdapterNextPageAction Events 5365
TAdapterNextPageField Class 5367
TAdapterNextPageField Members 5369
TAdapterNextPageField.Destroy Destructor 5371
TAdapterNextPageField Methods 5371
TAdapterNextPageField Properties 5374
TAdapterObjectErrorsList Class 5375
TAdapterObjectErrorsList Members 5376
TAdapterObjectErrorsList.Create Constructor 5377
TAdapterObjectErrorsList Methods 5378
TAdapterPasswordField Class 5379

TAdapterPasswordField Members 5380
TAdapterPasswordField Methods 5382
TAdapterPasswordField Properties 5383
TAdapterPrevPageAction Class 5384
TAdapterPrevPageAction Members 5385
TAdapterPrevPageAction Methods 5389
TAdapterPrevPageAction Properties 5390
TAdapterPrevPageAction Events 5391
TAdapterSubComponent Class 5394
TAdapterSubComponent Members 5395
TAdapterSubComponent Methods 5396
TAdapterSubComponent Properties 5406
TAdapterUpdateField Class 5407
TAdapterUpdateField Members 5409
TAdapterUpdateField.Create Constructor 5411
TAdapterUpdateField Methods 5412
TAdapterUpdateField Properties 5420
TAdapterUpdateField Events 5421
TAdapterUserNameField Class 5422
TAdapterUserNameField Members 5423
TAdapterUserNameField Methods 5425
TAdapterUserNameField Properties 5426
TApplicationAdapter Class 5427
TApplicationAdapter Members 5428
TApplicationAdapter Properties 5432
TApplicationAdapter Events 5433
TBaseAdapterAction Class 5435
TBaseAdapterAction Members 5436
TBaseAdapterAction Methods 5438
TBaseAdapterAction Properties 5461
TBaseAdapterAction Events 5464
TBaseAdapterErrorsList Class 5467
TBaseAdapterErrorsList Members 5467
TBaseAdapterErrorsList Methods 5469
TBaseAdapterErrorsList Properties 5474
TBaseAdapterMultiValueField Class 5475
TBaseAdapterMultiValueField Members 5477
TBaseAdapterMultiValueField.Create Constructor 5480
TBaseAdapterMultiValueField.Destroy Destructor 5480

TBaseAdapterMultiValueField Methods 5481
TBaseEndUserAdapter Class 5485
TBaseEndUserAdapter Members 5486
TBaseEndUserAdapter Methods 5490
TBaseEndUserAdapter Properties 5503
TBaseEndUserAdapter Events 5503
TBaseValuesListAdapter Class 5504
TBaseValuesListAdapter Members 5506
TBaseValuesListAdapter Methods 5509
TBaseValuesListField Class 5515
TBaseValuesListField Members 5516
TBaseValuesListField Methods 5518
TBaseValuesListField Properties 5518
TCustomAdapter Class 5519
TCustomAdapter Members 5520
TCustomAdapter.Create Constructor 5523
TCustomAdapter.Destroy Destructor 5523
TCustomAdapter Methods 5523
TCustomAdapter Properties 5558
TCustomAdapter Events 5561
TCustomAdapterAction Class 5565
TCustomAdapterAction Members 5566
TCustomAdapterAction Methods 5569
TCustomAdapterAction Properties 5572
TCustomAdapterAction Events 5573
TCustomAdapterBooleanField Class 5575
TCustomAdapterBooleanField Members 5577
TCustomAdapterBooleanField Methods 5580
TCustomAdapterBooleanField Events 5583
TCustomAdapterField Class 5583
TCustomAdapterField Members 5584
TCustomAdapterField Methods 5588
TCustomAdapterField Events 5589
TCustomAdapterFileField Class 5590
TCustomAdapterFileField Members 5591
TCustomAdapterFileField Methods 5594
TCustomAdapterFileField Properties 5598
TCustomAdapterFileField Events 5599
TCustomAdapterGotoPageAction Class 5600

TCustomAdapterGotoPageAction Members 5601
TCustomAdapterGotoPageAction Methods 5605
TCustomAdapterGotoPageAction Events 5608
TCustomAdapterImageField Class 5609
TCustomAdapterImageField Members 5610
TCustomAdapterImageField Methods 5613
TCustomAdapterImageField Properties 5620
TCustomAdapterImageField Events 5621
TCustomAdapterMemoField Class 5624
TCustomAdapterMemoField Members 5625
TCustomAdapterMemoField Methods 5628
TCustomAdapterMemoField Events 5630
TCustomAdapterMultiValueField Class 5632
TCustomAdapterMultiValueField Members 5633
TCustomAdapterMultiValueField Methods 5636
TCustomAdapterMultiValueField Properties 5639
TCustomAdapterMultiValueField Events 5640
TCustomAdapterPageAction Class 5642
TCustomAdapterPageAction Members 5643
TCustomAdapterPageAction Methods 5647
TCustomApplicationAdapter Class 5649
TCustomApplicationAdapter Members 5650
TCustomApplicationAdapter Methods 5654
TCustomApplicationAdapter Properties 5657
TCustomApplicationAdapterAction Class 5657
TCustomApplicationAdapterAction Members 5658
TCustomApplicationAdapterAction Properties 5662
TCustomEndUserAdapter Class 5662
TCustomEndUserAdapter Members 5664
TCustomEndUserAdapter Methods 5668
TCustomEndUserAdapter Events 5671
TCustomEndUserAdapterAction Class 5673
TCustomEndUserAdapterAction Members 5674
TCustomEndUserAdapterAction Properties 5677
TCustomEndUserSessionAdapter Class 5678
TCustomEndUserSessionAdapter Members 5680
TCustomEndUserSessionAdapter Methods 5684
TCustomEndUserSessionAdapter Properties 5686
TCustomLoginFormAdapter Class 5687

TCustomLoginFormAdapter Members 5688
TCustomLoginFormAdapter Methods 5692
TCustomLoginFormAdapter Properties 5699
TCustomLoginFormAdapter Events 5701
TCustomLoginFormAdapterAction Class 5703
TCustomLoginFormAdapterAction Members 5704
TCustomLoginFormAdapterAction Properties 5707
TCustomPagedAdapter Class 5707
TCustomPagedAdapterAction Class 5708
TCustomPagedAdapterAction Members 5709
TCustomPagedAdapterAction Methods 5713
TCustomPagedAdapterAction Properties 5713
TCustomStringsValuesList Class 5714
TCustomStringsValuesList Members 5715
TCustomStringsValuesList.Create Constructor 5719
TCustomStringsValuesList.Destroy Destructor 5719
TCustomStringsValuesList Methods 5719
TCustomStringsValuesList Properties 5723
TCustomStringsValuesList Events 5724
TDefaultFieldsAdapter Class 5724
TDefaultFieldsAdapter Members 5725
TDefaultFieldsAdapter Methods 5729
TDefaultFieldsAdapter Properties 5732
TDefaultFieldsPagedAdapter Class 5733
TDefaultFieldsPagedAdapter Members 5735
TDefaultFieldsPagedAdapter Methods 5739
TDefaultFieldsPagedAdapter Properties 5746
TDefaultFieldsPagedAdapter Events 5748
TEndUserAdapter Class 5750
TEndUserAdapter Members 5751
TEndUserAdapter Properties 5756
TEndUserAdapter Events 5757
TEndUserAdapterLoginFormAction Class 5762
TEndUserAdapterLoginFormAction Members 5763
TEndUserAdapterLoginFormAction Methods 5766
TEndUserAdapterLoginFormAction Properties 5767
TEndUserAdapterLoginFormAction Events 5768
TEndUserAdapterLogoutAction Class 5770
TEndUserAdapterLogoutAction Members 5771

TEndUserAdapterLogoutAction Methods 5774
TEndUserAdapterLogoutAction Properties 5776
TEndUserAdapterLogoutAction Events 5776
TEndUserSessionAdapter Class 5779
TEndUserSessionAdapter Members 5780
TEndUserSessionAdapter Properties 5785
TEndUserSessionAdapter Events 5786
TFieldValuesAdapter Class 5789
TFieldValuesAdapter Members 5791
TFieldValuesAdapter.Create Constructor 5795
TFieldValuesAdapter Methods 5795
TFieldValuesAdapter Properties 5798
TFieldValuesValueField Class 5799
TFieldValuesValueField Members 5799
TFieldValuesValueField Methods 5801
TFieldValuesValueField Properties 5803
TImplementedAdapterAction Class 5803
TLogicalPageList Class 5805
TLogicalPageList Members 5805
TLogicalPageList.Create Constructor 5810
TLoginFormAdapter Class 5810
TLoginFormAdapter Members 5812
TLoginFormAdapter Properties 5816
TLoginFormAdapter Events 5818
TLoginFormAdapterLoginAction Class 5821
TLoginFormAdapterLoginAction Members 5822
TLoginFormAdapterLoginAction Methods 5826
TLoginFormAdapterLoginAction Properties 5828
TLoginFormAdapterLoginAction Events 5828
TPagedAdapter Class 5830
TPagedAdapter Members 5831
TPagedAdapter Properties 5836
TPagedAdapter Events 5838
TStringsValuesList Class 5843
TStringsValuesList Members 5844
TStringsValuesList Properties 5848
TStringsValuesList Events 5849
TUpdateFileList Class 5850
TUpdateFileList Members 5850

TUpdateFileList.Create Constructor 5850
TUpdateFileList Properties 5851
TValuesListNameField Class 5852
TValuesListNameField Members 5852
TValuesListNameField Methods 5855
TValuesListValueField Class 5855
TValuesListValueField Members 5856
TValuesListValueField Methods 5858

Interfaces 5860
IAdapterEditor Interface 5860
IAdapterEditor Members 5861
IAdapterEditor Methods 5861
IAdapterNotifyAdapterChange Interface 5861
IAdapterNotifyAdapterChange Members 5862
IAdapterNotifyAdapterChange Methods 5862
ICheckOrigValue Interface 5862
ICheckOrigValue Members 5863
ICheckOrigValue Methods 5863
ICheckValueChange Interface 5863
ICheckValueChange Members 5864
ICheckValueChange Methods 5864
IClearAdapterValues Interface 5864
IClearAdapterValues Members 5865
IClearAdapterValues Methods 5865
ICreateActionRequestContext Interface 5865
ICreateActionRequestContext Members 5865
ICreateActionRequestContext Methods 5866
IEchoActionFieldValue Interface 5866
IEchoActionFieldValue Members 5866
IEchoActionFieldValue Methods 5867
IEchoActionFieldValue Properties 5867
IEchoAdapterFieldValues Interface 5868
IEchoAdapterFieldValues Members 5868
IEchoAdapterFieldValues Methods 5868
IEchoAdapterFieldValues Properties 5869
IExecuteValue Interface 5869
IExecuteValue Members 5870
IExecuteValue Methods 5870
IFullyQualifyInputNames Interface 5871

IFullyQualifyInputNames Members 5871
IFullyQualifyInputNames Methods 5871
IGetAdapterErrorsList Interface 5872
IGetAdapterErrorsList Members 5872
IGetAdapterErrorsList Methods 5872
IGetAdapterModeNames Interface 5873
IGetAdapterModeNames Members 5873
IGetAdapterModeNames Methods 5873
IGetRecordKeys Interface 5874
IGetRecordKeys Members 5874
IGetRecordKeys Methods 5874
IIdentifyAdapter Interface 5875
ISetSubComponents Interface 5875
ISetSubComponents Members 5875
ISetSubComponents Methods 5876
ISetWebFieldValue Interface 5876
ISetWebFieldValue Members 5877
ISetWebFieldValue Methods 5877
IUpdateValue Interface 5877
IUpdateValue Members 5877
IUpdateValue Methods 5878
IValidateValue Interface 5878
IValidateValue Members 5878
IValidateValue Methods 5879
IWebActionsSubcomponent Interface 5879
IWebFieldsSubcomponent Interface 5879
IWebGetFieldPrevValue Interface 5880
IWebGetFieldPrevValue Members 5880
IWebGetFieldPrevValue Methods 5880
IWebGetFieldPrevValue Properties 5881

Functions **5883**

WebAdapt.AddDefaultActions Function 5883
WebAdapt.AddDefaultFields Function 5883
WebAdapt.FindAdapterAction Function 5884
WebAdapt.FindAdapterField Function 5884
WebAdapt.FindAdapterInParent Function 5884
WebAdapt.MakeAdapterRequestIdentifier Function 5885
WebAdapt.MakeValidAdapterFieldName Function 5885
WebAdapt.RaiseFileUploadNotSupported Function 5885

WebAdapt.RaiseMultipleFilesException Function 5886
WebAdapt.RaiseMultipleValuesException Function 5886

Structs, Records, Enums **5887**
WebAdapt.TAdapterHiddenFieldOption Enumeration 5887
WebAdapt.TIteratorMethod Enumeration 5887

Types **5889**
WebAdapt.IActionFieldValue Type 5890
WebAdapt.IActionRequest Type 5890
WebAdapt.TAbstractWebRequestFile Type 5890
WebAdapt.TActionExecuteEvent Type 5891
WebAdapt.TActionExecuteHandledEvent Type 5891
WebAdapt.TActionGetEnabledEvent Type 5891
WebAdapt.TActionGetParamsEvent Type 5892
WebAdapt.TActionRequestFieldsEvent Type 5892
WebAdapt.TActionRequestFieldsHandledEvent Type 5892
WebAdapt.TAdapterActionExecuteEvent Type 5893
WebAdapt.TAdapterActionExecuteHandledEvent Type 5893
WebAdapt.TAdapterActionGetParamsEvent Type 5893
WebAdapt.TAdapterActionsClass Type 5894
WebAdapt.TAdapterFieldBooleanValueEvent Type 5894
WebAdapt.TAdapterFieldFileEvent Type 5894
WebAdapt.TAdapterFieldFileListEvent Type 5895
WebAdapt.TAdapterFieldGetBooleanEvent Type 5895
WebAdapt.TAdapterFieldGetMultiValueEvent Type 5895
WebAdapt.TAdapterFieldGetMultiValueStringEvent Type 5896
WebAdapt.TAdapterFieldGetStringEvent Type 5896
WebAdapt.TAdapterFieldGetValueCountEvent Type 5896
WebAdapt.TAdapterFieldGetValueEvent Type 5897
WebAdapt.TAdapterFieldHasValueEvent Type 5897
WebAdapt.TAdapterFieldsClass Type 5897
WebAdapt.TAdapterFieldStringEvent Type 5898
WebAdapt.TAdapterFieldStringsEvent Type 5898
WebAdapt.TAdapterFieldValidateValueEvent Type 5898
WebAdapt.TAdapterFieldValueEvent Type 5899
WebAdapt.TAdapterHiddenFieldOptions Type 5899
WebAdapt.TEndUserGetNameEvent Type 5899
WebAdapt.TEndUserGetUserIDEvent Type 5900
WebAdapt.TEndUserHandledEvent Type 5900

WebAdapt.TEndUserIsLoggedInEvent Type 5900
WebAdapt.TEndUserRightsEvent Type 5901
WebAdapt.TEndUserUserIDEvent Type 5901
WebAdapt.TGetEOFEvent Type 5901
WebAdapt.TGetRecordCountEvent Type 5902
WebAdapt.TGetRecordEvent Type 5902
WebAdapt.TGetRecordIndexEvent Type 5902
WebAdapt.THandledEvent Type 5903
WebAdapt.TImageFieldGetHREFEvent Type 5903
WebAdapt.TImageFieldGetImageEvent Type 5903
WebAdapt.TImageFieldGetParamsEvent Type 5904
WebAdapt.TIterateRecordsEvent Type 5904
WebAdapt.TLoginUserIDEvent Type 5904
WebAdapt.TObjectList Type 5905
WebAdapt.TValidateAdapterFieldEvent Type 5905
WebAdapt.TWebFormGetPageNameEvent Type 5905

Variables **5906**
WebAdapt.NotifyPossibleHTMLChange Variable 5906

Constants **5907**
WebAdapt.sActionRequestLong Constant 5907
WebAdapt.sCurrentPage Constant 5907
WebAdapt.sGotoPageAction Constant 5908
WebAdapt.sImageRequest Constant 5908
WebAdapt.sLoginPassword Constant 5908
WebAdapt.sLoginUserName Constant 5909
WebAdapt.sNextPageAction Constant 5909
WebAdapt.sPrevPageAction Constant 5909

WebBroker Namespace 5911

Classes **5912**
TWebApplication Class 5912
TWebApplication Members 5913
TWebApplication.Create Constructor 5913
TWebApplication.Destroy Destructor 5914
TWebApplication Methods 5914
TWebApplication Properties 5915

Types **5917**

WebBroker.THandleShutdownException Type 5917
WebBroker.TServerExceptionEvent Type 5917

Variables **5918**
WebBroker.Application Variable 5918
WebBroker.HandleShutdownException Variable 5918

WebBrokerSOAP Namespace 5919

Classes **5920**
THTTPSoapDispatcher Class 5920
THTTPSoapDispatcher Members 5921
THTTPSoapDispatcher.Create Constructor 5921
THTTPSoapDispatcher.Destroy Destructor 5922
THTTPSoapDispatcher Methods 5922
THTTPSoapDispatcher Properties 5924

Functions **5925**
WebBrokerSOAP.GetSOAPWebModule Function 5925

WebCntxt Namespace 5927

Classes **5929**
TAbstractAdapterRequest Class 5929
TAbstractAdapterRequest Members 5930
TAbstractAdapterRequest.Create Constructor 5931
TAbstractAdapterRequest Properties 5931
TAbstractAdapterResponse Class 5932
TAbstractAdapterResponse Members 5932
TAbstractAdapterResponse.Create Constructor 5934
TAbstractAdapterResponse Properties 5934
TAbstractWebContext Class 5935
TAbstractWebContext Members 5935
TAbstractWebContext.Create Constructor 5937
TAbstractWebContext.Destroy Destructor 5937
TAbstractWebContext Methods 5938
TAbstractWebContext Properties 5945
TWebContext Class 5950
TWebContext Members 5951
TWebContext.Create Constructor 5952
TWebContext.Destroy Destructor 5953

TWebContext Methods 5954

Interfaces 5962

IWebEndUser Interface 5962

IWebEndUser Members 5963

IWebEndUser Methods 5963

IWebEndUser Properties 5968

IWebPageInfo Interface 5970

IWebPageInfo Members 5971

IWebPageInfo Methods 5971

IWebPageInfo Properties 5973

Functions 5977

WebCntxt.WebContext Function 5977

Structs, Records, Enums 5978

WebCntxt.TFindPageOption Enumeration 5978

Types 5979

WebCntxt.TFindPageOptions Type 5979

WebCntxt.TGetWebContextProc Type 5979

WebCntxt.TGetWebEndUserProc Type 5980

WebCntxt.TSetWebContextProc Type 5980

Variables 5981

WebCntxt.GetModuleFileNameProc Variable 5981

WebCntxt.GetWebContextProc Variable 5981

WebCntxt.GetWebEndUserProc Variable 5982

WebCntxt.SetWebContextProc Variable 5982

WebComp Namespace 5983

Classes 5985

TLayoutAttributes Class 5985

TLayoutAttributes Members 5986

TLayoutAttributes.Create Constructor 5987

TLayoutAttributes Fields 5987

TLayoutWebContent Class 5988

TLayoutWebContent Members 5989

TLayoutWebContent.Create Constructor 5990

TLayoutWebContent Methods 5990

TLayoutWebContent Properties 5996

TWebComponentClassDescription Class 5996
TWebComponentClassDescription Members 5997
TWebComponentClassDescription Fields 5998
TWebComponentList Class 5999
TWebComponentList Members 6000
TWebComponentList.Create Constructor 6001
TWebComponentList.Destroy Destructor 6001
TWebComponentList Methods 6001
TWebComponentList Properties 6007
TWebComponentList Events 6009
TWebComponentsEditorHelper Class 6009
TWebComponentsEditorHelper Members 6009
TWebComponentsEditorHelper Methods 6010
TWebContainedComponent Class 6012
TWebContainedComponent Members 6012
TWebContainedComponent.Create Constructor 6013
TWebContainedComponent.Destroy Destructor 6014
TWebContainedComponent Methods 6014
TWebContainedComponent Properties 6017
TWebContainedComponentHelper Class 6018
TWebContainedComponentHelper Members 6019
TWebContainedComponentHelper.Create Constructor 6020
TWebContainedComponentHelper.Destroy Destructor 6021
TWebContainedComponentHelper Methods 6021
TWebContainedComponentHelper Properties 6024
TWebContainedComponentHelper Events 6025
TWebContainedContainerComponent Class 6026
TWebContainedContainerComponent Members 6027
TWebContainedContainerComponent.Create Constructor 6028
TWebContainedContainerComponent.Destroy Destructor 6028
TWebContainedContainerComponent Methods 6028
TWebContainedContainerComponent Properties 6035
TWebContainerComponent Class 6036
TWebContainerComponent Members 6037
TWebContainerComponent.Create Constructor 6037
TWebContainerComponent.Destroy Destructor 6038
TWebContainerComponent Methods 6038
TWebContainerComponent Properties 6041
TWebContainerComponentHelper Class 6041

TWebContainerComponentHelper Members 6042
TWebContainerComponentHelper.Create Constructor 6043
TWebContainerComponentHelper.Destroy Destructor 6044
TWebContainerComponentHelper Methods 6044
TWebContainerComponentHelper Properties 6047
TWebContainerComponentHelper Events 6048
TWebContentOptions Class 6049
TWebContentOptions Members 6050
TWebContentOptions.Create Constructor 6050
TWebContentOptions Properties 6051

Interfaces 6052
IAddFieldsEditor Interface 6052
IAddFieldsEditor Members 6052
IAddFieldsEditor Methods 6053
IGetWebComponentList Interface 6054
IGetWebComponentList Members 6054
IGetWebComponentList Methods 6054
IGetWebComponentNodeName Interface 6055
IGetWebComponentNodeName Members 6055
IGetWebComponentNodeName Methods 6056
IGetXMLStream Interface 6056
IGetXMLStream Members 6056
IGetXMLStream Methods 6057
ILayoutWebContent Interface 6057
ILayoutWebContent Members 6058
ILayoutWebContent Methods 6058
INotifyActivateModule Interface 6059
INotifyActivateModule Members 6060
INotifyActivateModule Methods 6060
INotifyWebActivate Interface 6060
INotifyWebActivate Members 6061
INotifyWebActivate Methods 6061
IProducerEditorViewSupport Interface 6062
IProducerEditorViewSupport Members 6062
IProducerEditorViewSupport Methods 6062
ISetWebContentOptions Interface 6064
ISetWebContentOptions Members 6064
ISetWebContentOptions Methods 6065
ITopLevelWebComponent Interface 6065

IWebComponent Interface 6065
IWebComponent Members 6066
IWebComponent Methods 6066
IWebComponent Properties 6068
IWebComponentCommands Interface 6069
IWebComponentCommands Members 6070
IWebComponentCommands Methods 6070
IWebComponentContainer Interface 6071
IWebComponentContainer Members 6071
IWebComponentContainer Methods 6072
IWebComponentContainer Properties 6072
IWebComponentEditor Interface 6073
IWebComponentEditor Members 6073
IWebComponentEditor Methods 6074
IWebComponentEditorHelper Interface 6074
IWebComponentEditorHelper Members 6075
IWebComponentEditorHelper Methods 6075
IWebContent Interface 6075
IWebContent Members 6076
IWebContent Methods 6076
IWebVariablesContainerName Interface 6077
IWebVariablesContainerName Members 6077
IWebVariablesContainerName Methods 6077
IWebVariablesContainerName Properties 6078

Functions **6079**
WebComp.DesigningComponent Function 6079
WebComp.EnumRegisteredWebComponents Function 6079
WebComp.FindWebComponentEditorHelper Function 6080
WebComp.QualifyFileName Function 6080
WebComp.RegisterWebComponents Function 6080
WebComp.UnRegisterWebComponents Function 6081
WebComp.ValidateWebParent Function 6081

Structs, Records, Enums **6082**
WebComp.TLabelPosition Enumeration 6082
WebComp.TWebContentFlag Enumeration 6082

Types **6084**
WebComp.TEnumWebComponentsProc Type 6084
WebComp.TGetComponentListEvent Type 6084

WebComp.TLayout Type 6085
WebComp.TSetComponentListEvent Type 6085
WebComp.TWebComponentContainer Type 6085
WebComp.TWebContentFlags Type 6086

Constants 6087
WebComp.EnumRegisteredWebComponentsProc Constant 6087
WebComp.FindWebComponentEditorHelperProc Constant 6087
WebComp.RegisterWebComponentsProc Constant 6088
WebComp.UnRegisterWebComponentsProc Constant 6088

WebContnrs Namespace 6089

Classes 6090
ENamedVariantsError Class 6090
TAbstractNamedVariants Class 6090
TAbstractNamedVariants Members 6091
TAbstractNamedVariants.Destroy Destructor 6092
TAbstractNamedVariants Methods 6092
TAbstractNamedVariants Properties 6103
TNamedVariantsList Class 6106
TNamedVariantsList Members 6106
TNamedVariantsList.Destroy Destructor 6107
TNamedVariantsList Methods 6108

Structs, Records, Enums 6113
WebContnrs.TNamedVariantItem Record 6113

Types 6114
WebContnrs.PNamedVariantItem Type 6114
WebContnrs.PNamedVariantList Type 6114
WebContnrs.TNamedVariantList Type 6114

WebDisp Namespace 6117

Classes 6120
EAdapterDispatcherException Class 6120
EPageDispatcherException Class 6121
ESessionException Class 6122
TAdapterDispatcher Class 6122
TAdapterDispatcher Members 6123
TAdapterDispatcher Events 6124

TAdapterDispatchParams Class 6128
TAdapterDispatchParams Members 6128
TAdapterDispatchParams.Create Constructor 6130
TAdapterDispatchParams.Destroy Destructor 6130
TAdapterDispatchParams Methods 6131
TAdapterDispatchParams Properties 6133
TAdapterItemRequestParams Class 6134
TAdapterItemRequestParams Members 6135
TAdapterItemRequestParams.Create Constructor 6136
TAdapterItemRequestParams.Destroy Destructor 6137
TAdapterItemRequestParams Methods 6137
TAdapterItemRequestParams Properties 6139
TBaseRequestHandler Class 6140
TBaseRequestHandler Members 6140
TBaseRequestHandler Methods 6141
TBaseRequestHandler Properties 6142
TCustomAdapterDispatcher Class 6142
TCustomAdapterDispatcher Members 6143
TCustomAdapterDispatcher Methods 6144
TCustomAdapterDispatcher Events 6151
TCustomLocateFileService Class 6155
TCustomLocateFileService Members 6155
TCustomLocateFileService Methods 6156
TCustomLocateFileService Events 6160
TCustomPageDispatcher Class 6162
TCustomPageDispatcher Members 6162
TCustomPageDispatcher Methods 6163
TCustomPageDispatcher Properties 6169
TCustomPageDispatcher Events 6170
TCustomWebAppComponents Class 6175
TCustomWebAppComponents Members 6176
TCustomWebAppComponents Methods 6178
TCustomWebAppComponents Properties 6185
TCustomWebAppComponents Events 6189
TInvalidWebSession Class 6190
TInvalidWebSession Members 6190
TInvalidWebSession.Create Constructor 6191
TInvalidWebSession Methods 6191
TLocateFileService Class 6194

TLocateFileService Members 6195
TLocateFileService Events 6195
TMultiModuleWebAppServices Class 6197
TMultiModuleWebAppServices Members 6197
TMultiModuleWebAppServices Methods 6198
TMultiModuleWebAppServices Properties 6207
TMultiModuleWebAppServices Events 6209
TPageDispatcher Class 6210
TPageDispatcher Members 6210
TPageDispatcher Properties 6212
TPageDispatcher Events 6212
TWebAppComponents Class 6217
TWebAppComponents Members 6218
TWebAppComponents Properties 6219
TWebAppComponents Events 6223
TWebSession Class 6225
TWebSession Members 6225
Create Constructor 6226
TWebSession Methods 6226

Interfaces 6231
IWebSessionAccess Interface 6231
IWebSessionAccess Members 6231
IWebSessionAccess Methods 6231

Functions 6233
WebDisp.AdapterRequestParamsAsStrings Function 6233
WebDisp.DispatchPageName Function 6233
WebDisp.EncodeParamNameValues Function 6234
WebDisp.EncodeParamsAsHTTPQuery Function 6234
WebDisp.ExpandScriptException Function 6234
WebDisp.ExtractParamNameValues Function 6235
WebDisp.FindDefaultAppPageName Function 6235
WebDisp.PageNameOfRequest Function 6235
WebDisp.RaiseDispatchBlankPageNameException Function 6236
WebDisp.RaisePageDoesNotProvideContent Function 6236
WebDisp.RaisePageNotFound Function 6236
WebDisp.RedirectToPageName Function 6237
WebDisp.RegisterAdapterRequestIdentifier Function 6237
WebDisp.SafeRedirect Function 6237

Structs, Records, Enums **6239**
WebDisp.TPageAccessDenied Enumeration 6239

Types **6240**
WebDisp.TActivateWebModuleEvent Type 6240
WebDisp.TCanViewPageEvent Type 6240
WebDisp.TDispatchActionEvent Type 6241
WebDisp.TDispatchActionHandledEvent Type 6241
WebDisp.TDispatchImageEvent Type 6241
WebDisp.TDispatchImageHandledEvent Type 6242
WebDisp.TDispatchPageEvent Type 6242
WebDisp.TDispatchPageHandledEvent Type 6242
WebDisp.TDispatchPageParamsHandledEvent Type 6243
WebDisp.TGetAdapterRequestParamsEvent Type 6243
WebDisp.TLocateFileServiceFindFileEvent Type 6243
WebDisp.TLocateFileServiceFindStreamEvent Type 6244
WebDisp.TLoginPagedHandledEvent Type 6244
WebDisp.TLoginRequiredEvent Type 6244
WebDisp.TPageAccessDeniedEvent Type 6245
WebDisp.TWebRequest Type 6245
WebDisp.TWebResponse Type 6245

Constants **6247**
WebDisp.sFailurePageParamName Constant 6247
WebDisp.sPageParamName Constant 6247
WebDisp.sRequestIDParamName Constant 6248
WebDisp.sSessionCookieName Constant 6248
WebDisp.sSuccessPageParamName Constant 6248

WebFact Namespace 6249

Classes **6250**
TBaseWebModuleFactory Class 6250
TBaseWebModuleFactory Members 6251
TBaseWebModuleFactory.Create Constructor 6251
TBaseWebModuleFactory Methods 6252
TBaseWebPageInfo Class 6255
TBaseWebPageInfo Members 6255
TBaseWebPageInfo.Create Constructor 6256
TBaseWebPageInfo Methods 6256

TBaseWebPageModuleFactory Class 6260
TBaseWebPageModuleFactory Members 6261
TBaseWebPageModuleFactory.Create Constructor 6261
TBaseWebPageModuleFactory Methods 6262
TWebAppDataModuleFactory Class 6265
TWebAppDataModuleFactory Members 6265
TWebAppDataModuleFactory.Create Constructor 6266
TWebAppDataModuleFactory Methods 6266
TWebAppPageModuleFactory Class 6267
TWebAppPageModuleFactory Members 6268
TWebAppPageModuleFactory.Create Constructor 6268
TWebAppPageModuleFactory Methods 6268
TWebDataModuleFactory Class 6269
TWebDataModuleFactory Members 6270
TWebDataModuleFactory.Create Constructor 6270
TWebPageInfo Class 6271
TWebPageInfo Members 6271
TWebPageInfo.Create Constructor 6272
TWebPageModuleFactory Class 6273
TWebPageModuleFactory Members 6273
TWebPageModuleFactory.Create Constructor 6274

Structs, Records, Enums **6275**
WebFact.TWebPageAccessFlags Enumeration 6275

Types **6276**
WebFact.TWebPageAccess Type 6276

WebForm Namespace 6277

Classes **6280**
TAbstractControlRequirements Class 6281
TAbstractControlRequirements Members 6281
TAbstractControlRequirements Methods 6282
TAbstractFormRequirements Class 6284
TAbstractFormRequirements Members 6284
TAbstractFormRequirements Methods 6285
TAdapterActionButton Class 6286
TAdapterActionButton Members 6287
TAdapterActionButton Properties 6288
TAdapterActionsListHelper Class 6294

TAdapterActionsListHelper Members 6295
TAdapterActionsListHelper.Create Constructor 6296
TAdapterActionsListHelper Methods 6297
TAdapterActionsListHelper Properties 6301
TAdapterCommandColumn Class 6301
TAdapterCommandColumn Members 6302
TAdapterCommandColumn Properties 6303
TAdapterCommandGroup Class 6307
TAdapterCommandGroup Members 6308
TAdapterCommandGroup Properties 6310
TAdapterControlGroup Class 6312
TAdapterControlGroup Members 6313
TAdapterControlGroup.Create Constructor 6313
TAdapterControlGroup.Destroy Destructor 6314
TAdapterControlGroup Methods 6314
TAdapterControlGroup Properties 6322
TAdapterDataDisplay Class 6324
TAdapterDataDisplay Members 6325
TAdapterDataDisplay.Create Constructor 6326
TAdapterDataDisplay Methods 6326
TAdapterDataDisplay Properties 6331
TAdapterDisplayColumn Class 6333
TAdapterDisplayColumn Members 6334
TAdapterDisplayColumn Properties 6335
TAdapterDisplayField Class 6339
TAdapterDisplayField Members 6341
TAdapterDisplayField Properties 6342
TAdapterDisplayReferenceGroup Class 6349
TAdapterDisplayReferenceGroup Members 6350
TAdapterDisplayReferenceGroup.Create Constructor 6351
TAdapterDisplayReferenceGroup.Destroy Destructor 6351
TAdapterDisplayReferenceGroup Methods 6351
TAdapterDisplayReferenceGroup Properties 6355
TAdapterEditColumn Class 6355
TAdapterEditColumn Members 6356
TAdapterEditColumn Properties 6358
TAdapterErrorList Class 6365
TAdapterErrorList Members 6366
TAdapterErrorList Properties 6367

TAdapterFieldGroup Class 6367
TAdapterFieldGroup Members 6369
TAdapterFieldGroup Properties 6370
TAdapterForm Class 6373
TAdapterForm Members 6374
TAdapterForm Properties 6375
TAdapterFormLayout Class 6376
TAdapterFormLayout Members 6377
TAdapterFormLayout.Create Constructor 6378
TAdapterFormLayout Methods 6379
TAdapterFormLayout Properties 6385
TAdapterGrid Class 6386
TAdapterGrid Members 6388
TAdapterGrid Properties 6389
TControlRequirements Class 6392
TControlRequirements Members 6393
TControlRequirements.Create Constructor 6393
TControlRequirements.Destroy Destructor 6394
TControlRequirements Methods 6394
TControlRequirements Properties 6396
TCustomAdapterActionButton Class 6397
TCustomAdapterActionButton Members 6398
TCustomAdapterActionButton.Create Constructor 6398
TCustomAdapterActionButton.Destroy Destructor 6399
TCustomAdapterActionButton Methods 6399
TCustomAdapterActionButton Properties 6404
TCustomAdapterCommandColumn Class 6409
TCustomAdapterCommandColumn Members 6410
TCustomAdapterCommandColumn.Create Constructor 6411
TCustomAdapterCommandColumn.Destroy Destructor 6412
TCustomAdapterCommandColumn Methods 6412
TCustomAdapterCommandColumn Properties 6416
TCustomAdapterCommandGroup Class 6421
TCustomAdapterCommandGroup Members 6421
TCustomAdapterCommandGroup.Create Constructor 6423
TCustomAdapterCommandGroup.Destroy Destructor 6423
TCustomAdapterCommandGroup Methods 6423
TCustomAdapterCommandGroup Properties 6425
TCustomAdapterDisplayColumn Class 6428

TCustomAdapterDisplayColumn Members 6428
TCustomAdapterDisplayColumn.Create Constructor 6430
TCustomAdapterDisplayColumn Methods 6430
TCustomAdapterDisplayColumn Properties 6433
TCustomAdapterDisplayField Class 6434
TCustomAdapterDisplayField Members 6435
TCustomAdapterDisplayField.Create Constructor 6436
TCustomAdapterDisplayField Methods 6436
TCustomAdapterDisplayField Properties 6438
TCustomAdapterEditColumn Class 6442
TCustomAdapterEditColumn Members 6443
TCustomAdapterEditColumn Methods 6444
TCustomAdapterErrorList Class 6445
TCustomAdapterErrorList Members 6445
TCustomAdapterErrorList.Create Constructor 6446
TCustomAdapterErrorList.Destroy Destructor 6446
TCustomAdapterErrorList Methods 6447
TCustomAdapterErrorList Properties 6448
TCustomAdapterFieldGroup Class 6449
TCustomAdapterFieldGroup Members 6450
TCustomAdapterFieldGroup.Create Constructor 6451
TCustomAdapterFieldGroup.Destroy Destructor 6451
TCustomAdapterFieldGroup Methods 6452
TCustomAdapterFieldGroup Properties 6455
TCustomAdapterForm Class 6456
TCustomAdapterForm Members 6457
TCustomAdapterForm.Create Constructor 6457
TCustomAdapterForm.Destroy Destructor 6458
TCustomAdapterForm Methods 6458
TCustomAdapterForm Properties 6461
TCustomAdapterGrid Class 6462
TCustomAdapterGrid Members 6464
TCustomAdapterGrid.Create Constructor 6465
TCustomAdapterGrid.Destroy Destructor 6465
TCustomAdapterGrid Methods 6466
TCustomAdapterGrid Properties 6468
TDesigntimeWarnings Class 6470
TDesigntimeWarnings Members 6470
TDesigntimeWarnings.Create Constructor 6470

TDesigntimeWarnings.Destroy Destructor 6470
TDesigntimeWarnings Methods 6471
TDesigntimeWarnings Properties 6472
TGeneratedFunctions Class 6472
TGeneratedFunctions Members 6472
TGeneratedFunctions.Create Constructor 6473
TGeneratedFunctions.Destroy Destructor 6473
TGeneratedFunctions Methods 6473
TGeneratedFunctions Properties 6474

Interfaces 6476
IAdapterDisplay Interface 6476
IGetAdapterFormLayout Interface 6476
IGetAdapterFormLayout Members 6477
IGetAdapterFormLayout Methods 6477
IGetAdapterModeOfDisplay Interface 6477
IGetAdapterModeOfDisplay Members 6478
IGetAdapterModeOfDisplay Methods 6478
IGetAdapterOfDisplay Interface 6478
IGetAdapterOfDisplay Members 6479
IGetAdapterOfDisplay Methods 6479
IGetAdapterVariableName Interface 6479
IGetAdapterVariableName Members 6480
IGetAdapterVariableName Methods 6480
IGetAdapterVariableName Properties 6480
IGetControlRequirements Interface 6481
IGetControlRequirements Members 6481
IGetControlRequirements Methods 6481
IGetFormRequirements Interface 6482
IGetFormRequirements Members 6482
IGetFormRequirements Methods 6482
IGetVariablesContainerOfDisplay Interface 6483
IGetVariablesContainerOfDisplay Members 6483
IGetVariablesContainerOfDisplay Methods 6483

Functions 6485
WebForm.CalcDisplayFieldHTMLElementType Function 6485
WebForm.CalcDisplayFieldViewMode Function 6485
WebForm.FindVariablesContainer Function 6486
WebForm.FindVariablesContainerInParent Function 6486

WebForm.GetAdapterModeOfAdapterInParent Function 6486
WebForm.MakeAdapterHTMLName Function 6487
WebForm.MakeAdapterVariableName Function 6487

Structs, Records, Enums **6488**
WebForm.TActionButtonHideOption Enumeration 6488
WebForm.TColumnHTMLElementType Enumeration 6489
WebForm.TCommandHTMLElementType Enumeration 6489
WebForm.TDisplayFieldHideOption Enumeration 6490
WebForm.TDisplayFieldHTMLElementType Enumeration 6491
WebForm.TDisplayFieldViewMode Enumeration 6491
WebForm.TInputFieldHTMLElementType Enumeration 6492
WebForm.TLayoutState Enumeration 6493

Types **6494**
WebForm.TActionButtonHideOptions Type 6494
WebForm.TAdapterDataDisplayClass Type 6494
WebForm.TDisplayFieldHideOptions Type 6494

Constants **6496**
WebForm.sDisplayText Constant 6496
WebForm.sFieldText Constant 6496
WebForm.sFormGet Constant 6497
WebForm.sFormPost Constant 6497
WebForm.sMultipartForm Constant 6497

WebModu Namespace 6499

Classes **6501**
EWebModuleException Class 6501
TAdapterActionName Class 6502
TAdapterActionName Members 6502
TAdapterActionName.Create Constructor 6503
TAdapterActionName Methods 6503
TAdapterActionName Properties 6504
TCustomWebAppDataModule Class 6505
TCustomWebAppDataModule Members 6506
TCustomWebAppDataModule.Create Constructor 6507
TCustomWebAppDataModule.CreateNew Constructor 6507
TCustomWebAppDataModule.Destroy Destructor 6508
TCustomWebAppDataModule Methods 6508

TCustomWebAppDataModule Properties 6509
TCustomWebAppPageModule Class 6510
TCustomWebAppPageModule Members 6511
TCustomWebAppPageModule.Create Constructor 6512
TCustomWebAppPageModule.CreateNew Constructor 6513
TCustomWebAppPageModule.Destroy Destructor 6513
TCustomWebAppPageModule Methods 6513
TCustomWebAppPageModule Properties 6515
TCustomWebDataModule Class 6516
TCustomWebDataModule Members 6517
TCustomWebDataModule.Create Constructor 6517
TCustomWebDataModule.CreateNew Constructor 6518
TCustomWebDataModule.Destroy Destructor 6518
TCustomWebDataModule Methods 6519
TCustomWebDataModule Properties 6520
TCustomWebDataModule Events 6521
TCustomWebPageModule Class 6522
TCustomWebPageModule Members 6523
TCustomWebPageModule.Create Constructor 6525
TCustomWebPageModule.CreateNew Constructor 6525
TCustomWebPageModule.Destroy Destructor 6526
TCustomWebPageModule Methods 6526
TCustomWebPageModule Properties 6527
TCustomWebPageModule Events 6529
TSiteModuleHelper Class 6531
TSiteModuleHelper Members 6532
TSiteModuleHelper.Create Constructor 6533
TSiteModuleHelper.Destroy Destructor 6533
TSiteModuleHelper Methods 6534
TSiteModuleHelper Properties 6537
TSiteModuleHelper Events 6539
TSitePageModuleHelper Class 6539
TSitePageModuleHelper Members 6540
TSitePageModuleHelper.Create Constructor 6542
TSitePageModuleHelper.Destroy Destructor 6542
TSitePageModuleHelper Methods 6543
TSitePageModuleHelper Properties 6548
TSitePageModuleHelper Events 6549
TWebAppDataModule Class 6550

TWebAppDataModule Members 6551
TWebAppDataModule Properties 6552
TWebAppDataModule Events 6553
TWebAppPageModule Class 6554
TWebAppPageModule Members 6554
TWebAppPageModule Properties 6556
TWebAppPageModule Events 6558
TWebAppPageModuleHelper Class 6560
TWebAppPageModuleHelper Members 6560
TWebAppPageModuleHelper Methods 6562
TWebDataModule Class 6562
TWebDataModule Members 6563
TWebDataModule Events 6564
TWebNotifyContainer Class 6565
TWebNotifyContainer Members 6565
TWebNotifyContainer.Create Constructor 6566
TWebNotifyContainer.Destroy Destructor 6566
TWebNotifyContainer Fields 6567
TWebNotifyContainer Methods 6567
TWebPageModule Class 6569
TWebPageModule Members 6570
TWebPageModule Properties 6571
TWebPageModule Events 6572
TWebPageModuleHelper Class 6574
TWebPageModuleHelper Members 6574
TWebPageModuleHelper Methods 6576

Functions **6578**
WebModu.FindNamedModuleVariable Function 6578
WebModu.GetModuleVariable Function 6578
WebModu.GetModuleVariableCount Function 6578

Types **6580**
WebModu.TGetParamsEvent Type 6580

WebNode Namespace 6581

Interfaces **6582**
IWebNode Interface 6582
IWebNode Members 6583
IWebNode Methods 6583

IWebNode Properties 6585

WebReq Namespace 6587

Classes **6588**

TWebModuleFactoryList Class 6588

TWebModuleFactoryList Members 6588

TWebModuleFactoryList.Create Constructor 6589

TWebModuleFactoryList.Destroy Destructor 6590

TWebModuleFactoryList Methods 6590

TWebModuleFactoryList Properties 6591

TWebModuleList Class 6592

TWebModuleList Members 6593

TWebModuleList.Create Constructor 6593

TWebModuleList.Destroy Destructor 6594

TWebModuleList Methods 6594

TWebModuleList Properties 6599

TWebRequestHandler Class 6600

TWebRequestHandler Members 6600

TWebRequestHandler.Create Constructor 6601

TWebRequestHandler.Destroy Destructor 6601

TWebRequestHandler Methods 6602

TWebRequestHandler Properties 6604

Functions **6608**

WebReq.WebRequestHandler Function 6608

Structs, Records, Enums **6609**

WebReq.TRequestNotification Enumeration 6609

Variables **6610**

WebReq.WebRequestHandlerProc Variable 6610

WebScript Namespace 6611

Classes **6613**

ERecursiveIncludeFile Class 6613

EScriptEvalError Class 6614

TAbstractScriptEngine Class 6614

TAbstractScriptEngine Members 6614

TAbstractScriptEngine.Create Constructor 6615

TAbstractScriptEngine.Destroy Destructor 6615

TAbstractScriptEngine Methods 6616
TAbstractScriptEngine Properties 6617
TAbstractScriptObjectFactories Class 6618
TAbstractScriptObjectFactories Members 6618
TAbstractScriptObjectFactories Methods 6619
TAbstractScriptObjectFactories Properties 6620
TAbstractScriptObjectFactory Class 6621
TAbstractScriptObjectFactory Members 6622
TAbstractScriptObjectFactory Methods 6622
TCurrentWebPageInfo Class 6623
TCurrentWebPageInfo Members 6624
TCurrentWebPageInfo.Create Constructor 6625
TCurrentWebPageInfo Methods 6626
THTMLItem Class 6628
THTMLItem Members 6629
THTMLItem.Create Constructor 6630
THTMLItem Methods 6630
THTMLItem Properties 6630
THTMLItems Class 6631
THTMLItems Members 6632
THTMLItems.Create Constructor 6632
THTMLItems.Destroy Destructor 6632
THTMLItems Methods 6633
THTMLItems Properties 6633
TLineNumberMap Class 6634
TLineNumberMap Members 6635
TLineNumberMap Methods 6636
TLineNumberMap Properties 6636
TScriptCodeItem Class 6637
TScriptCodeItem Members 6637
TScriptCodeItem Methods 6638
TScriptEnginesList Class 6639
TScriptEnginesList Members 6639
TScriptEnginesList.Create Constructor 6640
TScriptEnginesList.Destroy Destructor 6640
TScriptEnginesList Methods 6641
TScriptEnginesList Properties 6642
TScriptError Class 6643
TScriptError Members 6643

TScriptError.Create Constructor 6644
TScriptError Fields 6644
TScriptError Methods 6645
TScriptErrors Class 6647
TScriptErrors Members 6647
TScriptErrors.Create Constructor 6647
TScriptErrors.Destroy Destructor 6648
TScriptErrors Methods 6648
TScriptFile Class 6649
TScriptFile Members 6650
TScriptFile.Create Constructor 6650
TScriptFile.Destroy Destructor 6650
TScriptFile Fields 6651
TScriptFile Methods 6651
TScriptFile Properties 6652
TScriptGlobalObjects Class 6653
TScriptGlobalObjects Members 6653
TScriptGlobalObjects.Create Constructor 6654
TScriptGlobalObjects.Destroy Destructor 6655
TScriptGlobalObjects Methods 6655
TScriptGlobalObjects Properties 6657
TScriptIncludeFile Class 6658
TScriptIncludeFile Members 6658
TScriptIncludeFile.Create Constructor 6659
TScriptIncludeFile Methods 6659
TScriptIncludeFiles Class 6659
TScriptIncludeFiles Members 6660
TScriptIncludeFiles.Create Constructor 6660
TScriptIncludeFiles Methods 6660
TScriptIncludeFiles Properties 6661
TScriptIncludeItem Class 6662
TScriptIncludeItem Members 6662
TScriptIncludeItem.Create Constructor 6663
TScriptIncludeItem Methods 6663
TScriptItem Class 6664
TScriptItem Members 6664
TScriptItem.Create Constructor 6665
TScriptItem Methods 6665
TScriptItem Properties 6665

TScriptItems Class 6666
TScriptItems Members 6666
TScriptItems.Create Constructor 6667
TScriptItems Methods 6667
TScriptItems Properties 6667
TScriptObjectFactories Class 6668
TScriptObjectFactories Members 6669
TScriptObjectFactories.Create Constructor 6670
TScriptObjectFactories.Destroy Destructor 6670
TScriptObjectFactories Methods 6671
TScriptObjectFactories Properties 6672
TScriptProducer Class 6673
TScriptProducer Members 6673
TScriptProducer.Create Constructor 6674
TScriptProducer.Destroy Destructor 6674
TScriptProducer Fields 6675
TScriptProducer Methods 6676
TScriptProducer Properties 6680
TVariableLookup Class 6682
TVariableLookup Members 6682
TVariableLookup.Create Constructor 6683
TVariableLookup Methods 6683
TVariableLookup Properties 6684

Structs, Records, Enums **6686**
WebScript.TGlobalScriptVariables Enumeration 6686
WebScript.TIncludeType Enumeration 6687
WebScript.TLineNumbers Record 6687

Types **6688**
WebScript.CreateComponentStubFactory Type 6688
WebScript.PLineNumbers Type 6688

Constants **6689**
WebScript.GlobalScriptVariableNames Constant 6689

WebServExp Namespace 6691

Classes **6693**
TWebServExp Class 6693
TWebServExp Members 6693

TWebServExp.Create Constructor 6695
TWebServExp.Destroy Destructor 6695
TWebServExp Methods 6696
TWebServExp Properties 6697
TWebServExp Events 6699

Interfaces 6701
IWebServExp Interface 6701
IWebServExp Members 6701
IWebServExp Methods 6702
IWebServExpAccess Interface 6703
IWebServExpAccess Members 6703
IWebServExpAccess Methods 6703

Functions 6704
WebServExp.GetAliasBaseTypeInfo Function 6704
WebServExp.GetBindingType Function 6704
WebServExp.IsBaseClassTypeInfo Function 6705

Structs, Records, Enums 6706
WebServExp.ArgumentType Enumeration 6706
WebServExp.MessageType Enumeration 6706
WebServExp.TSchemaType Record 6707

Types 6708
WebServExp.TAfterPublishingWSDLEvent Type 6708
WebServExp.TBeforePublishingTypesEvent Type 6708
WebServExp.TPublishingTypeEvent Type 6709
WebServExp.TSchemaTypeArray Type 6709

WebSess Namespace 6711

Classes 6712
TCustomSessionsService Class 6712
TCustomSessionsService Members 6713
TCustomSessionsService.Create Constructor 6713
TCustomSessionsService Methods 6714
TCustomSessionsService Properties 6720
TCustomSessionsService Events 6721
TSessionsService Class 6722
TSessionsService Members 6723
TSessionsService Properties 6724

TSessionsService Events 6725
TWebSessionIDsImpl Class 6726
TWebSessionIDsImpl Members 6726
TWebSessionIDsImpl.Create Constructor 6727
TWebSessionIDsImpl.Destroy Destructor 6728
TWebSessionIDsImpl Methods 6728
TWebSessionImpl Class 6729
TWebSessionImpl Members 6730
TWebSessionImpl.Create Constructor 6731
TWebSessionImpl.Destroy Destructor 6731
TWebSessionImpl Methods 6732
TWebSessionImpl Properties 6736

Structs, Records, Enums **6739**
WebSess.TEndSessionReason Enumeration 6739

Types **6740**
WebSess.TEndSessionEvent Type 6740
WebSess.TStartSessionEvent Type 6740

WebUsers Namespace 6741

Classes **6742**
EUserIDNotFoundException Class 6742
EValidateUserException Class 6742
EWebUsersListException Class 6743
TCustomWebUserList Class 6744
TCustomWebUserList Members 6744
TCustomWebUserList.Create Constructor 6745
TCustomWebUserList.Destroy Destructor 6745
TCustomWebUserList Methods 6746
TCustomWebUserList Properties 6747
TCustomWebUserList Events 6748
TWebUserItem Class 6750
TWebUserItem Members 6751
TWebUserItem.Create Constructor 6751
TWebUserItem.Destroy Destructor 6752
TWebUserItem Methods 6752
TWebUserItem Properties 6753
TWebUserItems Class 6755
TWebUserItems Members 6755

TWebUserItems.Create Constructor 6756
TWebUserItems Methods 6756
TWebUserItems Properties 6759
TWebUserList Class 6759
TWebUserList Members 6760
TWebUserList Properties 6760
TWebUserList Events 6761

Structs, Records, Enums **6764**
WebUsers.TValidateUserError Enumeration 6764

Types **6765**
WebUsers.TCheckAccessRightsEvent Type 6765
WebUsers.TCheckAccessRightsHandledEvent Type 6765
WebUsers.TValidateUserErrorEvent Type 6766
WebUsers.TValidateUserEvent Type 6766
WebUsers.TValidateUserHandledEvent Type 6766

WideStrings Namespace 6767

Classes **6768**
TWideStringList Class 6768
TWideStringList Members 6769
Create Constructor 6770
TWideStringList.Destroy Destructor 6771
TWideStringList Methods 6772
TWideStringList Properties 6782
TWideStringList Events 6784
TWideStrings Class 6785
TWideStrings Members 6786
TWideStrings.Destroy Destructor 6788
TWideStrings Methods 6788
TWideStrings Properties 6806
TWideStringsEnumerator Class 6816
TWideStringsEnumerator Members 6816
TWideStringsEnumerator.Create Constructor 6817
TWideStringsEnumerator Methods 6817
TWideStringsEnumerator Properties 6818

Interfaces **6819**
IWideStringsAdapter Interface 6819

IWideStringsAdapter Members 6819
IWideStringsAdapter Methods 6819

Structs, Records, Enums **6821**
WideStrings.TWideStringItem Record 6821

Types **6822**
WideStrings.PWideStringItem Type 6822
WideStrings.PWideStringItemList Type 6822
WideStrings.TWideStringItemList Type 6823
WideStrings.TWideStringListSortCompare Type 6823

WideStrUtils Namespace 6825

Functions **6827**
WideStrUtils.AnsiToUtf8Ex Function 6828
WideStrUtils.ConvertStreamFromAnsiToUTF8 Function 6828
WideStrUtils.ConvertStreamFromUTF8ToAnsi Function 6828
WideStrUtils.DetectUTF8Encoding Function 6829
WideStrUtils.HasExtendCharacter Function 6829
WideStrUtils.HasUTF8BOM Function 6829
WideStrUtils.HasUTF8BOM Function 6830
WideStrUtils.inOpArray Function 6830
WideStrUtils.inOpSet Function 6830
WideStrUtils.IsUTF8LeadByte Function 6831
WideStrUtils.IsUTF8String Function 6831
WideStrUtils.IsUTF8TrailByte Function 6831
WideStrUtils.LoadResWideString Function 6832
WideStrUtils.LoadWideStr Function 6832
WideStrUtils.UTF8CharLength Function 6832
WideStrUtils.UTF8CharSize Function 6833
WideStrUtils.UTF8LowerCase Function 6833
WideStrUtils.Utf8ToAnsiEx Function 6833
WideStrUtils.UTF8UpperCase Function 6834
WideStrUtils.WideAdjustLineBreaks Function 6834
WideStrUtils.WideDequotedStr Function 6834
WideStrUtils.WideExtractQuotedStr Function 6835
WideStrUtils.WideLastChar Function 6835
WideStrUtils.WideQuotedStr Function 6835
WideStrUtils.WideReplaceStr Function 6836
WideStrUtils.WideReplaceText Function 6836

WideStrUtils.WideStringReplace Function 6836
WideStrUtils.WStrAlloc Function 6837
WideStrUtils.WStrBufSize Function 6837
WideStrUtils.WStrCat Function 6837
WideStrUtils.WStrComp Function 6838
WideStrUtils.WStrCopy Function 6838
WideStrUtils.WStrDispose Function 6838
WideStrUtils.WStrEnd Function 6839
WideStrUtils.WStrLCopy Function 6839
WideStrUtils.WStrLen Function 6839
WideStrUtils.WStrMove Function 6840
WideStrUtils.WStrNew Function 6840
WideStrUtils.WStrPCopy Function 6840
WideStrUtils.WStrPLCopy Function 6841
WideStrUtils.WStrPos Function 6841
WideStrUtils.WStrScan Function 6841

Structs, Records, Enums **6842**
WideStrUtils.TEncodeType Enumeration 6842

Types **6843**
WideStrUtils.CharSet Type 6843

Constants **6844**
WideStrUtils.sUTF8BOMString Constant 6844

WinHelpViewer Namespace 6845

Interfaces **6846**
IWinHelpTester Interface 6846
IWinHelpTester Members 6846
IWinHelpTester Methods 6846

Variables **6849**
WinHelpViewer.ViewerName Variable 6849
WinHelpViewer.WinHelpTester Variable 6849

WSDLBind Namespace 6851

Classes **6854**
TBinding Class 6854
TBinding Members 6855

TBinding Methods 6859
TBindingOperation Class 6861
TBindingOperation Members 6862
TBindingOperation Methods 6867
TBindingOperations Class 6869
TBindingOperations Members 6869
TBindingOperations Methods 6874
TBindings Class 6875
TBindings Members 6875
TBindings Methods 6881
TDefinition Class 6882
TDefinition Members 6882
TDefinition Methods 6887
TDocumentation Class 6890
TDocumentationCollection Class 6891
TDocumentationCollection Members 6891
TDocumentationCollection Methods 6896
TDocumented Class 6897
TDocumented Members 6897
TDocumented Methods 6902
TFault Class 6903
TFaults Class 6904
TFaults Members 6904
TFaults Methods 6909
TImport Class 6910
TImport Members 6910
TImport Methods 6915
TImports Class 6916
TImports Members 6917
TImports Methods 6922
TMessage Class 6923
TMessage Members 6923
TMessage Methods 6928
TMessages Class 6930
TMessages Members 6930
TMessages Methods 6935
TOperation Class 6936
TOperation Members 6936
TOperation Methods 6941

TOperations Class 6943
TOperations Members 6944
TOperations Methods 6949
TParam Class 6950
TParam Members 6951
TParam Methods 6955
TPart Class 6957
TPart Members 6957
TPart Methods 6962
TParts Class 6964
TParts Members 6964
TParts Methods 6969
TPort Class 6970
TPort Members 6971
TPort Methods 6975
TPorts Class 6977
TPorts Members 6977
TPorts Methods 6982
TPortType Class 6983
TPortType Members 6983
TPortType Methods 6988
TPortTypes Class 6990
TPortTypes Members 6990
TPortTypes Methods 6995
TService Class 6996
TService Members 6996
TService Methods 7001
TServices Class 7003
TServices Members 7003
TServices Methods 7008
TTypes Class 7009
TTypes Members 7009
TTypes Methods 7014
TWSDLDocument Class 7015
TWSDLDocument Members 7016
TWSDLDocument Methods 7020
TWSDLDocument Properties 7021
TXMLSchemaDefs Class 7022
TXMLSchemaDefs Members 7023

TXMLSchemaDefs Methods 7028

Interfaces 7029

IBinding Interface 7029

IBinding Members 7030

IBinding Methods 7031

IBinding Properties 7032

IBindingInput Interface 7033

IBindingOperation Interface 7034

IBindingOperation Members 7034

IBindingOperation Methods 7036

IBindingOperation Properties 7037

IBindingOperations Interface 7038

IBindingOperations Members 7038

IBindingOperations Methods 7040

IBindingOperations Properties 7041

IBindingOutput Interface 7041

IBindings Interface 7042

IBindings Members 7042

IBindings Methods 7043

IBindings Properties 7044

IDefinition Interface 7045

IDefinition Members 7045

IDefinition Methods 7047

IDefinition Properties 7049

IDocumentation Interface 7051

IDocumentationCollection Interface 7052

IDocumentationCollection Members 7052

IDocumentationCollection Methods 7054

IDocumentationCollection Properties 7054

IDocumented Interface 7055

IDocumented Members 7055

IDocumented Methods 7056

IDocumented Properties 7057

IFault Interface 7057

IFaults Interface 7058

IFaults Members 7058

IFaults Methods 7060

IFaults Properties 7060

IImport Interface 7061

IImport Members 7061
IImport Methods 7063
IImport Properties 7064
IImports Interface 7064
IImports Members 7065
IImports Methods 7066
IImports Properties 7067
IMessage Interface 7067
IMessage Members 7068
IMessage Methods 7069
IMessage Properties 7070
IMessages Interface 7071
IMessages Members 7071
IMessages Methods 7073
IMessages Properties 7073
IOperation Interface 7074
IOperation Members 7074
IOperation Methods 7076
IOperation Properties 7077
IOperations Interface 7078
IOperations Members 7078
IOperations Methods 7080
IOperations Properties 7081
IParam Interface 7082
IParam Members 7082
IParam Methods 7083
IParam Properties 7085
IPart Interface 7085
IPart Members 7086
IPart Methods 7087
IPart Properties 7089
IParts Interface 7090
IParts Members 7090
IParts Methods 7091
IParts Properties 7092
IPort Interface 7093
IPort Members 7093
IPort Methods 7095
IPort Properties 7096

IPorts Interface 7096
IPorts Members 7097
IPorts Methods 7098
IPorts Properties 7099
IPortType Interface 7099
IPortType Members 7100
IPortType Methods 7101
IPortType Properties 7102
IPortTypes Interface 7103
IPortTypes Members 7103
IPortTypes Methods 7105
IPortTypes Properties 7105
IService Interface 7106
IService Members 7106
IService Methods 7108
IService Properties 7108
IServices Interface 7109
IServices Members 7109
IServices Methods 7111
IServices Properties 7112
ITypes Interface 7112
ITypes Members 7113
ITypes Methods 7114
ITypes Properties 7114
IWSDLDocument Interface 7115
IWSDLDocument Members 7115
IWSDLDocument Methods 7117
IWSDLDocument Properties 7118
IXMLSchemaDefs Interface 7119
IXMLSchemaDefs Members 7119
IXMLSchemaDefs Methods 7120
IXMLSchemaDefs Properties 7121

Functions 7122

WSDLBind.GetDefinition Function 7122
WSDLBind.LoadWSDLDoc Function 7122
WSDLBind.NewWSDLDoc Function 7122

Constants 7124

WSDLBind.SBinding Constant 7124

WSDLBind.SDefinitions Constant 7124
WSDLBind.SFault Constant 7125
WSDLBind.SImport Constant 7125
WSDLBind.SInput Constant 7125
WSDLBind.SLocation Constant 7126
WSDLBind.SMessage Constant 7126
WSDLBind.SNamespace Constant 7126
WSDLBind.SOperation Constant 7127
WSDLBind.SOutput Constant 7127
WSDLBind.SPart Constant 7127
WSDLBind.SPort Constant 7128
WSDLBind.SPortType Constant 7128
WSDLBind.SService Constant 7128
WSDLBind.SType Constant 7129
WSDLBind.STypes Constant 7129

WSDLIntf Namespace 7131

Classes **7133**
TDOMStrings Class 7133
TDOMStrings Members 7133
TDOMStrings.Create Constructor 7134
TDOMStrings.Destroy Destructor 7134
TDOMStrings Methods 7134
TDOMStrings Properties 7136

Structs, Records, Enums **7137**
WSDLIntf.TWebServiceBindingType Enumeration 7137
WSDLIntf.TWSDLElement Enumeration 7137
WSDLIntf.TWString Record 7138

Types **7139**
WSDLIntf.TWSDLElements Type 7139

Constants **7140**
WSDLIntf.Directives Constant 7141
WSDLIntf.DynArrayTypes Constant 7141
WSDLIntf.Httpns Constant 7142
WSDLIntf.Operators Constant 7142
WSDLIntf.OtherReservedWords Constant 7142
WSDLIntf.ReservedWords Constant 7143

WSDLIntf.SAddress Constant 7143
WSDLIntf.SAnySimpleType Constant 7144
WSDLIntf.SAnyType Constant 7144
WSDLIntf.SArray Constant 7144
WSDLIntf.SArrayOf Constant 7145
WSDLIntf.SArrayType Constant 7145
WSDLIntf.SBinding Constant 7145
WSDLIntf.SBody Constant 7146
WSDLIntf.ScalarArrayTypes Constant 7146
WSDLIntf.SComplexType Constant 7146
WSDLIntf.SDefinitions Constant 7147
WSDLIntf.SDynArray Constant 7147
WSDLIntf.SElement Constant 7147
WSDLIntf.SEncodingStyle Constant 7148
WSDLIntf.SFault Constant 7148
WSDLIntf.SHeader Constant 7148
WSDLIntf.SImport Constant 7149
WSDLIntf.SInOut Constant 7149
WSDLIntf.SInput Constant 7149
WSDLIntf.SLocation Constant 7150
WSDLIntf.SMessage Constant 7150
WSDLIntf.SName Constant 7150
WSDLIntf.SNameSpace Constant 7151
WSDLIntf.SNsPrefix Constant 7151
WSDLIntf.Soapns Constant 7151
WSDLIntf.SOperation Constant 7152
WSDLIntf.SOutput Constant 7152
WSDLIntf.SPart Constant 7152
WSDLIntf.SParts Constant 7153
WSDLIntf.SPort Constant 7153
WSDLIntf.SPortType Constant 7153
WSDLIntf.SRequest Constant 7154
WSDLIntf.SResponse Constant 7154
WSDLIntf.SReturn Constant 7154
WSDLIntf.SSchema Constant 7155
WSDLIntf.SService Constant 7155
WSDLIntf.SSoapAction Constant 7155
WSDLIntf.SStyle Constant 7156
WSDLIntf.STns Constant 7156

WSDLIntf.STransport Constant 7156
WSDLIntf.SType Constant 7157
WSDLIntf.STypes Constant 7157
WSDLIntf.SUnknown Constant 7157
WSDLIntf.SUse Constant 7158
WSDLIntf.SVerb Constant 7158
WSDLIntf.tns Constant 7158
WSDLIntf.Wsdlns Constant 7159
WSDLIntf.xsdns Constant 7159

WSDLItems Namespace 7161

Classes 7163
TWSDLItems Class 7163
TWSDLItems Members 7164
Create Constructor 7169
TWSDLItems Methods 7170
TWSDLItems Properties 7197

Interfaces 7199
IHeaderInfo Interface 7199
IHeaderInfo Members 7199
IHeaderInfo Methods 7200
IHeaderInfo Properties 7203
IQualifiedName Interface 7204
IQualifiedName Members 7205
IQualifiedName Methods 7205
IQualifiedName Properties 7207
IWSDLItems Interface 7208
IWSDLItems Members 7208
IWSDLItems Methods 7208

Functions 7211
WSDLItems.HasDefinition Function 7211
WSDLItems.HasDefinition Function 7211
WSDLItems.NewQualifiedName Function 7211

Structs, Records, Enums 7213
WSDLItems.IterateImportOptions Enumeration 7213

Types 7214
WSDLItems.IHeaderInfoArray Type 7214

WSDLItems.IQualifiedNameArray Type 7214
WSDLItems.TWSDLIterateProc Type 7214

WSDLNode Namespace 7217

Classes 7218
EWSDLLoadException Class 7218
TWSDLClientNode Class 7218
TWSDLClientNode Members 7219
TWSDLClientNode Methods 7219
TWSDLClientNode Properties 7222
TWSDLView Class 7223
TWSDLView Members 7223
TWSDLView Fields 7224
TWSDLView Methods 7224
TWSDLView Properties 7225

Functions 7229
WSDLNode.ActivateWSDL Function 7229

WSDLPub Namespace 7231

Classes 7232
TWSDLHTMLPublish Class 7232
TWSDLHTMLPublish Members 7233
TWSDLHTMLPublish.Create Constructor 7233
TWSDLHTMLPublish.Destroy Destructor 7234
TWSDLHTMLPublish Methods 7234
TWSDLHTMLPublish Properties 7239
TWSDLHTMLPublish Events 7241
TWSDLPublish Class 7242
TWSDLPublish Members 7243
TWSDLPublish Methods 7243
TWSDLPublish Properties 7244
TWSDLPublish Events 7245

Interfaces 7247
IWSDLPublish Interface 7247
IWSDLPublish Members 7247
IWSDLPublish Methods 7247

Functions 7249

WSDLPub.WSDLPubFactory Function 7249

Structs, Records, Enums **7250**

WSDLPub.SOAPPublishOptions Enumeration 7250

Types **7251**

WSDLPub.TBeforePublishingWSDLEvent Type 7251

WSDLPub.TSOAPPublishOptions Type 7251

XMLBrokr Namespace 7253

Classes **7255**

TXMLBroker Class 7255

TXMLBroker Members 7256

TXMLBroker.Create Constructor 7257

TXMLBroker.Destroy Destructor 7258

TXMLBroker Methods 7258

TXMLBroker Properties 7273

TXMLBroker Events 7281

TXMLParams Class 7285

TXMLParams Members 7286

TXMLParams Methods 7286

Interfaces **7289**

INotifyConnectionChange Interface 7289

INotifyConnectionChange Members 7289

INotifyConnectionChange Methods 7289

Functions **7291**

XMLBrokr.FormatXML Function 7291

Structs, Records, Enums **7292**

XMLBrokr.TXMLOption Enumeration 7292

Types **7293**

XMLBrokr.TGetErrorResponseEvent Type 7293

XMLBrokr.TGetResponseEvent Type 7293

XMLBrokr.TRequestRecordsEvent Type 7294

XMLBrokr.TRequestUpdateEvent Type 7294

XMLBrokr.TXMLOptions Type 7294

Constants **7295**

XMLBrokr.sPostDelta Constant 7295

XMLBrokr.sProducer Constant 7295

XMLBrokr.sRedirect Constant 7295

XMLDoc Namespace 7297

Classes **7299**

TXMLDocument Class 7299

TXMLDocument Members 7300

Create Constructor 7303

TXMLDocument.Destroy Destructor 7304

TXMLDocument Methods 7304

TXMLDocument Properties 7332

TXMLDocument Events 7344

TXMLNode Class 7348

TXMLNode Members 7349

TXMLNode.Create Constructor 7352

TXMLNode.CreateHosted Constructor 7353

TXMLNode.Destroy Destructor 7353

TXMLNode Methods 7354

TXMLNode Properties 7389

TXMLNode Events 7395

TXMLNodeCollection Class 7396

TXMLNodeCollection Members 7397

TXMLNodeCollection Methods 7402

TXMLNodeCollection Properties 7409

TXMLNodeList Class 7411

TXMLNodeList Members 7412

TXMLNodeList.Create Constructor 7414

TXMLNodeList.Destroy Destructor 7415

TXMLNodeList Methods 7415

TXMLNodeList Properties 7425

TXMLNodeList Events 7428

Interfaces **7429**

IXMLDocumentAccess Interface 7429

IXMLDocumentAccess Members 7429

IXMLDocumentAccess Methods 7429

IXMLDocumentAccess Properties 7430

IXMLNodeAccess Interface 7431

IXMLNodeAccess Members 7431

IXMLNodeAccess Methods 7433

IXMLNodeAccess Properties 7438

Functions **7439**

XMLDoc.CheckEncoding Function 7439
XMLDoc.CreateDOMNode Function 7439
XMLDoc.DetectCharEncoding Function 7440
XMLDoc.FormatXMLData Function 7441
XMLDoc.LoadXMLData Function 7441
XMLDoc.LoadXMLDocument Function 7442
XMLDoc.NewXMLDocument Function 7442
XMLDoc.XMLDocError Function 7443
XMLDoc.XMLStringToUnicodeString Function 7443
XMLDoc.XMLStringToWideString Function 7443

Structs, Records, Enums **7445**

XMLDoc.TNodeChange Enumeration 7445
XMLDoc.TNodeClassInfo Record 7446
XMLDoc.TNodeListOperation Enumeration 7446
XMLDoc.TXMLDocumentSource Enumeration 7447
XMLDoc.TXMLPrologItem Enumeration 7448

Types **7449**

XMLDoc.TNodeChangeEvent Type 7449
XMLDoc.TNodeClassArray Type 7449
XMLDoc.TNodeListNotification Type 7450
XMLDoc.TXMLNodeArray Type 7451
XMLDoc.TXMLNodeClass Type 7451
XMLDoc.TXMLNodeCollectionClass Type 7452

Variables **7453**

XMLDoc.UseXSDBooleanStrings Variable 7453

xmldom Namespace 7455

Classes **7458**

DOMException Class 7458
DOMException Members 7458
DOMException Fields 7460
EDOMParseError Class 7460
EDOMParseError Members 7461
EDOMParseError.Create Constructor 7462
EDOMParseError Properties 7463

TDOMVendor Class 7466
TDOMVendor Members 7466
TDOMVendor Methods 7466
TDOMVendorList Class 7468
TDOMVendorList Members 7468
TDOMVendorList Methods 7469
TDOMVendorList Properties 7471

Interfaces 7473
IDOMAttr Interface 7473
IDOMAttr Members 7474
IDOMAttr Methods 7475
IDOMAttr Properties 7476
IDOMCDATASection Interface 7477
IDOMCharacterData Interface 7478
IDOMCharacterData Members 7478
IDOMCharacterData Methods 7479
IDOMCharacterData Properties 7481
IDOMComment Interface 7482
IDOMDocument Interface 7482
IDOMDocument Members 7482
IDOMDocument Methods 7484
IDOMDocument Properties 7488
IDOMDocumentFragment Interface 7489
IDOMDocumentType Interface 7490
IDOMDocumentType Members 7490
IDOMDocumentType Methods 7491
IDOMDocumentType Properties 7493
IDOMElement Interface 7494
IDOMElement Members 7495
IDOMElement Methods 7496
IDOMElement Properties 7500
IDOMEntity Interface 7501
IDOMEntity Members 7501
IDOMEntity Methods 7502
IDOMEntity Properties 7503
IDOMEntityReference Interface 7504
IDOMImplementation Interface 7504
IDOMImplementation Members 7504
IDOMImplementation Methods 7505

IDOMNamedNodeMap Interface 7506
IDOMNamedNodeMap Members 7506
IDOMNamedNodeMap Methods 7506
IDOMNamedNodeMap Properties 7508
IDOMNode Interface 7509
IDOMNode Members 7510
IDOMNode Methods 7511
IDOMNode Properties 7519
IDOMNodeEx Interface 7523
IDOMNodeEx Members 7523
IDOMNodeEx Methods 7524
IDOMNodeEx Properties 7526
IDOMNodeList Interface 7526
IDOMNodeList Members 7527
IDOMNodeList Methods 7527
IDOMNodeList Properties 7528
IDOMNodeSelect Interface 7528
IDOMNodeSelect Members 7529
IDOMNodeSelect Methods 7529
IDOMNotation Interface 7530
IDOMNotation Members 7530
IDOMNotation Methods 7531
IDOMNotation Properties 7532
IDOMParseError Interface 7532
IDOMParseError Members 7533
IDOMParseError Methods 7533
IDOMParseError Properties 7535
IDOMParseOptions Interface 7537
IDOMParseOptions Members 7537
IDOMParseOptions Methods 7537
IDOMParseOptions Properties 7539
IDOMPersist Interface 7541
IDOMPersist Members 7541
IDOMPersist Methods 7541
IDOMPersist Properties 7544
IDOMProcessingInstruction Interface 7545
IDOMProcessingInstruction Members 7545
IDOMProcessingInstruction Methods 7546
IDOMProcessingInstruction Properties 7547

IDOMText Interface 7548
IDOMText Members 7548
IDOMText Methods 7549
IDOMXMLProlog Interface 7550
IDOMXMLProlog Members 7550
IDOMXMLProlog Methods 7550
IDOMXMLProlog Properties 7552
IDOMXSLProcessor Interface 7553
IDOMXSLProcessor Members 7553
IDOMXSLProcessor Methods 7554
IDOMXSLProcessor Properties 7556

Functions **7557**
xmldom.DOMVendorNotSupported Function 7557
xmldom.ExtractLocalName Function 7557
xmldom.ExtractPrefix Function 7558
xmldom.GetDOM Function 7559
xmldom.GetDOMNodeEx Function 7559
xmldom.GetDOMVendor Function 7560
xmldom.IsPrefixed Function 7560
xmldom.MakeNodeName Function 7561
xmldom.NodeMatches Function 7561
xmldom.RegisterDOMVendor Function 7562
xmldom.SameNamespace Function 7562
xmldom.UnRegisterDOMVendor Function 7563

Types **7565**
xmldom.DOMNodeType Type 7565
xmldom.DOMString Type 7565
xmldom.DOMTimeStamp Type 7566
xmldom.TAsyncEventHandler Type 7566
xmldom.TDOMVendorArray Type 7567

Variables **7568**
xmldom.DefaultDOMVendor Variable 7568
xmldom.DOMVendors Variable 7568

Constants **7569**
xmldom.ATTRIBUTE_NODE Constant 7569
xmldom.CDATA_SECTION_NODE Constant 7570
xmldom.COMMENT_NODE Constant 7570

xmldom.DOCUMENT_FRAGMENT_NODE Constant 7570
xmldom.DOCUMENT_NODE Constant 7571
xmldom.DOCUMENT_TYPE_NODE Constant 7571
xmldom.DOMSTRING_SIZE_ERR Constant 7571
xmldom.DOMWrapperVersion Constant 7572
xmldom.ELEMENT_NODE Constant 7572
xmldom.ENTITY_NODE Constant 7572
xmldom.ENTITY_REFERENCE_NODE Constant 7573
xmldom.HIERARCHY_REQUEST_ERR Constant 7573
xmldom.INDEX_SIZE_ERR Constant 7573
xmldom.INUSE_ATTRIBUTE_ERR Constant 7574
xmldom.INVALID_ACCESS_ERR Constant 7574
xmldom.INVALID_CHARACTER_ERR Constant 7574
xmldom.INVALID_MODIFICATION_ERR Constant 7575
xmldom.INVALID_STATE_ERR Constant 7575
xmldom.NAMESPACE_ERR Constant 7575
xmldom.NO_DATA_ALLOWED_ERR Constant 7576
xmldom.NO_MODIFICATION_ALLOWED_ERR Constant 7576
xmldom.NOT_FOUND_ERR Constant 7576
xmldom.NOT_SUPPORTED_ERR Constant 7577
xmldom.NOTATION_NODE Constant 7577
xmldom.NSDelim Constant 7577
xmldom.PROCESSING_INSTRUCTION_NODE Constant 7578
xmldom.SEncoding Constant 7578
xmldom.SHttp Constant 7578
xmldom.SStandalone Constant 7579
xmldom.SVersion Constant 7579
xmldom.SXML Constant 7579
xmldom.SXMLNamespaceURI Constant 7580
xmldom.SXMLNS Constant 7580
xmldom.SXMLPrefixNamespaceURI Constant 7580
xmldom.SYNTAX_ERR Constant 7581
xmldom.TEXT_NODE Constant 7581
xmldom.WRONG_DOCUMENT_ERR Constant 7581

XMLIntf Namespace 7583

Classes **7585**

EXMLDocError Class 7585

Interfaces **7586**
IXMLDocument Interface 7586
IXMLDocument Members 7587
IXMLDocument Methods 7588
IXMLDocument Properties 7607
IXMLNode Interface 7615
IXMLNode Members 7616
IXMLNode Methods 7617
IXMLNode Properties 7633
IXMLNodeCollection Interface 7644
IXMLNodeCollection Members 7644
IXMLNodeCollection Methods 7646
IXMLNodeCollection Properties 7648
IXMLNodeList Interface 7649
IXMLNodeList Members 7650
IXMLNodeList Methods 7651
IXMLNodeList Properties 7660

Structs, Records, Enums **7662**
XMLIntf.TNodeType Enumeration 7662
XMLIntf.TParseOption Enumeration 7664
XMLIntf.TXMLDocOption Enumeration 7664
XMLIntf.TXMLEncodingType Enumeration 7665

Types **7667**
XMLIntf.TParseOptions Type 7667
XMLIntf.TXMLDocOptions Type 7667

Constants **7668**
XMLIntf.NodeTypeNames Constant 7668

Xmlxform Namespace 7669

Classes **7670**
TranslateException Class 7670
TXMLTransform Class 7671
TXMLTransform Members 7671
TXMLTransform Methods 7672
TXMLTransform Properties 7675
TXMLTransform Events 7681
TXMLTransformClient Class 7683

TXMLTransformClient Members 7684
TXMLTransformClient.Create Constructor 7685
TXMLTransformClient.Destroy Destructor 7685
TXMLTransformClient Methods 7685
TXMLTransformClient Properties 7688
TXMLTransformProvider Class 7691
TXMLTransformProvider Members 7693
TXMLTransformProvider.Create Constructor 7693
TXMLTransformProvider.Destroy Destructor 7694
TXMLTransformProvider Methods 7694
TXMLTransformProvider Properties 7695
TXMLTransformProvider Events 7698

Types **7705**
Xmlxform.TRowEvent Type 7705
Xmlxform.TTranslateEvent Type 7705

XPActnCtrls Namespace 7707

Classes **7708**
TXPStyleAddRemoveItem Class 7708
TXPStyleAddRemoveItem Members 7708
TXPStyleAddRemoveItem Methods 7709
TXPStyleButton Class 7711
TXPStyleButton Members 7711
TXPStyleButton Methods 7711
TXPStyleCustomizePopup Class 7713
TXPStyleCustomizePopup Members 7713
TXPStyleCustomizePopup.Create Constructor 7713
TXPStyleCustomizePopup Methods 7714
TXPStyleDropDownBtn Class 7714
TXPStyleDropDownBtn Members 7715
TXPStyleDropDownBtn Methods 7715
TXPStyleExpandBtn Class 7716
TXPStyleExpandBtn Members 7717
TXPStyleExpandBtn Methods 7717
TXPStyleMenuButton Class 7718
TXPStyleMenuButton Members 7718
TXPStyleMenuButton Methods 7718
TXPStyleMenuItem Class 7719

TXPStyleMenuItem Members 7719
TXPStyleMenuItem Methods 7720
TXPStylePopupMenu Class 7722
TXPStylePopupMenu Members 7723
TXPStylePopupMenu.Create Constructor 7723
TXPStylePopupMenu.Destroy Destructor 7723
TXPStylePopupMenu Methods 7724
TXPStyleToolScrollBtn Class 7726
TXPStyleToolScrollBtn Members 7726
TXPStyleToolScrollBtn Methods 7726

XPMan Namespace 7729

Classes **7730**
TXPManifest Class 7730

XPStyleActnCtrls Namespace 7731

Classes **7732**
TXPStyleActionBars Class 7732
TXPStyleActionBars Members 7732
TXPStyleActionBars Methods 7732

Variables **7735**
XPStyleActnCtrls.XPStyle Variable 7735

XSBuiltIns Namespace 7737

Classes **7739**
EXSDateTimeException Class 7739
EXSDecimalException Class 7740
EXSHexBinaryException Class 7740
TXMLData Class 7741
TXMLData Members 7741
TXMLData.Create Constructor 7742
TXMLData.Destroy Destructor 7742
TXMLData Methods 7742
TXMLData Properties 7745
TXSBoolean Class 7745
TXSBoolean Members 7745
TXSBoolean Methods 7746

TXSBoolean Properties 7747
TXSCustomDateTime Class 7747
TXSCustomDateTime Members 7748
TXSCustomDateTime.Create Constructor 7749
TXSCustomDateTime.Destroy Destructor 7749
TXSCustomDateTime Fields 7750
TXSCustomDateTime Methods 7750
TXSCustomDateTime Properties 7759
TXSDate Class 7766
TXSDate Members 7766
TXSDate.Create Constructor 7767
TXSDate Methods 7767
TXSDate Properties 7770
TXSDateTime Class 7772
TXSDateTime Members 7773
TXSDateTime Methods 7774
TXSDateTime Properties 7776
TXSDecimal Class 7776
TXSDecimal Members 7777
TXSDecimal Methods 7777
TXSDecimal Properties 7778
TXSDuration Class 7780
TXSDuration Members 7780
TXSDuration.Create Constructor 7782
TXSDuration Methods 7782
TXSDuration Properties 7783
TXSHexBinary Class 7784
TXSHexBinary Members 7785
TXSHexBinary Methods 7785
TXSHexBinary Properties 7786
TXSInteger Class 7787
TXSInteger Members 7787
TXSInteger Methods 7787
TXSInteger Properties 7788
TXSLong Class 7789
TXSLong Members 7789
TXSLong Methods 7789
TXSLong Properties 7790
TXSString Class 7791

TXSString Members 7791
TXSString Methods 7791
TXSTime Class 7792
TXSTime Members 7793
TXSTime Methods 7794
TXSTime Properties 7800

Functions 7805
XSBuiltIns.DateTimeToXMLTime Function 7805
XSBuiltIns.DateTimeToXSDateTime Function 7805
XSBuiltIns.GetDataFromFile Function 7806
XSBuiltIns.InitXSTypes Function 7806
XSBuiltIns.SoapFloatToStr Function 7807
XSBuiltIns.SoapStrToFloat Function 7807
XSBuiltIns.XMLTimeToDateTime Function 7807

Types 7809
XSBuiltIns.TXSTimeInstant Type 7809

Constants 7810
XSBuiltIns.IncrementAmount Constant 7810
XSBuiltIns.MaxDay Constant 7811
XSBuiltIns.MaxMonth Constant 7811
XSBuiltIns.MinDay Constant 7811
XSBuiltIns.MinMonth Constant 7812
XSBuiltIns.SDecimal Constant 7812
XSBuiltIns.SHexMarker Constant 7812
XSBuiltIns.SLocalTimeMarker Constant 7813
XSBuiltIns.SNAN Constant 7813
XSBuiltIns.SNegative Constant 7813
XSBuiltIns.SoapDecimalSeparator Constant 7814
XSBuiltIns.SoapTimePrefix Constant 7814
XSBuiltIns.SPlus Constant 7814
XSBuiltIns.SSciNotationMarker Constant 7815
XSBuiltIns.XMLDateSeparator Constant 7815
XSBuiltIns.XMLDayPos Constant 7815
XSBuiltIns.XMLDefaultYearDigits Constant 7816
XSBuiltIns.XMLDurationDay Constant 7816
XSBuiltIns.XMLDurationHour Constant 7816
XSBuiltIns.XMLDurationMinute Constant 7817
XSBuiltIns.XMLDurationMonth Constant 7817

XSBuiltIns.XMLDurationSecond Constant 7817
XSBuiltIns.XMLDurationStart Constant 7818
XSBuiltIns.XMLDurationYear Constant 7818
XSBuiltIns.XMLHourOffsetMinusMarker Constant 7818
XSBuiltIns.XMLHourOffsetPlusMarker Constant 7819
XSBuiltIns.XMLMilSecPos Constant 7819
XSBuiltIns.XMLMonthPos Constant 7819
XSBuiltIns.XMLTimeSeparator Constant 7820
XSBuiltIns.XMLYearPos Constant 7820

XSLProd Namespace 7821

Classes **7822**
EXSLPageProducerError Class 7822
TCustomXSLPageProducer Class 7822
TCustomXSLPageProducer Members 7823
TCustomXSLPageProducer Methods 7828
TCustomXSLPageProducer Properties 7835
TXSLPageProducer Class 7837
TXSLPageProducer Members 7838
TXSLPageProducer Properties 7843

ZLib Namespace 7845

Classes **7848**
EZCompressionError Class 7848
EZDecompressionError Class 7848
EZLibError Class 7849
TCustomZStream Class 7849
TCustomZStream Members 7849
TCustomZStream.Create Constructor 7850
TCustomZStream Methods 7850
TCustomZStream Events 7850
TZCompressionStream Class 7851
TZCompressionStream Members 7851
Create Constructor 7852
TZCompressionStream.Destroy Destructor 7852
TZCompressionStream Methods 7853
TZCompressionStream Properties 7854
TZCompressionStream Events 7855

TZDecompressionStream Class 7855
TZDecompressionStream Members 7855
TZDecompressionStream.Create Constructor 7856
TZDecompressionStream.Destroy Destructor 7856
TZDecompressionStream Methods 7857
TZDecompressionStream Events 7858

Functions **7859**
ZLib.adler32 Function 7859
ZLib.deflate Function 7860
ZLib.deflateEnd Function 7860
ZLib.deflateInit_ Function 7860
ZLib.DeflateInit2_ Function 7861
ZLib.inflate Function 7861
ZLib.inflateEnd Function 7861
ZLib.inflateInit_ Function 7862
ZLib.inflateInit2_ Function 7862
ZLib.inflateReset Function 7862
ZLib.MoveI32 Function 7863
ZLib.ZCompress Function 7863
ZLib.ZCompress Function 7863
ZLib.ZCompressStr Function 7864
ZLib.ZCompressStr Function 7864
ZLib.ZCompressStream Function 7864
ZLib.ZCompressStream Function 7865
ZLib.ZDecompress Function 7865
ZLib.ZDecompressStr Function 7865
ZLib.ZDecompressStream Function 7866
ZLib.zlibAllocMem Function 7866
ZLib.zlibFreeMem Function 7866
ZLib.ZSendToBrowser Function 7867

Structs, Records, Enums **7868**
ZLib.TCompressionLevel Enumeration 7868
ZLib.TZCompressionLevel Enumeration 7869
ZLib.TZStreamRec Record 7869

Types **7871**
ZLib.TCompressionStream Type 7871
ZLib.TCustomZLibStream Type 7872
ZLib.TDecompressionStream Type 7872

ZLib.TZAlloc Type 7872
ZLib.TZFree Type 7873

Constants 7874

ZLib._z_errmsg Constant 7874
ZLib.SZInvalid Constant 7875
ZLib.Z_ASCII Constant 7875
ZLib.Z_BEST_COMPRESSION Constant 7875
ZLib.Z_BEST_SPEED Constant 7876
ZLib.Z_BINARY Constant 7876
ZLib.Z_BUF_ERROR Constant 7876
ZLib.Z_DATA_ERROR Constant 7877
ZLib.Z_DEFAULT_COMPRESSION Constant 7877
ZLib.Z_DEFAULT_STRATEGY Constant 7877
ZLib.Z_DEFLATED Constant 7878
ZLib.Z_ERRNO Constant 7878
ZLib.Z_FILTERED Constant 7878
ZLib.Z_FINISH Constant 7879
ZLib.Z_FULL_FLUSH Constant 7879
ZLib.Z_HUFFMAN_ONLY Constant 7879
ZLib.Z_MEM_ERROR Constant 7880
ZLib.Z_NEED_DICT Constant 7880
ZLib.Z_NO_COMPRESSION Constant 7880
ZLib.Z_NO_FLUSH Constant 7881
ZLib.Z_OK Constant 7881
ZLib.Z_PARTIAL_FLUSH Constant 7881
ZLib.Z_STREAM_END Constant 7882
ZLib.Z_STREAM_ERROR Constant 7882
ZLib.Z_SYNC_FLUSH Constant 7882
ZLib.Z_UNKNOWN Constant 7883
ZLib.Z_VERSION_ERROR Constant 7883
ZLib.ZLevels Constant 7883
ZLib.ZLIB_VERSION Constant 7884

ZLibConst Namespace 7885

Constants 7886

ZLibConst.sError Constant 7886
ZLibConst.sInvalidStreamOp Constant 7886
ZLibConst.sTargetBufferTooSmall Constant 7886

Index

a

1 ScktComp Namespace

This is namespace ScktComp.

Topics

Name	Description
Classes (see page 3)	The following table lists classes in this documentation.
Functions (see page 133)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 134)	The following table lists structs, records, enums in this documentation.
Types (see page 139)	The following table lists types in this documentation.
Constants (see page 143)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 3)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 4)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 14)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 25)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 29)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 46)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 54)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 80)	This is class ScktComp.TServerAcceptThread.
	TServerClientThread (see page 85)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 93)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 97)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 109)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 127)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Constants

Name	Description
CM_DEFERFREE (see page 143)	This is constant ScktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 143)	This is constant ScktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 143)	This is constant ScktComp.CM_SOCKETMESSAGE.

Functions

	Name	Description
	SetErrorProc (see page 133)	Replaces the exception handler for error messages that are received from a Windows socket connection.

Structs, Records, Enums

	Name	Description
	TAsyncStyle (see page 134)	This is record ScktComp.TAsyncStyle.
	TClientType (see page 135)	This is record ScktComp.TClientType.
	TCMLookupComplete (see page 135)	This is record ScktComp.TCMLookupComplete.
	TCMSocketMessage (see page 136)	This is record ScktComp.TCMSocketMessage.
	TErrorEvent (see page 136)	This is record ScktComp.TErrorEvent.
	TLookupState (see page 137)	This is record ScktComp.TLookupState.
	TServerType (see page 137)	This is record ScktComp.TServerType.
	TSocketEvent (see page 137)	TSocketEvent identifies the operation that a socket is performing.

Types

Name	Description
TAsyncStyles (see page 139)	This is type ScktComp.TAsyncStyles.
TGetSocketEvent (see page 139)	This is type ScktComp.TGetSocketEvent.
TGetThreadEvent (see page 140)	This is type ScktComp.TGetThreadEvent.
TSocketErrorEvent (see page 140)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 141)	This is type ScktComp.TSocketErrorProc.
TSocketEventEvent (see page 141)	This is type ScktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 141)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 142)	This is type ScktComp.TThreadNotifyEvent.

1.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 3)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 4)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 14)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 25)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 29)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 46)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 54)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 80)	This is class ScktComp.TServerAcceptThread.
	TServerClientThread (see page 85)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 93)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 97)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 109)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 127)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

1.1.1 ESocketError Class

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

Class Hierarchy

Pascal

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

C++

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

File

ScktComp

Description

ESocketError is raised if

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

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

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

A socket cannot be properly closed.

There are two ESocketError classes, one in the ScktComp (see page 1) unit and one in the Sockets (see page 1299) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

See Also

TCustomWinSocket (see page 54)

1.1.2 TAbstractSocket Class

TAbstractSocket is the abstract base class for all socket components.

Class Hierarchy

Pascal

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

C++

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

File

ScktComp

Description

TAbstractSocket introduces properties and methods to enable an application to work with sockets. A socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Sockets (see page 1299) provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 3165) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 1299) allow an application to form connections to other machines without being concerned with the details of the actual networking software.

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

Do not create instances of TAbstractSocket. Use a descendant of TAbstractSocket to add the ability to communicate with other machines to an application. To create a socket that initiates the connection with another machine, use TClientSocket (see page 14). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 97).

See Also

TClientSocket (see page 14)

TCustomWinSocket (see page 54)

TServerSocket (see page 97)

TCustomSocket (see page 46)

1.1.2.1 TAbstractSocket Members

The following tables list the members exposed by TAbstractSocket.

Protected Methods

	Name	Description
	DoActivate (see page 6)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 6)	Provides the prototype of a method that responds to error conditions.
	Event (see page 7)	Provides the prototype of a method that responds when specific socket events occur.
	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetService (see page 11)	Sets the Service (see page 13) property.

Protected Properties

	Name	Description
	Active (see page 11)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 12)	Specifies the IP address of the server system.
	Host (see page 12)	Specifies an alias for the IP address of the server system.
	Port (see page 13)	Specifies the ID number that identifies the server socket connection.
	Service (see page 13)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
	Open (see page 9)	Opens the socket connection.

1.1.2.2 TAbstractSocket Methods

The methods of the TAbstractSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 6)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 6)	Provides the prototype of a method that responds to error conditions.
	Event (see page 7)	Provides the prototype of a method that responds when specific socket events occur.
	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetService (see page 11)	Sets the Service (see page 13) property.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
	Open (see page 9)	Opens the socket connection.

1.1.2.2.1 TAbstractSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 11) property to false.

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

See Also

Active (see page 11)

1.1.2.2.2 TAbstractSocket.DoActivate Method

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

Pascal

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

C++

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

Description

Descendants of TAbstractSocket (see page 4) override DoActivate to provide a method that initiates or terminates the socket connection. The steps that need to be taken to open or close a connection depend on whether the socket is a client socket or a server socket.

See Also

Active (see page 11)

Open (see page 9)

Close (see page 6)

1.1.2.2.3 TAbstractSocket.Error Method

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

Pascal

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

C++

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

Description

Descendants of TAbstractSocket (see page 4) must override the abstract or, in C++ terminology, pure virtual Error method to provide a response when socket errors occur. Applications cannot call Error. The Windows socket object that encounters the error condition calls Error automatically.

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

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

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

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

See Also

OnErrorEvent (see page 79)

InitSocket (see page 8)

1.1.2.2.4 TAbstractSocket.Event Method

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

Pascal

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

C++

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

Description

Descendants of TAbstractSocket (see page 4) must override the abstract or, in C++ terminology, pure virtual Event method to provide a response to calls that arise during the normal functioning of the socket. Applications cannot call Event. The Windows socket object calls Event at appropriate times.

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

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

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

See Also

OnSocketEvent (see page 79)

InitSocket (see page 8)

1.1.2.2.5 TAbstractSocket.InitSocket Method

Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.

Pascal

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

C++

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

Description

Applications cannot call this protected method. InitSocket is called automatically by the socket's constructor, immediately after the Windows socket object is created. It allows the socket component to respond to events on the Windows socket object by assigning an OnErrorEvent handler that calls Error (see page 6) and an OnSocketEvent handler that calls Event (see page 7).

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

See Also

Error (see page 6)

TAbstractSocket (see page 4)

OnErrorEvent (see page 79)

OnSocketEvent (see page 79)

1.1.2.2.6 TAbstractSocket.Loaded Method

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

page 11) property.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Applications cannot call this protected method. Loaded is called automatically after the socket component has been loaded from a stream. Loaded overrides the default method to implement the value of the Active (see page 11) property that was saved with the socket component.

See Also

Active (see page 11)

1.1.2.2.7 TAbstractSocket.Open Method

Opens the socket connection.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 11) property to true.

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

See Also

Active (see page 11)

1.1.2.2.8 TAbstractSocket.SetActive Method

Sets the Active (see page 11) property.

Pascal

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

C++

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

Description

SetActive is the protected implementation of the Active (see page 11) property. SetActive calls the DoActivate (see page 6) method to open or close the socket connection.

See Also

Active (see page 11)

DoActivate (see page 6)

1.1.2.2.9 TAbstractSocket.SetAddress Method

Sets the Address (see page 12) property.

Pascal

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

C++

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

Description

SetAddress is the protected implementation of the Address (see page 12) property. SetAddress raises an ESocketError (see page 3) exception if an attempt is made to set the Address (see page 12) property when the socket connection is open.

See Also

Active (see page 11)

Address (see page 12)

ESocketError

1.1.2.2.10 TAbstractSocket.SetHost Method

Sets the Host (see page 12) property.

Pascal

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

C++

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

Description

SetHost is the protected implementation of the Host (see page 12) property. SetHost raises an ESocketError (see page 3) exception if an attempt is made to set the Host (see page 12) property when the socket connection is open.

See Also

Active (see page 11)

ESocketError

Host (see page 12)

1.1.2.2.11 TAbstractSocket.SetPort Method

Sets the Port (see page 13) property.

Pascal

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

C++

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

Description

SetPort is the protected implementation of the Port (see page 13) property. SetPort raises an ESocketError (see page 3) exception if an attempt is made to set the Port (see page 13) property when the socket connection is open.

See Also

Active (see page 11)

ESocketError

Port (see page 13)

1.1.2.2.12 TAbstractSocket.SetService Method

Sets the Service (see page 13) property.

Pascal

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

C++

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

Description

SetService is the protected implementation of the Service (see page 13) property. SetService raises an ESocketError (see page 3) exception if an attempt is made to set the Service (see page 13) property when the socket connection is open.

See Also

Active (see page 11)

ESocketError

Service (see page 13)

1.1.2.3 TAbstractSocket Properties

The properties of the TAbstractSocket class are listed here.

Protected Properties

	Name	Description
	Active (see page 11)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 12)	Specifies the IP address of the server system.
	Host (see page 12)	Specifies an alias for the IP address of the server system.
	Port (see page 13)	Specifies the ID number that identifies the server socket connection.
	Service (see page 13)	Specifies the name of the service for which the socket connection is used.

1.1.2.3.1 TAbstractSocket.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

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

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open (see page 9) or Close (see page 6) method to open or close the connection.

See Also

Close (see page 6)

Open (see page 9)

SetActive (see page 9)

1.1.2.3.2 TAbstractSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

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

```
123.197.1.2
```

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

If the socket specifies the Host (see page 12) property, the address for the connection is taken from the IP address associated with Host (see page 12), rather than the value of Address. Specifying the server system by giving the IP address is faster, because the socket doesn't need to search for the IP address associated with the host name before it can locate the server system.

A single system may support more than one IP address.

Note: Trying to change Address when the connection is open will raise an ESocketError (see page 3) exception.

See Also

ESocketError

Host (see page 12)

SetAddress (see page 10)

1.1.2.3.3 TAbstractSocket.Host Property

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

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

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

`http://www.wSite.Com`

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

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

If Host is set, it takes precedence over the Address (see page 12) property when looking up the address of the server.

Note: Trying to change Host when the connection is open will raise an ESocketError (see page 3) exception.

See Also

Address (see page 12)

ESocketError

SetHost (see page 10)

1.1.2.3.4 TAbstractSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host (see page 12) or Address (see page 12) property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

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

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

Note: Trying to change Port when the connection is open will raise an ESocketError (see page 3) exception.

See Also

Address (see page 12)

ESocketError

Host (see page 12)

Service (see page 13)

SetPort (see page 10)

1.1.2.3.5 TAbstractSocket.Service Property

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

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

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

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port (see page 13) to use for the socket connection. For server sockets, using Service rather than Port (see page 13) ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError (see page 3) exception.

See Also

ESocketError

Port (see page 13)

SetService (see page 11)

LookupService (see page 65)

1.1.3 TClientSocket Class

TClientSocket manages socket connections for a TCP/IP client.

Class Hierarchy

Pascal

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

C++

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

File

SocktComp

Description

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

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

See Also

TClientWinSocket (see page 25)

TServerSocket (see page 97)

1.1.3.1 TClientSocket Members

The following tables list the members exposed by TClientSocket.

Protected Events

	Name	Description
	OnAccept (see page 49)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnListen (see page 51)	Occurs just before a server socket is opened for listening.

Protected Methods

	Name	Description
V	DoActivate (see page 17)	Provides the prototype of a method to open or close the socket connection.
A	Error (see page 6)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 48)	Generates an OnError (see page 51) event.
A	Event (see page 7)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 48)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetClientType (see page 17)	This is GetClientType, a member of class TClientSocket.
	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
V	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetClientType (see page 17)	This is SetClientType, a member of class TClientSocket.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetService (see page 11)	Sets the Service (see page 13) property.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
V	Create (see page 16)	Creates an instance of TClientSocket (see page 14).
V	Destroy (see page 16)	Destroys an instance of TClientSocket (see page 14).
	Open (see page 9)	Opens the socket connection.

Public Properties

	Name	Description
R	Socket (see page 21)	Describes the client endpoint of the Windows client socket connection.

Published Events

	Name	Description
	OnConnect (see page 22)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 22)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 23)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 23)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 24)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 24)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 25)	Occurs when a client socket should write information to the socket connection.

Published Properties

	Name	Description
	Active (see page 18)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18)	Specifies the IP address of the server system.
	ClientType (see page 19)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 20)	Specifies an alias for the IP address of the server system.
	Port (see page 20)	Specifies the ID number that identifies the server socket connection.
	Service (see page 21)	Specifies the name of the service for which the socket connection is used.

1.1.3.2 TClientSocket.Create Constructor

Creates an instance of TClientSocket (see page 14).

Pascal

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

C++

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

Description

Call Create to create and initialize a client socket component at runtime. TClientSocket (see page 14) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TClientWinSocket (see page 25) object that describes the client endpoint of the Windows socket connection.

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

See Also

TClientWinSocket (see page 25)

Create

1.1.3.3 TClientSocket.Destroy Destructor

Destroys an instance of TClientSocket (see page 14).

Pascal

```
destructor Destroy; override;
```

C++

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

Description

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

Destroy frees the TClientWinSocket (see page 25) object that describes the client endpoint of the Windows socket connection.

See Also

System::TObject::Free (see page 3382)

TClientWinSocket (see page 25)

1.1.3.4 TClientSocket Methods

The methods of the TClientSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 17)	Provides the prototype of a method to open or close the socket connection.
	GetClientType (see page 17)	This is GetClientType, a member of class TClientSocket.
	SetClientType (see page 17)	This is SetClientType, a member of class TClientSocket.

1.1.3.4.1 TClientSocket.DoActivate Method

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

Pascal

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

C++

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

Description

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

See Also

Active (see page 11)

Open (see page 9)

Close (see page 6)

1.1.3.4.2 TClientSocket.GetClientType Method

Pascal

```
function GetClientType: TClientType;
```

C++

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

Description

This is GetClientType, a member of class TClientSocket.

1.1.3.4.3 TClientSocket.SetClientType Method

Pascal

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

C++

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

Description

This is SetClientType, a member of class TClientSocket.

1.1.3.5 TClientSocket Properties

The properties of the TClientSocket class are listed here.

Public Properties

	Name	Description
R	Socket (see page 21)	Describes the client endpoint of the Windows client socket connection.

Published Properties

	Name	Description
	Active (see page 18)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18)	Specifies the IP address of the server system.
	ClientType (see page 19)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 20)	Specifies an alias for the IP address of the server system.
	Port (see page 20)	Specifies the ID number that identifies the server socket connection.
	Service (see page 21)	Specifies the name of the service for which the socket connection is used.

1.1.3.5.1 TClientSocket.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

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

See Also

Close (see page 6)

Open (see page 9)

SetActive (see page 9)

1.1.3.5.2 TClientSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

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

`123.197.1.2`

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

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

A single system may support more than one IP address.

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

See Also

ESocketError

Host (see page 12)

SetAddress (see page 10)

1.1.3.5.3 TClientSocket.ClientType Property

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

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Set ClientType to ctNonBlocking to enable the client socket to respond to asynchronous reading and writing events. When ClientType is ctNonBlocking, execution is not blocked by reading and writing over the socket connection. OnRead (see page 24) or OnWrite (see page 25) events occur when the socket needs to read or write over the connection.

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

When ClientType is ctBlocking, use a TWinSocketStream (see page 127) object for reading and writing. TWinSocketStream (see page 127) prevents the application from hanging indefinitely if a problem occurs while reading or writing. It also can wait for the socket connection to indicate its readiness for reading.

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

See Also

OnRead (see page 53)

OnWrite (see page 53)

TThread

TWinSocketStream (see page 127)

1.1.3.5.4 TClientSocket.Host Property

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

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

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

`http://www.wSite.Com`

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

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

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

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

See Also

Address (see page 12)

ESocketError

SetHost (see page 10)

1.1.3.5.5 TClientSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

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

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

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

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

See Also

Address (see page 12)

ESocketError

Host (see page 12)

Service (see page 13)

SetPort (see page 10)

1.1.3.5.6 TClientSocket.Service Property

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

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

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

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

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

See Also

ESocketError

Port (see page 13)

SetService (see page 11)

LookupService (see page 65)

1.1.3.5.7 TClientSocket.Socket Property

Describes the client endpoint of the Windows client socket connection.

Pascal

```
property Socket: TClientWinSocket;
```

C++

```
__property TClientWinSocket Socket;
```

Description

Use Socket to

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

Read (see page 3727) or write information through the socket connection.

Determine what server notifications the client will respond to.

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

1.1.3.6 TClientSocket Events

The events of the TClientSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 22)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 22)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 23)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 23)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 24)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 24)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 25)	Occurs when a client socket should write information to the socket connection.

1.1.3.6.1 TClientSocket.OnConnect Event

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

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

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

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

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

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

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

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

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 49)

OnConnecting (see page 50)

OnLookup (see page 52)

1.1.3.6.2 TClientSocket.OnConnecting Event

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

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

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

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

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

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

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

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

5An OnConnect event occurs after the connection is established.

See Also

OnConnect (see page 50)

OnLookup (see page 52)

1.1.3.6.3 TClientSocket.OnDisconnect Event

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

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

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

See Also

Active (see page 11)

Close (see page 6)

OnConnect (see page 50)

1.1.3.6.4 TClientSocket.OnError Event

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

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

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

See Also

ESocketError

1.1.3.6.5 TClientSocket.OnLookup Event

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

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

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

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

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

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

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

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

5An OnConnect event occurs after the connection is established.

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

See Also

Address (see page 12)

Host (see page 12)

OnConnect (see page 50)

OnConnecting (see page 50)

Open (see page 9)

Port (see page 13)

Service (see page 13)

1.1.3.6.6 TClientSocket.OnRead Event

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

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

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

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

See Also

receivebuf (see page 66)

ReceiveText (see page 67)

TWinSocketStream (see page 127)

OnDisconnect (see page 51)

1.1.3.6.7 TClientSocket.OnWrite Event

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

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

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

See Also

SendBuf (see page 68)

sendstream (see page 68)

SendStreamThenDrop (see page 69)

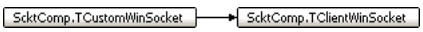
SendText (see page 69)

TWinSocketStream (see page 127)

1.1.4 TClientWinSocket Class

TClientWinSocket describes the client endpoint of a Windows socket connection.

Class Hierarchy



Pascal

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

C++

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

File

ScktComp

Description

Client socket components use TClientWinSocket to manage the Windows socket API calls for a TCP/IP client application. TClientWinSocket differs from its immediate ancestor TCustomWinSocket (see page 54) only in that it introduces a ClientType (see page 28) property to distinguish blocking from non-blocking client sockets.

See Also

TClientSocket (see page 14)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.4.1 TClientWinSocket Members

The following tables list the members exposed by TClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 58)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 60)	Opens a listening connection.
	DoOpen (see page 61)	Opens a connection to a remote socket.
	Error (see page 61)	Generates an OnErrorEvent (see page 79) event.
	Event (see page 62)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 62)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 70)	Sets the AsyncStyles property.
	SetClientType (see page 28)	This is SetClientType, a member of class TClientWinSocket.

Public Events

	Name	Description
	OnErrorEvent (see page 79)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 79)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
	Accept (see page 58)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 59)	Shuts down the socket connection if it is open.
	Connect (see page 28)	Responds to connection notifications.
	Create (see page 56)	Creates an instance of TCustomWinSocket (see page 54) from a Windows socket handle.
	DefaultHandler (see page 60)	Processes message records.
	Destroy (see page 56)	Destroys an instance of TCustomWinSocket (see page 54).
	Disconnect (see page 60)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 63)	Opens a listening connection for a server socket.
	Lock (see page 64)	Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.
	LookupName (see page 64)	Generates the portion of a Windows Internet socket address structure that describes the host.

	LookupService (see page 65)	Returns the port number associated with a specific service.
	Open (see page 65)	Opens a connection to a remote socket.
V	Read (see page 66)	Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 66)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 67)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 67)	Reads a string from the socket connection.
	SendBuf (see page 68)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 68)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 69)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 69)	Writes the string S to the socket connection.
	Unlock (see page 70)	Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.
V	Write (see page 70)	Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 72)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 72)	Determines which asynchronous events the socket can receive.
	ClientType (see page 28)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.
R	Connected (see page 73)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 73)	Points to application-specific data associated with the socket.
R	Handle (see page 73)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 74)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 74)	Specifies the name of the local system.
R	LocalPort (see page 75)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 75)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 76)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 76)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 77)	Specifies the name of the remote system.
R	RemotePort (see page 77)	Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.
R	SocketHandle (see page 78)	Specifies the Windows handle for the socket.

1.1.4.2 TClientWinSocket Methods

The methods of the TClientWinSocket class are listed here.

Protected Methods

	Name	Description
	SetClientType (see page 28)	This is SetClientType, a member of class TClientWinSocket.

Public Methods

	Name	Description
V	Connect (see page 28)	Responds to connection notifications.

1.1.4.2.1 TClientWinSocket.Connect Method

Responds to connection notifications.

Pascal

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

C++

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

Description

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

Connect generates an OnSocketEvent (see page 79) event with the SocketEvent parameter set to seConnect. Socket components assign an OnSocketEvent (see page 79) event handler that responds to this event by generating an OnConnect event on the socket component.

Note: Connect is only called if the ASyncStyles (see page 72) property includes asConnect.

See Also

ASyncStyles (see page 72)

OnSocketEvent (see page 79)

OnConnect (see page 50)

1.1.4.2.2 TClientWinSocket.SetClientType Method

Pascal

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

C++

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

Description

This is SetClientType, a member of class TClientWinSocket.

1.1.4.3 TClientWinSocket Properties

The properties of the TClientWinSocket class are listed here.

Public Properties

	Name	Description
	ClientType (see page 28)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

1.1.4.3.1 TClientWinSocket.ClientType Property

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

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Client socket components set ClientType to reflect their own ClientType property. When ClientType is ctNonBlocking, the client Windows socket responds to asynchronous reading and writing events, and execution is not blocked by reading and writing over the connection. OnSocketEvent (see page 79) events occur when the socket needs to read or write over the connection.

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

When ClientType is ctBlocking, use a TWinSocketStream (see page 127) object for reading and writing. TWinSocketStream (see page 127) times out so that the application does not hang when a problem occurs while reading or writing. TWinSocketStream (see page 127) also provides a method to wait, before reading, until the socket connection is ready to send information.

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

See Also

OnSocketEvent (see page 79)

TThread

TWinSocketStream (see page 127)

1.1.5 TCustomServerSocket Class

TCustomServerSocket is the base class for all server socket components.

Class Hierarchy

Pascal

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

C++

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

File

ScktComp

Description

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

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

See Also

TServerWinSocket (see page 109)

TServerSocket (see page 97)

1.1.5.1 TCustomServerSocket Members

The following tables list the members exposed by TCustomServerSocket.

Protected Events

	Name	Description
	OnAccept (see page 49)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 41)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 42)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 42)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 43)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 43)	Occurs when the server socket should write information to a client socket.
	OnConnect (see page 50)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 50)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 51)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 51)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 44)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 44)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 51)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 52)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 53)	Occurs when a client socket should read information from the socket connection.
	OnThreadEnd (see page 45)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 45)	Occurs when the execution thread for a connection to a client socket starts up.
	OnWrite (see page 53)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FServerSocket (see page 32)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
V	DoActivate (see page 32)	Opens or Closes the socket connection.
A	Error (see page 6)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 48)	Generates an OnError (see page 51) event.
A	Event (see page 7)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 48)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetGetSocketEvent (see page 33)	Returns the OnGetSocket (see page 44) event handler.
	GetGetThreadEvent (see page 33)	Returns the OnGetThread (see page 44) event handler.
	GetOnClientError (see page 34)	Returns the OnClientError (see page 42) event handler.
	GetOnClientEvent (see page 34)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 34)	Returns the OnThreadEnd (see page 45) event handler.
	GetOnThreadStart (see page 35)	Returns the OnThreadStart (see page 45) event handler.
	GetServerType (see page 35)	Returns the ServerType (see page 39) property.
	GetThreadCacheSize (see page 35)	Returns the value of the ThreadCacheSize (see page 40) property.

	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
V	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetGetSocketEvent (see page 36)	Sets the OnGetSocket (see page 44) event handler.
	SetGetThreadEvent (see page 36)	Sets the OnGetThread (see page 44) event handler.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetOnClientError (see page 37)	Sets the OnClientError (see page 42) event handler.
	SetOnClientEvent (see page 37)	Sets the client event handlers of the associated TServerWinSocket (see page 109) object.
	SetOnThreadEnd (see page 37)	Sets the OnThreadEnd (see page 45) event handler.
	SetOnThreadStart (see page 38)	Sets the OnThreadStart (see page 45) event handler.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetServerType (see page 38)	Sets the ServerType (see page 39) property.
	SetService (see page 11)	Sets the Service (see page 13) property.
	SetThreadCacheSize (see page 39)	Sets the ThreadCacheSize (see page 40) property.

Protected Properties

	Name	Description
	Active (see page 11)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 12)	Specifies the IP address of the server system.
	Host (see page 12)	Specifies an alias for the IP address of the server system.
	Port (see page 13)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 39)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 13)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 40)	Specifies the maximum number of threads that can be reused for new client connections.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
V	Destroy (see page 31)	Destroys an instance of TCustomServerSocket (see page 29).
	Open (see page 9)	Opens the socket connection.

1.1.5.2 TCustomServerSocket.Destroy Destructor

Destroys an instance of TCustomServerSocket (see page 29).

Pascal

```
destructor Destroy; override;
```

C++

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

Description

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

Destroy frees the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

System::TObject::Free (see page 3382)

TServerWinSocket (see page 109)

1.1.5.3 TCustomServerSocket Fields

The fields of the TCustomServerSocket class are listed here.

Protected Fields

	Name	Description
	FServerSocket (see page 32)	This is FServerSocket, a member of class TCustomServerSocket.

1.1.5.3.1 TCustomServerSocket.FServerSocket Field

Pascal

```
FServerSocket: TServerWinSocket;
```

C++

```
TServerWinSocket FServerSocket;
```

Description

This is FServerSocket, a member of class TCustomServerSocket.

1.1.5.4 TCustomServerSocket Methods

The methods of the TCustomServerSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 32)	Opens or Closes the socket connection.
	GetGetSocketEvent (see page 33)	Returns the OnGetSocket (see page 44) event handler.
	GetGetThreadEvent (see page 33)	Returns the OnGetThread (see page 44) event handler.
	GetOnClientError (see page 34)	Returns the OnClientError (see page 42) event handler.
	GetOnClientEvent (see page 34)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 34)	Returns the OnThreadEnd (see page 45) event handler.
	GetOnThreadStart (see page 35)	Returns the OnThreadStart (see page 45) event handler.
	GetServerType (see page 35)	Returns the ServerType (see page 39) property.
	GetThreadCacheSize (see page 35)	Returns the value of the ThreadCacheSize (see page 40) property.
	SetGetSocketEvent (see page 36)	Sets the OnGetSocket (see page 44) event handler.
	SetGetThreadEvent (see page 36)	Sets the OnGetThread (see page 44) event handler.
	SetOnClientError (see page 37)	Sets the OnClientError (see page 42) event handler.
	SetOnClientEvent (see page 37)	Sets the client event handlers of the associated TServerWinSocket (see page 109) object.
	SetOnThreadEnd (see page 37)	Sets the OnThreadEnd (see page 45) event handler.
	SetOnThreadStart (see page 38)	Sets the OnThreadStart (see page 45) event handler.
	SetServerType (see page 38)	Sets the ServerType (see page 39) property.
	SetThreadCacheSize (see page 39)	Sets the ThreadCacheSize (see page 40) property.

1.1.5.4.1 TCustomServerSocket.DoActivate Method

Opens or Closes the socket connection.

Pascal

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

C++

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

Description

DoActivate provides the underlying implementation of the Open (see page 9) and Close (see page 6) methods. When Value is true, DoActivate opens a listening connection to accept connection requests. When Value is false, DoActivate terminates the current listening connection and any connections to client sockets.

See Also

Active (see page 11)

Open (see page 9)

Close (see page 6)

1.1.5.4.2 TCustomServerSocket.GetGetSocketEvent Method

Returns the OnGetSocket (see page 44) event handler.

Pascal

```
function GetGetSocketEvent: TGetSocketEvent;
```

C++

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

Description

Applications cannot call this protected method. GetGetSocketEvent returns the OnGetSocket (see page 44) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

OnGetSocket (see page 44)

OnGetSocket (see page 126)

1.1.5.4.3 TCustomServerSocket.GetGetThreadEvent Method

Returns the OnGetThread (see page 44) event handler.

Pascal

```
function GetGetThreadEvent: TGetThreadEvent;
```

C++

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

Description

Applications cannot call this protected method. GetGetThreadEvent returns the OnGetThread (see page 44) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

OnGetThread (see page 44)

OnGetThread (see page 126)

1.1.5.4.4 TCustomServerSocket.GetOnClientError Method

Returns the OnClientError (see page 42) event handler.

Pascal

```
function GetOnClientError: TSocketErrorEvent;
```

C++

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

Description

Applications cannot call this protected method. GetOnClientError returns the OnClientError (see page 42) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

OnClientError (see page 42)

OnClientError (see page 124)

1.1.5.4.5 TCustomServerSocket.GetOnClientEvent Method

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

Pascal

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

C++

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

Description

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

See Also

OnClientRead (see page 43)

OnClientWrite (see page 43)

OnClientConnect (see page 41)

OnClientDisconnect (see page 42)

OnClientRead (see page 124)

OnClientWrite (see page 125)

OnClientConnect (see page 122)

OnClientDisconnect (see page 123)

1.1.5.4.6 TCustomServerSocket.GetOnThreadEnd Method

Returns the OnThreadEnd (see page 45) event handler.

Pascal

```
function GetOnThreadEnd: TThreadNotifyEvent;
```

C++

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

Description

Applications cannot call this protected method. GetOnThreadEnd returns the OnThreadEnd (see page 45) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

OnThreadEnd (see page 45)

OnThreadEnd (see page 127)

1.1.5.4.7 TCustomServerSocket.GetOnThreadStart Method

Returns the OnThreadStart (see page 45) event handler.

Pascal

```
function GetOnThreadStart: TThreadNotifyEvent;
```

C++

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

Description

Applications cannot call this protected method. GetOnThreadStart returns the OnThreadStart (see page 45) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

OnThreadStart (see page 45)

OnThreadStart (see page 127)

1.1.5.4.8 TCustomServerSocket.GetServerType Method

Returns the ServerType (see page 39) property.

Pascal

```
function GetServerType: TServerType;
```

C++

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

Description

Applications cannot call this protected method. GetServerType returns the value of the ServerType (see page 39) property of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

ServerType (see page 39)

ServerType (see page 120)

1.1.5.4.9 TCustomServerSocket.GetThreadCacheSize Method

Returns the value of the ThreadCacheSize (see page 40) property.

Pascal

```
function GetThreadCacheSize: Integer;
```

C++

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

Description

Applications cannot call this protected method. GetThreadCacheSize returns the value of the ThreadCacheSize (see page 40) property of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

ThreadCacheSize (see page 40)

ThreadCacheSize (see page 121)

1.1.5.4.10 TCustomServerSocket.SetGetSocketEvent Method

Sets the OnGetSocket (see page 44) event handler.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetGetSocketEvent sets the OnGetSocket (see page 44) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their OnGetSocket (see page 44) event.

See Also

OnGetSocket (see page 44)

OnGetSocket (see page 126)

1.1.5.4.11 TCustomServerSocket.SetGetThreadEvent Method

Sets the OnGetThread (see page 44) event handler.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetGetThreadEvent sets the OnGetThread (see page 44) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their OnGetThread (see page 44) event.

See Also

OnGetThread (see page 44)

OnGetThread (see page 126)

1.1.5.4.12 TCustomServerSocket.SetOnClientError Method

Sets the OnClientError (see page 42) event handler.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetOnClientError sets the OnClientError (see page 42) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their OnClientError (see page 42) event.

See Also

OnClientError (see page 42)

OnClientError (see page 124)

1.1.5.4.13 TCustomServerSocket.SetOnClientEvent Method

Sets the client event handlers of the associated TServerWinSocket (see page 109) object.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetOnClientEvent sets the various event handlers that respond to events from the server endpoint of a client connection. Server socket components use TServerWinSocket (see page 109) to implement these events.

See Also

OnClientRead (see page 43)

OnClientWrite (see page 43)

OnClientConnect (see page 41)

OnClientDisconnect (see page 42)

OnClientRead (see page 124)

OnClientWrite (see page 125)

OnClientConnect (see page 122)

OnClientDisconnect (see page 123)

1.1.5.4.14 TCustomServerSocket.SetOnThreadEnd Method

Sets the OnThreadEnd (see page 45) event handler.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetOnThreadEnd sets the OnThreadEnd (see page 45) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their OnThreadEnd (see page 45) event.

See Also

OnThreadEnd (see page 45)

OnThreadEnd (see page 127)

1.1.5.4.15 TCustomServerSocket.SetOnThreadStart Method

Sets the OnThreadStart (see page 45) event handler.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetOnThreadStart sets the OnThreadEnd (see page 45) event handler of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their OnThreadEnd (see page 45) event.

See Also

OnThreadStart (see page 45)

OnThreadEnd (see page 127)

1.1.5.4.16 TCustomServerSocket.SetServerType Method

Sets the ServerType (see page 39) property.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetServerType sets the ServerType (see page 39) property of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their ServerType (see page 39) property.

See Also

ServerType (see page 39)

ServerType (see page 120)

1.1.5.4.17 TCustomServerSocket.SetThreadCacheSize Method

Sets the ThreadCacheSize (see page 40) property.

Pascal

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

C++

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

Description

Applications cannot call this protected method. SetThreadCacheSize sets the ThreadCacheSize (see page 40) property of the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 109) to implement their ThreadCacheSize (see page 40) property.

See Also

ThreadCacheSize (see page 40)

ThreadCacheSize (see page 121)

1.1.5.5 TCustomServerSocket Properties

The properties of the TCustomServerSocket class are listed here.

Protected Properties

	Name	Description
	ServerType (see page 39)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 40)	Specifies the maximum number of threads that can be reused for new client connections.

1.1.5.5.1 TCustomServerSocket.ServerType Property

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

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the server socket. When ServerType is stThreadBlocking, the execution of the connection thread is suspended while reading or writing until all information has been transferred over the connection. The thread for each connection generates OnClientRead (see page 43) or OnClientWrite (see page 43) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over the socket connections asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead (see page 43) or OnClientWrite (see page 43) events occur when the client socket on the other end of one of the connections tries to send or receive information over the connection.

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

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

See Also

OnClientRead (see page 43)

OnClientWrite (see page 43)

TServerClientThread (see page 85)

1.1.5.5.2 TCustomServerSocket.ThreadCacheSize Property

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

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 39) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New (see page 3712) connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

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

See Also

ServerType (see page 39)

TServerClientThread (see page 85)

ActiveThreads (see page 119)

IdleThreads (see page 120)

1.1.5.6 TCustomServerSocket Events

The events of the TCustomServerSocket class are listed here.

Protected Events

	Name	Description
	OnClientConnect (see page 41)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 42)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 42)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 43)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 43)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 44)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to represent the server endpoint of a connection to a client socket.

	OnGetThread (see page 44)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnThreadEnd (see page 45)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 45)	Occurs when the execution thread for a connection to a client socket starts up.

1.1.5.6.1 TCustomServerSocket.OnClientConnect Event

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

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write (see page 3789) an OnClientConnect event handler to take specific action when a client socket completes the socket connection to the server socket. For example, the socket may start reading or writing over the connection in an OnClientConnect event handler.

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

1An OnListen (see page 51) event occurs just before the server socket is opened for listening.

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

3The server socket generates an OnGetSocket (see page 44) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 93) object for the server endpoint of the new connection is not created in an OnGetSocket (see page 44) event handler, the server socket creates one. The TServerWinSocket (see page 109) object continues to listen for other clients.

4An OnAccept (see page 49) event occurs, using the new TServerClientWinSocket (see page 93) object.

5If ServerType (see page 39) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 44) event occurs. If the OnGetThread (see page 44) event handler does not create a thread, the server socket creates a TServerClientThread (see page 85).

6If ServerType (see page 39) is stThreadBlocking, an OnThreadStart (see page 45) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 93) object and an OnClientConnect event occurs.

Note: If ServerType (see page 39) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientConnect event handler for TServerSocket (see page 97) is also set when setting the OnClientConnect event handler of the associated TServerWinSocket (see page 109).

See Also

OnAccept (see page 49)

OnGetSocket (see page 44)

OnGetThread (see page 44)

OnListen (see page 51)

OnThreadStart (see page 45)

ServerType (see page 39)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

GetClientThread (see page 116)

1.1.5.6.2 TCustomServerSocket.OnClientDisconnect Event

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

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write (see page 3789) an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client connection does not close the server socket. The server socket remains open and listening for client requests on its listening connection.

The TServerClientWinSocket (see page 93) that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType (see page 39) is stThreadBlocking, OnThreadEnd (see page 45) occurs after OnClientDisconnect.

Note: If ServerType (see page 39) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientDisconnect event handler for TServerSocket (see page 97) is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket (see page 109).

See Also

OnThreadEnd (see page 45)

ServerType (see page 39)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

GetClientThread (see page 116)

1.1.5.6.3 TCustomServerSocket.OnClientError Event

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

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write (see page 3789) an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError (see page 3) from being raised.

Note: If ServerType (see page 39) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientError event handler for TServerSocket (see page 97) is also set when setting the OnClientError event handler of the associated TServerWinSocket (see page 109).

See Also

ESocketError

ServerType (see page 39)

TServerClientWinSocket (see page 93)

GetClientThread (see page 116)

1.1.5.6.4 TCustomServerSocket.OnClientRead Event

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

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write (see page 3789) an OnClientRead event handler to write to the socket connection. If the ServerType (see page 39) property is stThreadBlocking, use a TWinSocketStream (see page 127) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType (see page 39) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 39)

receivebuf (see page 66)

ReceiveText (see page 67)

TServerClientWinSocket (see page 93)

GetClientThread (see page 116)

TWinSocketStream (see page 127)

1.1.5.6.5 TCustomServerSocket.OnClientWrite Event

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

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write (see page 3789) an OnClientWrite event handler to write to the socket connection. If the ServerType (see page 39) property is stThreadBlocking, use a TWinSocketStream (see page 127) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType (see page 39) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 39)

SendBuf (see page 68)

sendstream (see page 68)

SendText (see page 69)

GetClientThread (see page 116)

TWinSocketStream (see page 127)

1.1.5.6.6 TCustomServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write (see page 3789) an OnGetSocket event handler to create a specialized descendant of TServerClientWinSocket (see page 93) for the server socket component to use. Return the new TServerClientWinSocket (see page 93) object in the ClientSocket parameter. The Sender parameter is the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection.

See Also

TServerClientWinSocket (see page 93)

Create (see page 95)

TServerWinSocket (see page 109)

1.1.5.6.7 TCustomServerSocket.OnGetThread Event

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

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write (see page 3789) an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 85) for the connection to the client socket. Return the new TServerClientThread (see page 85) object in the SocketThread parameter.

Most applications that use thread-blocking server components will want to provide an OnGetThread event handler and implement a TServerClientThread (see page 85) that handles its own reading and writing in a thread-safe manner rather than relying on the default TServerClientThread (see page 85) which triggers OnClientRead (see page 43) and OnClientWrite (see page 43) events. This is because the OnClientRead (see page 43) and OnClientWrite (see page 43) event handlers reside on the server socket, which is in global memory.

The Sender parameter is the TServerWinSocket (see page 109) object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket (see page 93) object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 39)

TServerClientThread (see page 85)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.5.6.8 TCustomServerSocket.OnThreadEnd Event

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

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write (see page 3789) an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 42) event.

The Sender parameter is the TServerWinSocket (see page 109) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 42)

TServerWinSocket (see page 109)

1.1.5.6.9 TCustomServerSocket.OnThreadStart Event

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

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write (see page 3789) an OnThreadStart to initialize the TServerClientThread (see page 85) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 109) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is about to start execution.

See Also

TServerClientThread (see page 85)

TServerWinSocket (see page 109)

1.1.6 TCustomSocket Class

TCustomSocket is the base class for socket components.

Class Hierarchy

Pascal

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

C++

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

File

ScktComp

Description

Override TCustomSocket to create a socket component that can be assigned event handlers for the events and error conditions that occur. In addition to the properties and methods introduced by TAbstractSocket (see page 4) that enable an application to work with Windows sockets, TCustomSocket introduces several events that can be used by applications.

Do not create instances of TCustomSocket. Use a descendant of TCustomSocket to add the ability to communicate with other machines using TCP/IP. To create a socket that initiates the connection with another machine, use TClientSocket (see page 14). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 97).

See Also

TClientSocket (see page 14)

TCustomWinSocket (see page 54)

TServerSocket (see page 97)

1.1.6.1 TCustomSocket Members

The following tables list the members exposed by TCustomSocket.

Protected Events

	Name	Description
	OnAccept (see page 49)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 50)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 50)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 51)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 51)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 51)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 52)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 53)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 53)	Occurs when a client socket should write information to the socket connection.

Protected Methods

	Name	Description
	DoActivate (see page 6)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 48)	Generates an OnError (see page 51) event.
	Event (see page 48)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetService (see page 11)	Sets the Service (see page 13) property.

Protected Properties

	Name	Description
	Active (see page 11)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 12)	Specifies the IP address of the server system.
	Host (see page 12)	Specifies an alias for the IP address of the server system.
	Port (see page 13)	Specifies the ID number that identifies the server socket connection.
	Service (see page 13)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
	Open (see page 9)	Opens the socket connection.

1.1.6.2 TCustomSocket Methods

The methods of the TCustomSocket class are listed here.

Protected Methods

	Name	Description
	Error (see page 48)	Generates an OnError (see page 51) event.
	Event (see page 48)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.

1.1.6.2.1 TCustomSocket.Error Method

Generates an OnError (see page 51) event.

Pascal

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

C++

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

Description

Applications cannot call Error. It responds to error conditions by calling the OnError (see page 51) event handler if it is assigned. Override Error to add class-specific error processing or to block the OnError (see page 51) event.

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

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

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

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

See Also

OnError (see page 51)

1.1.6.2.2 TCustomSocket.Event Method

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

Pascal

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

C++

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

Description

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

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

The SocketEvent parameter indicates the event that occurred.

See Also

OnAccept (see page 49)

OnConnect (see page 50)

OnConnecting (see page 50)

OnDisconnect (see page 51)

OnListen (see page 51)

OnLookup (see page 52)

OnRead (see page 53)

OnWrite (see page 53)

1.1.6.3 TCustomSocket Events

The events of the TCustomSocket class are listed here.

Protected Events

	Name	Description
	OnAccept (see page 49)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 50)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 50)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 51)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 51)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 51)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 52)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 53)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 53)	Occurs when a client socket should write information to the socket connection.

1.1.6.3.1 TCustomSocket.OnAccept Event

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

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

Write (see page 3789) an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that will be used in the server endpoint of the connection. The port number, and possibly the IP address, will differ from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect (see page 50) event occurs for the client socket.

See Also

OnConnect (see page 50)

OnListen (see page 51)

1.1.6.3.2 TCustomSocket.OnConnect Event

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

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write (see page 3789) an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

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

1An OnLookup (see page 52) event occurs prior to an attempt to locate the server socket.

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

3An OnConnecting (see page 50) event occurs after the server socket is located.

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

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 49)

OnConnecting (see page 50)

OnLookup (see page 52)

1.1.6.3.3 TCustomSocket.OnConnecting Event

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

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

Write (see page 3789) an OnConnecting event handler for a client socket to take specific action just before the connection to a server socket is established. This is the first opportunity to obtain the actual port and IP address of the server endpoint of the connection that is about to form.

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

1An OnLookup (see page 52) event occurs prior to an attempt to locate the server socket.

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

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

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

5An OnConnect (see page 50) event occurs after the connection is established.

See Also

OnConnect (see page 50)

OnLookup (see page 52)

1.1.6.3.4 TCustomSocket.OnDisconnect Event

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

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write (see page 3789) an OnDisconnect event handler to take specific action when the connection to a server socket is about to be terminated. OnDisconnect occurs after the value of the Active (see page 11) property is set to false, but before the actual connection is closed.

See Also

Active (see page 11)

Close (see page 6)

OnConnect (see page 50)

1.1.6.3.5 TCustomSocket.OnError Event

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

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write (see page 3789) an OnError event handler to respond to errors that arise with the socket connection. Set the ErrorCode parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 3) from being raised.

See Also

ESocketError

1.1.6.3.6 TCustomSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

Write (see page 3789) an OnListen event handler for a server socket to take specific action just before the socket is opened for listening. OnListen occurs after Address (see page 12) and Port (see page 13) have been bound to the socket connection, but before it is opened. This is the last opportunity to make changes to the socket endpoint before it is opened for listening. Use the SocketHandle property of the Socket parameter for Windows API calls that change the socket.

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

See Also

Accept (see page 58)

Address (see page 12)

Port (see page 13)

1.1.6.3.7 TCustomSocket.OnLookup Event

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

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

Write (see page 3789) an OnLookup event handler for a client socket to take specific action just before attempting to locate a server socket. This is the first opportunity to make Windows API calls that affect the client properties of the socket, such as specifying a particular port number. Use the SocketHandle property of the Socket parameter for Windows API calls.

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

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

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

3An OnConnecting (see page 50) event occurs after the server socket is located.

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

5An OnConnect (see page 50) event occurs after the connection is established.

Note: Changing the Address (see page 12), Host (see page 12), Port (see page 13), or Service (see page 13) properties of socket component in an OnLookup event handler has no effect on the address or port used to locate a server socket. These properties must be correct before the Open (see page 9) method is called.

See Also

Address (see page 12)

Host (see page 12)

OnConnect (see page 50)

OnConnecting (see page 50)

Open (see page 9)

Port (see page 13)

Service (see page 13)

1.1.6.3.8 TCustomSocket.OnRead Event

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

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

Write (see page 3789) an OnRead event handler to read from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 127) object to read from the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual reading.

Note: Non-blocking sockets do not always receive an OnRead event for the last bit of data passed over the connection. When using a non-blocking socket, check for any unread data in the OnDisconnect (see page 51) event to make sure that everything is handled.

See Also

receivebuf (see page 66)

ReceiveText (see page 67)

TWinSocketStream (see page 127)

OnDisconnect (see page 51)

1.1.6.3.9 TCustomSocket.OnWrite Event

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

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

Write (see page 3789) an OnWrite to write from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 127) object to write to the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual writing.

See Also

SendBuf (see page 68)

sendstream (see page 68)

SendStreamThenDrop (see page 69)

SendText (see page 69)

TWinSocketStream (see page 127)

1.1.7 TCustomWinSocket Class

TCustomWinSocket is the base class for all Windows socket objects.

Class Hierarchy

Pascal

```
TCustomWinSocket = class;
```

C++

```
class TCustomWinSocket;
```

File

ScktComp

Description

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

A Windows socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Windows sockets provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 3165) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 1299) allow an application to form connections to other machines without being concerned with the details of the actual networking software.

Descendants of TCustomWinSocket represent different endpoints in socket connections. TClientWinSocket (see page 25) represents the client endpoint of a socket communication link to a server socket. TServerWinSocket (see page 109) represents the server endpoint of a listening connection. TServerClientWinSocket (see page 93) represents the server endpoint of a socket communication link to a client socket.

See Also

TAbstractSocket (see page 4)

TClientWinSocket (see page 25)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.7.1 TCustomWinSocket Members

The following tables list the members exposed by TCustomWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 58)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 60)	Opens a listening connection.
	DoOpen (see page 61)	Opens a connection to a remote socket.
	Error (see page 61)	Generates an OnErrorEvent (see page 79) event.
	Event (see page 62)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 62)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 70)	Sets the AsyncStyles property.

Public Events

	Name	Description
	OnErrorEvent (see page 79)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 79)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
V	Accept (see page 58)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 59)	Shuts down the socket connection if it is open.
V	Connect (see page 59)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 56)	Creates an instance of TCustomWinSocket (see page 54) from a Windows socket handle.
V	DefaultHandler (see page 60)	Processes message records.
V	Destroy (see page 56)	Destroys an instance of TCustomWinSocket (see page 54).
V	Disconnect (see page 60)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 63)	Opens a listening connection for a server socket.
	Lock (see page 64)	Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.
	LookupName (see page 64)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 65)	Returns the port number associated with a specific service.
	Open (see page 65)	Opens a connection to a remote socket.
V	Read (see page 66)	Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 66)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 67)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 67)	Reads a string from the socket connection.
	SendBuf (see page 68)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 68)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 69)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 69)	Writes the string S to the socket connection.
	Unlock (see page 70)	Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.
V	Write (see page 70)	Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 72)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 72)	Determines which asynchronous events the socket can receive.
R	Connected (see page 73)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 73)	Points to application-specific data associated with the socket.

	Handle (see page 73)	Specifies the handle of the window that receives notifications from the Windows socket.
	LocalAddress (see page 74)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 74)	Specifies the name of the local system.
	LocalPort (see page 75)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 75)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 76)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 76)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 77)	Specifies the name of the remote system.
	RemotePort (see page 77)	Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.
	SocketHandle (see page 78)	Specifies the Windows handle for the socket.

1.1.7.2 TCustomWinSocket.Create Constructor

Creates an instance of TCustomWinSocket (see page 54) from a Windows socket handle.

Pascal

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

C++

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

Description

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

After calling the inherited constructor, Create

Makes sure that Winsock.dll is initialized.

Allocates helper objects to manage multithreading issues.

Initializes ASyncStyles (see page 72) to [asRead, asWrite, asConnect, asClose]

Initializes the SocketHandle (see page 78) property to the ASocket parameter and sets the Connected (see page 73) property to reflect whether ASocket is the handle of a valid open Windows socket.

Initializes the Addr (see page 72) property.

See Also

Addr (see page 72)

ASyncStyles (see page 72)

Connected (see page 73)

SocketHandle (see page 78)

OnGetSocket (see page 44)

1.1.7.3 TCustomWinSocket.Destroy Destructor

Destroys an instance of TCustomWinSocket (see page 54).

Pascal

```
destructor Destroy; override;
```

C++

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

Description

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

See Also

System::TObject::Free (see page 3382)

TCustomSocket (see page 46)

1.1.7.4 TCustomWinSocket Methods

The methods of the TCustomWinSocket class are listed here.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 58)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 60)	Opens a listening connection.
	DoOpen (see page 61)	Opens a connection to a remote socket.
	Error (see page 61)	Generates an OnErrorEvent (see page 79) event.
	Event (see page 62)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 62)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 70)	Sets the AsyncStyles property.

Public Methods

	Name	Description
	Accept (see page 58)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 59)	Shuts down the socket connection if it is open.
	Connect (see page 59)	Provides the interface for a method that responds to connection notifications from client sockets.
	DefaultHandler (see page 60)	Processes message records.
	Disconnect (see page 60)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 63)	Opens a listening connection for a server socket.
	Lock (see page 64)	Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.
	LookupName (see page 64)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 65)	Returns the port number associated with a specific service.
	Open (see page 65)	Opens a connection to a remote socket.
	Read (see page 66)	Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 66)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 67)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 67)	Reads a string from the socket connection.
	SendBuf (see page 68)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 68)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 69)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 69)	Writes the string S to the socket connection.

	Unlock (see page 70)	Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.
	Write (see page 70)	Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

1.1.7.4.1 TCustomWinSocket.Accept Method

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

Pascal

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

C++

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

Description

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

The Accept method for TCustomWinSocket (see page 54) does nothing with the notification. Descendants of TCustomWinSocket (see page 54) that represent listening server sockets override this method to accept the connection to the client socket.

Accept is only called if the ASyncStyles (see page 72) property includes asAccept.

See Also

ASyncStyles (see page 72)

1.1.7.4.2 TCustomWinSocket.AsyncInitSocket Method

Looks up information or forms socket connections asynchronously.

Pascal

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

C++

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

Description

AsyncInitSocket is used internally to open a socket connection in non-blocking sockets. It is called by the Open (see page 65) or Listen (see page 63) method when that method's Block parameter is false.

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

See Also

ASyncStyles (see page 72)

ServerType (see page 120)

ClientType (see page 28)

Addr (see page 72)

Listen (see page 63)

LookupName (see page 64)

LookupService (see page 65)

Open (see page 65)

RemoteAddr (see page 76)

Address (see page 12)

Host (see page 12)

Port (see page 13)

Service (see page 13)

1.1.7.4.3 TCustomWinSocket.Close Method

Shuts down the socket connection if it is open.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to terminate an open socket connection. Close triggers an OnSocketEvent (see page 79) of type seDisconnect before the connection is shut down. If the Windows socket is a listening server socket, Close shuts down all open connections to client sockets before shutting down the listening connection.

See Also

Connected (see page 73)

OnSocketEvent (see page 79)

1.1.7.4.4 TCustomWinSocket.Connect Method

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

Pascal

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

C++

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

Description

Connect is called automatically in response to notifications that the client socket has successfully completed a connection to a server socket. The Socket parameter is the Windows socket handle for the socket that receives the notification.

The Connect method for TCustomWinSocket (see page 54) does nothing with the notification. Descendants of TCustomWinSocket (see page 54) that represent client sockets override this method to respond to connection notifications.

Connect is only called if the ASyncStyles (see page 72) property includes asConnect.

See Also

ASyncStyles (see page 72)

1.1.7.4.5 TCustomWinSocket.DefaultHandler Method

Processes message records.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

DefaultHandler overrides the inherited method in order to process all messages that are sent to the Windows socket object. DefaultHandler passes the messages to the Windows CallWindowProc function.

See Also

TMessage

1.1.7.4.6 TCustomWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

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

C++

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

Description

Disconnect is called by the Close (see page 59) method to terminate an individual socket connection.

Disconnect calls the Lock (see page 64) method to make sure the method is thread-safe. Then, it generates an OnSocketEvent (see page 79) of type seDisconnect. After the OnSocketEvent (see page 79), Disconnect closes the connection. Finally, it calls Unlock (see page 70) to end the critical section started by Lock (see page 64).

See Also

Close (see page 59)

Lock (see page 64)

OnSocketEvent (see page 79)

Unlock (see page 70)

1.1.7.4.7 TCustomWinSocket.DoListen Method

Opens a listening connection.

Pascal

```
procedure DoListen(QueueSize: Integer);
```

C++

```
__fastcall DoListen(int QueueSize);
```

Description

The socket uses DoListen internally to open a listening connection with a queue that can hold QueueSize client connection requests. DoListen handles those tasks that are common to opening a listening connection synchronously or asynchronously.

Do not call DoListen to open a listening connection. Instead, call the public Listen (see page 63) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Listen (see page 63)

1.1.7.4.8 TCustomWinSocket.DoOpen Method

Opens a connection to a remote socket.

Pascal

```
procedure DoOpen;
```

C++

```
__fastcall DoOpen();
```

Description

The socket uses DoOpen internally to open a connection to a remote socket. DoOpen handles those tasks that are common to establishing a connection synchronously or asynchronously.

Do not call DoOpen to open a connection. Instead, call the public Open (see page 65) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Open (see page 65)

1.1.7.4.9 TCustomWinSocket.Error Method

Generates an OnErrorEvent (see page 79) event.

Pascal

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

C++

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

Description

Error is called automatically when the Windows socket object receives error notifications. It generates the OnErrorEvent (see page 79) event. When overriding this method, be sure to call the inherited method, because socket components rely on the OnErrorEvent (see page 79) event for their own error handling.

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

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

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

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

See Also

OnErrorEvent (see page 79)

1.1.7.4.10 TCustomWinSocket.Event Method

Pascal

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

C++

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

Description

This is Event, a member of class TCustomWinSocket.

1.1.7.4.11 TCustomWinSocket.InitSocket Method

Generates a windows Internet socket address structure from a description of the desired socket port.

Pascal

```
function InitSocket(const Name: string; const Address: string; const Service: string; Port:
Word; Client: Boolean): TSockAddrIn;
```

C++

```
__fastcall TSockAddrIn InitSocket(const UnicodeString Name, const UnicodeString Address, const
UnicodeString Service, Word Port, Boolean Client);
```

Description

Applications can not call InitSocket. It is called by the Open (see page 65) or Listen (see page 63) method to obtain a Windows Internet socket address structure for the parameters obtained from a socket component.

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

The host portion of the Windows Internet socket address is obtained from the host name if it is given. Otherwise the Address

parameter is used. If Client is true, either the Name or the Address parameter must be specified. If Client is false, they may both be left blank to specify a server that listens on multiple addresses.

If port number for the Windows Internet socket address is obtained from the Service parameter if it is given. Otherwise, the Port parameter is used. One of these two parameters must be specified.

See Also

Addr (see page 72)

Listen (see page 63)

LookupName (see page 64)

LookupService (see page 65)

Open (see page 65)

RemoteAddr (see page 76)

Address (see page 12)

Host (see page 12)

Port (see page 13)

Service (see page 13)

AsyncInitSocket (see page 58)

1.1.7.4.12 TCustomWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(const Name: string; const Address: string; const Service: string; Port: Word;
QueueSize: Integer; Block: Boolean = True);
```

C++

```
__fastcall Listen(const UnicodeString Name, const UnicodeString Address, const UnicodeString
Service, Word Port, int QueueSize, Boolean Block = True);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 79) of type seListen. Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 73) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component.

QueueSize indicates how many client requests the listening connection can hold. TServerSocket (see page 97) sets the QueueSize to 5.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 73)

InitSocket (see page 62)

OnSocketEvent (see page 79)

Address (see page 12)

Host (see page 12)

Port (see page 13)

Service (see page 13)

TServerSocket (see page 97)

1.1.7.4.13 TCustomWinSocket.Lock Method

Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock before beginning a section of code that is not thread-safe. Call Unlock (see page 70) when the critical section of code is completed. Unless absolutely necessary, do not use Lock to lock lengthy operations such as reading and writing over a blocking connection. Locking lengthy operations can severely impact performance, because all execution threads in the application that use the Windows socket object must wait until Unlock (see page 70) is called.

See Also

TThread

Unlock (see page 70)

1.1.7.4.14 TCustomWinSocket.LookupName Method

Generates the portion of a Windows Internet socket address structure that describes the host.

Pascal

```
function LookupName(const name: string): TInAddr;
```

C++

```
__fastcall TInAddr LookupName(const UnicodeString name);
```

Description

Call LookupName to obtain the Internet address associated with the system identified by the name parameter. LookupName is used by InitSocket (see page 62) to generate the portion of the Windows Internet socket address structure that identifies the host.

See Also

InitSocket (see page 62)

Host (see page 12)

1.1.7.4.15 TCustomWinSocket.LookupService Method

Returns the port number associated with a specific service.

Pascal

```
function LookupService(const service: string): Integer;
```

C++

```
__fastcall int LookupService(const UnicodeString service);
```

Description

Call LookupService to obtain the standard port number for the service named by the service parameter. Windows provides a number of standard service names such as ftp, http, finger, or time. Each standard service is associated, by convention, with a specific port number. These standard port numbers allow client sockets to indicate the desired service by requesting a specific port number. They allow server sockets to listen on an expected port number for the service offered. Servers can specify additional services and their associated ports in a SERVICES file. For more information on the SERVICES file, see the Microsoft documentation for Windows sockets.

InitSocket (see page 62) calls LookupService to obtain the port number for the Service specified by a socket component.

See Also

InitSocket (see page 62)

Service (see page 13)

1.1.7.4.16 TCustomWinSocket.Open Method

Opens a connection to a remote socket.

Pascal

```
procedure Open(const Name: string; const Address: string; const Service: string; Port: Word;
Block: Boolean = True);
```

C++

```
__fastcall Open(const UnicodeString Name, const UnicodeString Address, const UnicodeString
Service, Word Port, Boolean Block = True);
```

Description

Client socket components call Open to open a connection to a server socket.

Open creates the Windows socket that will connect to the server socket, and generates an OnSocketEvent (see page 79) of type seLookup. After the OnSocketEvent (see page 79), Open binds the Windows Internet socket address derived from the parameters to the socket, locating the server socket. Open then generates an OnSocketEvent (see page 79) of type seConnecting. When the server socket accepts the connection, Open completes the connection to the server socket. Then it generates an OnSocketEvent (see page 79) of type seConnect. After the OnSocketEvent (see page 79), the Connected (see page 73) property is set to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the client socket component.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 73)

InitSocket (see page 62)

OnSocketEvent (see page 79)

Address (see page 12)

Host (see page 12)

Port (see page 13)

Service (see page 13)

TClientSocket (see page 14)

1.1.7.4.17 TCustomWinSocket.Read Method

Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.

Pascal

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

C++

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

Description

Read is called in response to asynchronous notifications that the socket connection is ready for reading. The Windows socket can then read from the connection in the OnSocketEvent (see page 79) event handler or the socket component can read from the connection in an OnRead or OnClientRead event handler. The event handler can use the ReceiveBuf (see page 66) method or the ReceiveText (see page 67) method to read the information from the connection.

Note: Read is only called for asynchronous read notifications in a non-blocking socket. Blocking sockets must use a TWinSocketStream (see page 127) object for reading. The TWinSocketStream (see page 127) object waits for the remote socket to be ready before transferring information.

See Also

ASyncStyles (see page 72)

OnSocketEvent (see page 79)

receivebuf (see page 66)

ReceiveText (see page 67)

OnRead (see page 53)

OnClientRead (see page 43)

TWinSocketStream (see page 127)

1.1.7.4.18 TCustomWinSocket.ReceiveBuf Method

Pascal

```
function ReceiveBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int Count);
```

Description

This is ReceiveBuf, a member of class TCustomWinSocket.

1.1.7.4.19 TCustomWinSocket.ReceiveLength Method

Returns the number of bytes ready to be sent over the socket connection.

Pascal

```
function ReceiveLength: Integer;
```

C++

```
__fastcall int ReceiveLength();
```

Description

Call ReceiveLength to determine the amount of information to read over the socket connection in response to an asynchronous read notification.

Note: ReceiveLength is not guaranteed to be accurate for streaming socket connections.

See Also

ASyncStyles (see page 72)

receivebuf (see page 66)

1.1.7.4.20 TCustomWinSocket.ReceiveText Method

Reads a string from the socket connection.

Pascal

```
function ReceiveText: string;
```

C++

```
__fastcall UnicodeString ReceiveText();
```

Description

Use ReceiveText to read a string from the socket connection in the OnSocketEvent (see page 79) event handler of a Windows socket object or in the OnRead or OnClientRead event handler of a socket component. ReceiveText returns the string that was read.

ReceiveText only works in response to a read notification to a non-blocking windows socket. Blocking sockets must use a TWinSocketStream (see page 127) for reading. The TWinSocketStream (see page 127) object waits for the remote socket to be ready before transferring information.

See Also

ASyncStyles (see page 72)

OnSocketEvent (see page 79)

receivebuf (see page 66)

OnRead (see page 53)

OnClientRead (see page 43)

TWinSocketStream (see page 127)

1.1.7.4.21 TCustomWinSocket.SendBuf Method

Writes Count bytes to the socket connection from the Buf parameter.

Pascal

```
function SendBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int Count);
```

Description

Use SendBuf to write to the socket connection. Call this method from the OnSocketEvent (see page 79) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, Use SendBuf to write when a connection is first formed when the socket does not expect notification that the socket on the other end of the connection is expecting to read.

For non-blocking sockets, the data is sent to the WinSock DLL which has it's own internal buffers. If the WinSock can accept additional data, SendBuf returns immediately with the number of bytes queued. If the WinSock internal buffer space is not able to accept the buffer being sent, SendBuf returns -1 and no data is queued at all. In this case, wait a bit for the WinSock to have a chance to send out already-queued data; then try again.

For blocking sockets, SendBuf returns the number of bytes actually written.

If an error occurs while writing to the connection, SendBuf terminates the connection and raises an ESocketError (see page 3) exception.

See Also

ESocketError

OnSocketEvent (see page 79)

sendstream (see page 68)

SendText (see page 69)

OnWrite (see page 53)

OnClientWrite (see page 43)

TWinSocketStream (see page 127)

1.1.7.4.22 TCustomWinSocket.SendStream Method

Pascal

```
function SendStream(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStream(TStream AStream);
```

Description

This is SendStream, a member of class TCustomWinSocket.

1.1.7.4.23 TCustomWinSocket.SendStreamThenDrop Method

Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.

Pascal

```
function SendStreamThenDrop(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStreamThenDrop(TStream AStream);
```

Description

Use SendStreamThenDrop to write to a socket connection when the connection is no longer needed after the writing is complete. SendStream (see page 68) reads information from the stream indicated by AStream and writes it to the socket connection, shutting down the connection when it is done. The value returned by SendStream (see page 68) indicates whether any information was successfully written to the connection.

Note: The Stream passed as a parameter to SendStream (see page 68) becomes "owned" by the windows socket object. The Windows socket object frees the stream when it is finished with it. Do not attempt to free the stream after it has been passed as a parameter.

See Also

Disconnect (see page 60)

SendBuf (see page 68)

sendstream (see page 68)

SendText (see page 69)

TStream

TWinSocketStream (see page 127)

1.1.7.4.24 TCustomWinSocket.SendText Method

Writes the string S to the socket connection.

Pascal

```
function SendText(const S: AnsiString): Integer;
```

C++

```
__fastcall int SendText(const AnsiString S);
```

Description

Use SendText to write a string to the socket connection. The writing may occur in the OnSocketEvent (see page 79) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, SendText may write from a socket that is expected to write to the connection without a notification to signal the connection's readiness to read. If an error occurs while writing to the connection, SendText terminates the connection and raises an ESocketError (see page 3) exception.

SendText returns the number of bytes sent. Note that this may be less than the length of the string S if the socket is nonblocking.

See Also

ESocketError

OnSocketEvent (see page 79)

SendBuf (see page 68)

sendstream (see page 68)

OnWrite (see page 53)

OnClientWrite (see page 43)

TWinSocketStream (see page 127)

1.1.7.4.25 TCustomWinSocket.SetAsyncStyles Method

Sets the AsyncStyles property.

Pascal

```
procedure SetAsyncStyles(Value: TASyncStyles);
```

C++

```
__fastcall SetAsyncStyles(TASyncStyles Value);
```

Description

SetAsyncStyles implements the AsyncStyles property. Override SetAsyncStyles to limit the value of AsyncStyles to notifications that are appropriate to a particular type of Windows socket.

See Also

ASyncStyles (see page 72)

1.1.7.4.26 TCustomWinSocket.Unlock Method

Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a section of code that is not thread-safe. Every call to Unlock should match a call to the Lock (see page 64) method at the beginning of the critical section of code.

See Also

Lock (see page 64)

TThread

1.1.7.4.27 TCustomWinSocket.Write Method

Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

Pascal

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

C++

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

Description

Write is called in response to asynchronous notifications that the socket connection is ready for the socket to send information. The Windows socket can write to the connection in the OnSocketEvent (see page 79) event handler or the socket component can write to the connection in an OnWrite or OnClientWrite event handler. The event handler can use the SendBuf (see page 68) method, the SendStream (see page 68) method, or the SendText (see page 69) method to write the information to the connection.

Note: Write is only called for asynchronous write notifications in a non-blocking socket. Blocking sockets must use their own mechanisms for determining when the connection is ready for the socket to write.

See Also

ASyncStyles (see page 72)

OnSocketEvent (see page 79)

SendBuf (see page 68)

sendstream (see page 68)

SendText (see page 69)

OnWrite (see page 53)

OnClientWrite (see page 43)

TWinSocketStream (see page 127)

1.1.7.5 TCustomWinSocket Properties

The properties of the TCustomWinSocket class are listed here.

Public Properties

	Name	Description
R	Addr (see page 72)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 72)	Determines which asynchronous events the socket can receive.
R	Connected (see page 73)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 73)	Points to application-specific data associated with the socket.
R	Handle (see page 73)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 74)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 74)	Specifies the name of the local system.
R	LocalPort (see page 75)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 75)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 76)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 76)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 77)	Specifies the name of the remote system.
R	RemotePort (see page 77)	Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.
R	SocketHandle (see page 78)	Specifies the Windows handle for the socket.

1.1.7.5.1 TCustomWinSocket.Addr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.

Pascal

```
property Addr: TSockAddrIn;
```

C++

```
__property TSockAddrIn Addr;
```

Description

Use Addr to represent the port used by the socket when working directly with Windows socket API calls. For client sockets, the value of Addr represents the target remote port. For server sockets, Addr is determined by the values of the LocalAddress (see page 74) and LocalPort (see page 75) properties.

See Also

LocalAddress (see page 74)

LocalPort (see page 75)

RemoteAddr (see page 76)

1.1.7.5.2 TCustomWinSocket.ASyncStyles Property

Determines which asynchronous events the socket can receive.

Pascal

```
property ASyncStyles: TAsyncStyles;
```

C++

```
__property TAsyncStyles ASyncStyles;
```

Description

Use ASyncStyles to determine what sorts of notifications the socket needs to respond to. Set ASyncStyles to change the notifications that the socket receives from the socket connection. ASyncStyles is a set drawn from the following values:

Value	Meaning
asRead	The socket receives notification that the connection is ready for reading.
asWrite	The socket receives notification that the connection is ready for writing.
asOOB	The socket receives notification when out-of-band data arrives.
asAccept	The socket receives notification when another socket requests a connection.
asConnect	The socket receives notification when a communication link to another socket is opened.
asClose	The socket receives notification when a communication link to another socket is terminated.

Any notification specified by ASyncStyles arrives as a window message to the Handle (see page 73) property.

Note: If the socket is a non-blocking socket, then ASyncStyles should include asRead and asWrite so that the socket will be informed of asynchronous reading and writing events.

See Also

Handle (see page 73)

ClientType (see page 28)

ServerType (see page 120)

1.1.7.5.3 TCustomWinSocket.Connected Property

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

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read (see page 66) Connected before trying to use or change the Windows socket connection. Connected indicates whether the socket connection has been established. When Connected is true, the Windows socket is open and available for use. When Connected is false, the Windows socket is closed and can be changed.

To open a client Windows socket, use the Open (see page 65) method. To open a server Windows socket, use the Listen (see page 63) method. The Windows socket that represents the server endpoint of a connection to a client socket is opened in its constructor.

To close a Windows socket, use the Close (see page 59) method.

See Also

Open (see page 65)

Close (see page 59)

Listen (see page 63)

Active (see page 11)

1.1.7.5.4 TCustomWinSocket.Data Property

Points to application-specific data associated with the socket.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information specific to the socket connection with the Windows socket object. For example, Data can be used to store access and authentication information used by a server socket to evaluate client connection requests.

Data can store information in a thread-safe manner that would otherwise require the use of global variables. This is useful given the multi-threaded nature of many client-server applications.

1.1.7.5.5 TCustomWinSocket.Handle Property

Specifies the handle of the window that receives notifications from the Windows socket.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Use Handle to send messages to the Windows socket or to subclass the Window procedure that receives messages of asynchronous events from the socket connection.

See Also

ASyncStyles (see page 72)

1.1.7.5.6 TCustomWinSocket.LocalAddress Property

Specifies the IP address of the local system that was bound to the socket connection.

Pascal

```
property LocalAddress: string;
```

C++

```
__property UnicodeString LocalAddress;
```

Description

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

```
123.197.1.2
```

Each address identifies the system and indicates the types of interfaces supported by the socket. A single system (machine) can support multiple addresses. LocalAddress is the address from among the possible addresses supported by the local system that is bound to the socket connection.

A single LocalAddress can be used simultaneously by multiple socket connections. Each of those connections must have a unique LocalPort (see page 75) number to distinguish it from other connections to the same LocalAddress.

See Also

LocalHost (see page 74)

LocalPort (see page 75)

RemoteAddress (see page 76)

Address (see page 12)

1.1.7.5.7 TCustomWinSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: string;
```

C++

```
__property UnicodeString LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the Windows socket. LocalHost is a string

containing the domain name and service of the local socket endpoint, such as

`http://www.wSite.Com`

LocalHost provides an alias for the local IP address given in the LocalAddress (see page 74) property. Most Intranets provide host names for the IP addresses of systems on the net. On Windows 95 and NT machines, if a host name is not available, create one for the local IP address by entering the name into the HOSTS file. See the Microsoft documentation on Windows sockets for more information on the HOSTS file.

See Also

LocalAddress (see page 74)

RemoteHost (see page 77)

Host (see page 12)

1.1.7.5.8 TCustomWinSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: Integer;
```

C++

```
__property int LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Port numbers allow a single system, identified by the LocalAddress (see page 74) property, to host multiple connections simultaneously. Each combination of LocalPort and LocalAddress (see page 74) can only be bound to a single socket connection. This combination is represented by the Addr (see page 72) property.

Many values of LocalPort are associated by convention with a particular service such as ftp or http. To determine whether the value of LocalPort is associated with a specific service, use the LookupService (see page 65) method.

See Also

Addr (see page 72)

LocalAddress (see page 74)

LookupService (see page 65)

RemotePort (see page 77)

1.1.7.5.9 TCustomWinSocket.LookupState Property

Indicates any asynchronous activities occurring while opening a connection.

Pascal

```
property LookupState: TLookupState;
```

C++

```
__property TLookupState LookupState;
```

Description

LookupState is used internally to coordinate asynchronous activities while opening a connection. The AsyncInitSocket (see page 58) method sets the value of LookupState to indicate when it is looking up various types of information. The following table lists the possible values:

Value	Meaning
lsIdle	The socket is not currently looking up any information asynchronously.
lsLookupAddress	The socket is looking up an IP address asynchronously and has not yet received the result.
lsLookupService	The socket is looking up a Service asynchronously and has not yet received the result.

See Also

AsyncInitSocket (see page 58)

Open (see page 65)

Listen (see page 63)

1.1.7.5.10 TCustomWinSocket.RemoteAddr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.

Pascal

```
property RemoteAddr: TSockAddrIn;
```

C++

```
__property TSockAddrIn RemoteAddr;
```

Description

When working directly with Windows socket API calls, use RemoteAddr to represent the port used by the socket on the other end of the connection. The value of RemoteAddr is determined by the values of the RemoteAddress (see page 76) and RemotePort (see page 77) properties.

See Also

Addr (see page 72)

RemoteAddress (see page 76)

RemotePort (see page 77)

1.1.7.5.11 TCustomWinSocket.RemoteAddress Property

Specifies the IP address of the remote system that was bound to the socket connection.

Pascal

```
property RemoteAddress: string;
```

C++

```
__property UnicodeString RemoteAddress;
```

Description

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

`123.197.1.2`

Each address identifies the system on the other end of the socket connection and indicates the types of interfaces supported by the remote socket. A single system can support multiple addresses. RemoteAddress is the address from among the possible addresses supported by the remote system that is bound to the socket connection.

A single RemoteAddress may be participating in multiple socket connections. Each of those connections has a unique RemotePort (see page 77) number to distinguish it from other connections to the same RemoteAddress. The RemoteAddr (see page 76) property uniquely identifies the RemoteAddress and RemotePort (see page 77) combination used by the socket connection.

See Also

LocalAddress (see page 74)

RemoteAddr (see page 76)

RemoteHost (see page 77)

RemotePort (see page 77)

Address (see page 12)

1.1.7.5.12 TCustomWinSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: string;
```

C++

```
__property UnicodeString RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as

`http://www.wSite.Com`

LocalHost (see page 74) provides an alias for the local IP address given in the RemoteAddress (see page 76) property.

See Also

LocalHost (see page 74)

RemoteAddress (see page 76)

Address (see page 12)

Host (see page 12)

1.1.7.5.13 TCustomWinSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.

Pascal

```
property RemotePort: Integer;
```

C++

```
__property int RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteAddress (see page 76) or RemoteHost (see page 77) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteAddress (see page 76) can only be bound to a single socket connection. This combination is represented by the RemoteAddr (see page 76) property.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupService (see page 65) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

LookupState (see page 75)

RemoteAddr (see page 76)

RemoteAddress (see page 76)

RemoteHost (see page 77)

Port (see page 13)

1.1.7.5.14 TCustomWinSocket.SocketHandle Property

Specifies the Windows handle for the socket.

Pascal

```
property SocketHandle: TSocket;
```

C++

```
__property TSocket SocketHandle;
```

Description

Use SocketHandle for Windows socket API calls that require the handle to the socket. When the Connected (see page 73) property is false, SocketHandle is INVALID_SOCKET.

SocketHandle is set to a valid Windows socket handle by the Listen (see page 63) or Open (see page 65) method.

See Also

Connected (see page 73)

Create (see page 56)

Listen (see page 63)

Open (see page 65)

1.1.7.6 TCustomWinSocket Events

The events of the TCustomWinSocket class are listed here.

Public Events

	Name	Description
	OnErrorEvent (see page 79)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 79)	Occurs when the Windows socket receives notification of asynchronous events.

1.1.7.6.1 TCustomWinSocket.OnErrorEvent Event

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

Pascal

```
property OnErrorEvent: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnErrorEvent;
```

Description

Socket components provide an OnErrorEvent event handler that converts error notifications into error events on the socket component. Most applications will therefore not provide an event handler for OnErrorEvent.

See Also

ESocketError

InitSocket (see page 8)

ASyncStyles (see page 72)

1.1.7.6.2 TCustomWinSocket.OnSocketEvent Event

Occurs when the Windows socket receives notification of asynchronous events.

Pascal

```
property OnSocketEvent: TSocketEventEvent;
```

C++

```
__property TSocketEventEvent OnSocketEvent;
```

Description

Socket components provide an OnSocketEvent event handler that converts Windows socket notifications into events on the socket component. Most applications will therefore not provide an event handler for OnSocketEvent. Replacing this event handler can prevent some expected events from occurring.

The Sender parameter is the Windows socket object that owns the event handler. Usually, this is the same as the Socket parameter.

The Socket parameter indicates the Windows socket that generates or receives notification of the event.

The SocketEvent parameter indicates the event that occurred.

See Also

ASyncStyles (see page 72)

InitSocket (see page 8)

1.1.8 TServerAcceptThread Class

Class Hierarchy

Pascal

```
TServerAcceptThread = class(TThread);
```

C++

```
class TServerAcceptThread : public TThread;
```

File

ScktComp

Description

This is class ScktComp.TServerAcceptThread.

1.1.8.1 TServerAcceptThread Members

The following tables list the members exposed by TServerAcceptThread.

Public Methods

	Name	Description
	Create (see page 80)	This is Create, a member of class TServerAcceptThread.
V	Execute (see page 81)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Properties

	Name	Description
R	ServerSocket (see page 85)	This is ServerSocket, a member of class TServerAcceptThread.

1.1.8.2 TServerAcceptThread.Create Constructor

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerWinSocket);
```

C++

```
__fastcall TServerAcceptThread(Boolean CreateSuspended, TServerWinSocket ASocket);
```

Description

This is Create, a member of class TServerAcceptThread.

1.1.8.3 TServerAcceptThread Methods

The methods of the TServerAcceptThread class are listed here.

Public Methods

	Name	Description
	Execute (see page 81)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

1.1.8.3.1 TServerAcceptThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
```

```
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended - secondprocess does not run yet
```

```
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
```

```
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

1.1.8.4 TServerAcceptThread Properties

The properties of the TServerAcceptThread class are listed here.

Public Properties

	Name	Description
	ServerSocket (see page 85)	This is ServerSocket, a member of class TServerAcceptThread.

1.1.8.4.1 TServerAcceptThread.ServerSocket Property

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

This is ServerSocket, a member of class TServerAcceptThread.

1.1.9 TServerClientThread Class

TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.

Class Hierarchy

Pascal

```
TServerClientThread = class(TThread);
```

C++

```
class TServerClientThread : public TThread;
```

File

SckiComp

Description

TServerWinSocket (see page 109) objects spawn TServerClientThread objects to handle separate client connections in separate execution threads. This allows server sockets to perform slow reading and writing operations with client sockets without adversely affecting the performance on other connections.

TServerClientThread introduces new properties to

Access the TServerWinSocket (see page 109) object that spawned the thread and the TServerClientWinSocket (see page 93) that is connected to the client socket.

Store and access thread-specific data.

Determine if the thread is stored in a thread cache maintained by the TServerWinSocket (see page 109).

Applications that receive frequent client requests will want to derive a descendant class from TServerClientThread to handle reading and writing to separate clients. This is because TServerClientThread objects use the OnClientRead and OnClientWrite events of the associated server socket. These events are not thread-local.

See Also

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.9.1 TServerClientThread Members

The following tables list the members exposed by TServerClientThread.

Protected Methods

	Name	Description
	ClientExecute (see page 88)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
	DoTerminate (see page 88)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 89)	Indicates whether the thread should execute again for another client.
	Error (see page 89)	Generates an OnClientError event for the associated server socket.
	Event (see page 90)	This is Event, a member of class TServerClientThread.
	Execute (see page 90)	Manages the caching of server client threads and calls the ClientExecute (see page 88) method to run the thread for individual client connections.
	HandleException (see page 90)	Displays an error message using the main VCL thread.
	ReActivate (see page 91)	This is ReActivate, a member of class TServerClientThread.
	StartConnect (see page 91)	Indicates when the thread is ready to execute.

Public Methods

	Name	Description
	Create (see page 86)	Creates an instance of TServerClientThread (see page 85).
	Destroy (see page 87)	Destroys the server client thread object and releases the memory allocated to it.

Public Properties

	Name	Description
	ClientSocket (see page 91)	Specifies the TServerClientWinSocket (see page 93) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 92)	Points to application-specific data associated with the thread.
	KeepInCache (see page 92)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 93).
	ServerSocket (see page 93)	Specifies the TServerWinSocket (see page 109) object that manages all the TServerClientThread (see page 85) objects for its client connections.

1.1.9.2 TServerClientThread.Create Constructor

Creates an instance of TServerClientThread (see page 85).

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerClientWinSocket);
```

C++

```
__fastcall TServerClientThread(Boolean CreateSuspended, TServerClientWinSocket ASocket);
```

Description

Use Create to create an instance of TServerClientThread (see page 85) for a connection to a client socket, and add the thread to the cache maintained by the associated listening socket.

Initializes FreeOnTerminate to true.

Initializes Priority to tpHigher.

Initializes ClientSocket (see page 91) to the ASocket parameter and ServerSocket (see page 93) to ASocket.ServerWinSocket.

Starts the thread executing.

Adds the thread to the cache maintained by ServerSocket (see page 93).

Sets the OnErrorEvent and OnSocketEvent event handlers of ClientSocket (see page 91) so that events are sent to ServerSocket (see page 93).

Signals when initialization is complete and the thread is ready for reading from or writing to the client connection.

See Also

ClientSocket (see page 91)

FreeOnTerminate

Priority

ServerSocket (see page 93)

OnErrorEvent (see page 79)

OnSocketEvent (see page 79)

1.1.9.3 TServerClientThread.Destroy Destructor

Destroys the server client thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientThread();
```

Description

Do not call Destroy directly in an application. ServerClientThread objects are freed by the TServerWinSocket (see page 109) object that manages them.

Destroy frees the TServerClientWinSocket (see page 93) object that encapsulates the connection to the client socket before calling the inherited destructor.

See Also

ClientSocket (see page 91)

ServerSocket (see page 93)

1.1.9.4 TServerClientThread Methods

The methods of the TServerClientThread class are listed here.

Protected Methods

	Name	Description
	ClientExecute (see page 88)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
	DoTerminate (see page 88)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 89)	Indicates whether the thread should execute again for another client.
	Error (see page 89)	Generates an OnClientError event for the associated server socket.
	Event (see page 90)	This is Event, a member of class TServerClientThread.
	Execute (see page 90)	Manages the caching of server client threads and calls the ClientExecute (see page 88) method to run the thread for individual client connections.

	HandleException (see page 90)	Displays an error message using the main VCL thread.
	ReActivate (see page 91)	This is ReActivate, a member of class TServerClientThread.
	StartConnect (see page 91)	Indicates when the thread is ready to execute.

1.1.9.4.1 TServerClientThread.ClientExecute Method

Simulates OnClientRead and OnClientWrite events for the associated server socket.

Pascal

```
procedure ClientExecute; virtual;
```

C++

```
virtual __fastcall ClientExecute();
```

Description

TServerClientThread (see page 85) calls ClientExecute from its Execute (see page 90) method. ClientExecute checks whether there is any information to read from the associated socket connection, and if so, simulates an OnClientRead event on the associated server socket. If there is no information sent from the client, ClientExecute simulates an OnClientWrite event.

Because the server socket that receives these events is not local to the thread, the events occur within the main VCL thread using the Synchronize method. This method can be too slow if the server socket has many client connections, because each thread must wait for any other thread to finish reading or writing. When a server socket receives many client connection requests, override ClientExecute to handle the reading or writing using a thread-local instance of TWinSocketStream (see page 127). In the ClientExecute method, be sure to check the Terminated property periodically.

Note: Do not use the properties and methods of global objects directly in the ClientExecute method. Instead, separate the use of objects that are not thread-local into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Execute (see page 90)

Synchronize

TWinSocketStream (see page 127)

Terminated

DoTerminate (see page 88)

1.1.9.4.2 TServerClientThread.DoTerminate Method

Removes the thread from the list of active threads maintained by the associated server socket.

Pascal

```
procedure DoTerminate; override;
```

C++

```
virtual __fastcall DoTerminate();
```

Description

DoTerminate overrides the inherited method to suppress the OnTerminate event. Instead, it removes the thread from the Connections property of the associated TServerWinSocket (see page 109). The thread is then either added to the server socket's thread cache or destroyed.

See Also

ClientExecute (see page 88)

Terminated

Connections (see page 119)

1.1.9.4.3 TServerClientThread.EndConnect Method

Indicates whether the thread should execute again for another client.

Pascal

```
function EndConnect: Boolean;
```

C++

```
__fastcall Boolean EndConnect();
```

Description

The Execute (see page 90) method calls EndConnect when the ClientExecute (see page 88) method finishes. It frees all memory associated with the current client connection and checks whether the Thread is being reused for another client connection. If the thread should be reused, EndConnect returns true to indicate that the ClientExecute (see page 88) method should be called again.

See Also

ClientExecute (see page 88)

Execute (see page 90)

1.1.9.4.4 TServerClientThread.Error Method

Generates an OnClientError event for the associated server socket.

Pascal

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

C++

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

Description

TServerClientThread (see page 85) assigns the Error method as an event handler for the OnErrorEvent method of the associated TClientWinSocket (see page 25) object. This allows errors received from the client to generate OnClientError events on the server socket.

Warning: Error is not thread-safe. It should not be called from the server client thread. It is only intended for use by the associated client socket, which is not local to the server client thread.

See Also

OnClientError (see page 42)

OnErrorEvent (see page 79)

1.1.9.4.5 TServerClientThread.Event Method

Pascal

```
procedure Event(SocketEvent: TSocketEvent); virtual;
```

C++

```
virtual __fastcall Event(TSocketEvent SocketEvent);
```

Description

This is Event, a member of class TServerClientThread.

1.1.9.4.6 TServerClientThread.Execute Method

Manages the caching of server client threads and calls the ClientExecute (see page 88) method to run the thread for individual client connections.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Execute is called automatically when the thread runs. A thread runs when Create (see page 86) is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Do not override the Execute method to implement the thread function of a TServerClientThread (see page 85) descendant. Instead, override the ClientExecute (see page 88) method, which is run after the thread is initialized for a particular client connection.

See Also

ClientExecute (see page 88)

Reactivate (see page 91)

1.1.9.4.7 TServerClientThread.HandleException Method

Displays an error message using the main VCL thread.

Pascal

```
procedure HandleException; virtual;
```

C++

```
virtual __fastcall HandleException();
```

Description

HandleException obtains the current exception object and calls TApplication.ShowException (see page 4284) (for descendants of Exception (see page 4015)) or the global ShowException (see page 4284) procedure (for other objects) to display an error message. HandleException wraps the call to ShowException (see page 4284) in the Synchronize method to ensure that the call is thread-safe.

See Also

ShowException

ShowException (see page 4284)

Synchronize

1.1.9.4.8 TServerClientThread.ReActivate Method

Pascal

```
procedure ReActivate(ASocket: TServerClientWinSocket);
```

C++

```
__fastcall ReActivate(TServerClientWinSocket ASocket);
```

Description

This is ReActivate, a member of class TServerClientThread.

1.1.9.4.9 TServerClientThread.StartConnect Method

Indicates when the thread is ready to execute.

Pascal

```
function StartConnect: Boolean;
```

C++

```
__fastcall Boolean StartConnect();
```

Description

The Execute (see page 90) method calls StartConnect to determine when to call ClientExecute (see page 88). StartConnect waits until the Reactivate method has signaled that the thread is properly initialized, and then returns true if the thread is ready to run. If the thread is terminated for any reason before it is fully initialized, StartConnect returns false.

See Also

Reactivate (see page 91)

ClientExecute (see page 88)

Execute (see page 90)

1.1.9.5 TServerClientThread Properties

The properties of the TServerClientThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 91)	Specifies the TServerClientWinSocket (see page 93) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 92)	Points to application-specific data associated with the thread.
	KeepInCache (see page 92)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 93).
R	ServerSocket (see page 93)	Specifies the TServerWinSocket (see page 109) object that manages all the TServerClientThread (see page 85) objects for its client connections.

1.1.9.5.1 TServerClientThread.ClientSocket Property

Specifies the TServerClientWinSocket (see page 93) object that uses the thread for all reading and writing over the socket

connection.

Pascal

```
property ClientSocket: TServerClientWinSocket;
```

C++

```
__property TServerClientWinSocket ClientSocket;
```

Description

Use ClientSocket to gain direct access to the Windows socket component that uses the thread. If ClientSocket is nil (Delphi) or NULL (C++), the thread is idle and available for reuse by another TServerClientWinSocket (see page 93) object. TServerClientThread (see page 85) uses ClientSocket to read from or write to the socket connection.

See Also

TServerClientWinSocket (see page 93)

1.1.9.5.2 TServerClientThread.Data Property

Points to application-specific data associated with the thread.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information with the thread object. Data can store information in a thread-safe manner that would otherwise require the use of global variables.

See Also

Data (see page 73)

1.1.9.5.3 TServerClientThread.KeepInCache Property

Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 93).

Pascal

```
property KeepInCache: Boolean;
```

C++

```
__property Boolean KeepInCache;
```

Description

The TServerWinSocket (see page 109) that spawns server client threads for its connections to client sockets maintains a cache of server client threads for reuse. As server client threads are activated, the TServerWinSocket (see page 109) object sets the KeepInCache property to indicate whether the thread will fit in the thread cache.

When KeepInCache is true, FreeOnTerminate is set to false before the thread terminates, and the thread is left in the thread cache. When KeepInCache is false, the thread is freed when execution terminates.

See Also

FreeOnTerminate

ServerSocket (see page 93)

ThreadCacheSize (see page 121)

1.1.9.5.4 TServerClientThread.ServerSocket Property

Specifies the TServerWinSocket (see page 109) object that manages all the TServerClientThread (see page 85) objects for its client connections.

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening Windows socket object that spawned the thread for a client connection it accepted. TServerClientThread (see page 85) passes events and errors to the appropriate event handlers assigned to ServerSocket. TServerClientThread (see page 85) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

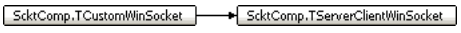
Create (see page 86)

TServerWinSocket (see page 109)

1.1.10 TServerClientWinSocket Class

TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.

Class Hierarchy



Pascal

```
TServerClientWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerClientWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

TServerClientWinSocket is used by server socket components to manage the Windows socket API calls for the socket connection to an individual client socket. TServerClientWinSocket objects are created when a client socket connection is accepted by a listening server socket.

Applications can create and customize the properties of a TServerClientWinSocket object in an OnGetSocket event handler for a server socket component. Component writers can create descendants of TServerClientWinSocket that may be substituted for TServerClientWinSocket in the OnGetSocket event handler.

See Also

TClientWinSocket (see page 25)

TCustomServerSocket (see page 29)

TServerSocket (see page 97)

TServerWinSocket (see page 109)

1.1.10.1 TServerClientWinSocket Members

The following tables list the members exposed by TServerClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 58)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 60)	Opens a listening connection.
	DoOpen (see page 61)	Opens a connection to a remote socket.
	Error (see page 61)	Generates an OnErrorEvent (see page 79) event.
	Event (see page 62)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 62)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 70)	Sets the AsyncStyles property.

Public Events

	Name	Description
	OnErrorEvent (see page 79)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 79)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
	Accept (see page 58)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 59)	Shuts down the socket connection if it is open.
	Connect (see page 59)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 95)	Creates and initializes an instance of TServerClientWinSocket (see page 93).
	DefaultHandler (see page 60)	Processes message records.
	Destroy (see page 96)	Destroys an instance of TServerClientWinSocket (see page 93).
	Disconnect (see page 60)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 63)	Opens a listening connection for a server socket.
	Lock (see page 64)	Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.
	LookupName (see page 64)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 65)	Returns the port number associated with a specific service.
	Open (see page 65)	Opens a connection to a remote socket.
	Read (see page 66)	Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 66)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 67)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 67)	Reads a string from the socket connection.
	SendBuf (see page 68)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 68)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 69)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 69)	Writes the string S to the socket connection.
	Unlock (see page 70)	Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.
	Write (see page 70)	Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
	Addr (see page 72)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 72)	Determines which asynchronous events the socket can receive.
	Connected (see page 73)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 73)	Points to application-specific data associated with the socket.
	Handle (see page 73)	Specifies the handle of the window that receives notifications from the Windows socket.
	LocalAddress (see page 74)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 74)	Specifies the name of the local system.
	LocalPort (see page 75)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 75)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 76)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 76)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 77)	Specifies the name of the remote system.
	RemotePort (see page 77)	Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.
	ServerWinSocket (see page 96)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 93).
	SocketHandle (see page 78)	Specifies the Windows handle for the socket.

1.1.10.2 TServerClientWinSocket.Create Constructor

Creates and initializes an instance of TServerClientWinSocket (see page 93).

Pascal

```
constructor Create(Socket: TSocket; ServerWinSocket: TServerWinSocket);
```

C++

```
__fastcall TServerClientWinSocket(TSocket Socket, TServerWinSocket ServerWinSocket);
```

Description

Use Create to create an instance of TServerClientWinSocket (see page 93) in the OnGetSocket event handler of a server socket component. The Socket parameter is the Windows socket handle for the accepted socket connection to a client socket. The ServerWinSocket (see page 96) parameter is the Windows socket object for the listening connection that accepted the connection to the client socket.

Create adds the new TServerClientWinSocket (see page 93) object to the list of connections maintained by the listening server socket. It then sets the OnSocketEvent (see page 79) event handler so that notifications from the connection to the client can be passed on to the server socket. After setting up the connection to the server socket, Create calls the inherited constructor.

See Also

OnSocketEvent (see page 79)

Create (see page 56)

Connections (see page 119)

1.1.10.3 TServerClientWinSocket.Destroy Destructor

Destroys an instance of TServerClientWinSocket (see page 93).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TServerClientWinSocket (see page 93) reference is not nil, and only then calls Destroy. Instances of TServerClientWinSocket (see page 93) are freed automatically when the connection is closed.

Destroy removes the TServerClientWinSocket (see page 93) from the list of connections maintained by the listening windows socket.

See Also

System::TObject::Free (see page 3382)

Connections (see page 119)

1.1.10.4 TServerClientWinSocket Properties

The properties of the TServerClientWinSocket class are listed here.

Public Properties

	Name	Description
	ServerWinSocket (see page 96)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 93).

1.1.10.4.1 TServerClientWinSocket.ServerWinSocket Property

Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 93).

Pascal

```
property ServerWinSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerWinSocket;
```

Description

Use ServerWinSocket to gain access to the Windows socket object for the listening connection that spawned this TServerClientWinSocket (see page 93). Use the ServerWinSocket to trigger events that can be handled by the event handlers of the server socket, to get execution threads from the cache maintained by the Windows server socket, or to get information about the other active connections accepted by the server socket.

See Also

TServerWinSocket (see page 109)

1.1.11 TServerSocket Class

TServerSocket manages server socket connections for a TCP/IP server.

Class Hierarchy

Pascal

```
TServerSocket = class(TCustomServerSocket);
```

C++

```
class TServerSocket : public TCustomServerSocket;
```

File

ScktComp

Description

Add a TServerSocket object to a form or data module to turn an application into a TCP/IP server. TServerSocket listens for requests for TCP/IP connections from other machines, and establishes connections when requests are received.

See Also

TClientSocket (see page 14)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.11.1 TServerSocket Members

The following tables list the members exposed by TServerSocket.

Protected Events

	Name	Description
	OnConnect (see page 50)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 50)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 51)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 51)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 52)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 53)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 53)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FServerSocket (see page 32)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
A	DoActivate (see page 6)	Provides the prototype of a method to open or close the socket connection.
V	DoActivate (see page 32)	Opens or Closes the socket connection.
A	Error (see page 6)	Provides the prototype of a method that responds to error conditions.

	Error (see page 48)	Generates an OnError (see page 51) event.
	Event (see page 7)	Provides the prototype of a method that responds when specific socket events occur.
	Event (see page 48)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetGetSocketEvent (see page 33)	Returns the OnGetSocket (see page 44) event handler.
	GetGetThreadEvent (see page 33)	Returns the OnGetThread (see page 44) event handler.
	GetOnClientError (see page 34)	Returns the OnClientError (see page 42) event handler.
	GetOnClientEvent (see page 34)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 34)	Returns the OnThreadEnd (see page 45) event handler.
	GetOnThreadStart (see page 35)	Returns the OnThreadStart (see page 45) event handler.
	GetServerType (see page 35)	Returns the ServerType (see page 39) property.
	GetThreadCacheSize (see page 35)	Returns the value of the ThreadCacheSize (see page 40) property.
	InitSocket (see page 8)	Tells a Windows socket object to call the Error (see page 6) and Event (see page 7) methods.
	Loaded (see page 8)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 11) property.
	SetActive (see page 9)	Sets the Active (see page 11) property.
	SetAddress (see page 10)	Sets the Address (see page 12) property.
	SetGetSocketEvent (see page 36)	Sets the OnGetSocket (see page 44) event handler.
	SetGetThreadEvent (see page 36)	Sets the OnGetThread (see page 44) event handler.
	SetHost (see page 10)	Sets the Host (see page 12) property.
	SetOnClientError (see page 37)	Sets the OnClientError (see page 42) event handler.
	SetOnClientEvent (see page 37)	Sets the client event handlers of the associated TServerWinSocket (see page 109) object.
	SetOnThreadEnd (see page 37)	Sets the OnThreadEnd (see page 45) event handler.
	SetOnThreadStart (see page 38)	Sets the OnThreadStart (see page 45) event handler.
	SetPort (see page 10)	Sets the Port (see page 13) property.
	SetServerType (see page 38)	Sets the ServerType (see page 39) property.
	SetService (see page 11)	Sets the Service (see page 13) property.
	SetThreadCacheSize (see page 39)	Sets the ThreadCacheSize (see page 40) property.

Protected Properties

	Name	Description
	Address (see page 12)	Specifies the IP address of the server system.
	Host (see page 12)	Specifies an alias for the IP address of the server system.

Public Methods

	Name	Description
	Close (see page 6)	Shuts down the socket connection.
	Create (see page 99)	Creates an instance of TServerSocket (see page 97).
	Destroy (see page 31)	Destroys an instance of TCustomServerSocket (see page 29).
	Open (see page 9)	Opens the socket connection.

Public Properties

	Name	Description
	Socket (see page 102)	Specifies the TServerWinSocket (see page 109) object that describes the endpoint of the listening connection.

Published Events

	Name	Description
	OnAccept (see page 103)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 104)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 105)	Occurs when one of the connections to a client socket is closed.

	OnClientError (see page 105)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 106)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 106)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 107)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 107)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 108)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 108)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 109)	Occurs when the execution thread for a connection to a client socket starts up.

Published Properties

	Name	Description
	Active (see page 100)	Indicates whether the socket connection is open and available for communication with other machines.
	Port (see page 100)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 101)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 101)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 102)	Specifies the maximum number of threads that can be reused for new client connections.

1.1.11.2 TServerSocket.Create Constructor

Creates an instance of TServerSocket (see page 97).

Pascal

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

C++

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

Description

Call Create to create and initialize a server socket component at runtime. TServerSocket (see page 97) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TServerWinSocket (see page 109) object that encapsulates the Windows socket connection for the server socket.

Initializes the event and error dispatchers for the server socket.

Initializes ThreadCacheSize (see page 102) to 10.

See Also

Create

ThreadCacheSize (see page 40)

1.1.11.3 TServerSocket Properties

The properties of the TServerSocket class are listed here.

Public Properties

	Name	Description
	Socket (see page 102)	Specifies the TServerWinSocket (see page 109) object that describes the endpoint of the listening connection.

Published Properties

	Name	Description
	Active (see page 100)	Indicates whether the socket connection is open and available for communication with other machines.
	Port (see page 100)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 101)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 101)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 102)	Specifies the maximum number of threads that can be reused for new client connections.

1.1.11.3.1 TServerSocket.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

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

See Also

Close (see page 6)

Open (see page 9)

SetActive (see page 9)

1.1.11.3.2 TServerSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

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

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets

generally set Port to a predefined value which clients can use to initiate connections.

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

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

See Also

Address (see page 12)

ESocketError

Host (see page 12)

Service (see page 13)

SetPort (see page 10)

1.1.11.3.3 TServerSocket.ServerType Property

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

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

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

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

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

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

See Also

OnClientRead (see page 43)

OnClientWrite (see page 43)

TServerClientThread (see page 85)

1.1.11.3.4 TServerSocket.Service Property

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

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

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

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

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

See Also

ESocketError

Port (see page 13)

SetService (see page 11)

LookupService (see page 65)

1.1.11.3.5 TServerSocket.Socket Property

Specifies the TServerWinSocket (see page 109) object that describes the endpoint of the listening connection.

Pascal

```
property Socket: TServerWinSocket;
```

C++

```
__property TServerWinSocket Socket;
```

Description

Use Socket to obtain

Information about the connections currently active on the server socket.

Information about the connection threads that are cached for use by the server socket.

Access to the Windows socket handle for the listening connection that is needed for Windows socket API calls.

See Also

ThreadCacheSize (see page 40)

1.1.11.3.6 TServerSocket.ThreadCacheSize Property

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

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution

thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

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

See Also

ServerType (see page 39)

TServerClientThread (see page 85)

ActiveThreads (see page 119)

IdleThreads (see page 120)

1.1.11.4 TServerSocket Events

The events of the TServerSocket class are listed here.

Published Events

	Name	Description
	OnAccept (see page 103)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 104)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 105)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 105)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 106)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 106)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 107)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 107)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 108)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 108)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 109)	Occurs when the execution thread for a connection to a client socket starts up.

1.1.11.4.1 TServerSocket.OnAccept Event

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

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

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

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

See Also

OnConnect (see page 50)

OnListen (see page 51)

1.1.11.4.2 TServerSocket.OnClientConnect Event

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

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

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

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

1An OnListen event occurs just before the server socket is opened for listening.

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

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

4An OnAccept event occurs, using the new TServerClientWinSocket object.

5If ServerType is stThreadBlocking and no thread is available in the cache, an OnGetThread event occurs. If the OnGetThread event handler does not create a thread, the server socket creates a TServerClientThread.

6If ServerType is stThreadBlocking, an OnThreadStart event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket object and an OnClientConnect event occurs.

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

Note: The OnClientConnect event handler for TServerSocket is also set when setting the OnClientConnect event handler of the associated TServerWinSocket.

See Also

OnAccept (see page 49)

OnGetSocket (see page 44)

OnGetThread (see page 44)

OnListen (see page 51)

OnThreadStart (see page 45)

ServerType (see page 39)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

GetClientThread (see page 116)

1.1.11.4.3 TServerSocket.OnClientDisconnect Event

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

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

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

The TServerClientWinSocket that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType is stThreadBlocking, OnThreadEnd occurs after OnClientDisconnect.

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

Note: The OnClientDisconnect event handler for TServerSocket is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket.

See Also

OnThreadEnd (see page 45)

ServerType (see page 39)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

GetClientThread (see page 116)

1.1.11.4.4 TServerSocket.OnClientError Event

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

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode

parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError from being raised.

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

Note: The OnClientError event handler for TServerSocket is also set when setting the OnClientError event handler of the associated TServerWinSocket.

See Also

ESocketError

ServerType (see page 39)

TServerClientWinSocket (see page 93)

GetClientThread (see page 116)

1.1.11.4.5 TServerSocket.OnClientRead Event

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

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write an OnClientRead event handler to write to the socket connection. If the ServerType property is stThreadBlocking, use a TWinSocketStream object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

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

See Also

ServerType (see page 39)

receivebuf (see page 66)

ReceiveText (see page 67)

TServerClientWinSocket (see page 93)

GetClientThread (see page 116)

TWinSocketStream (see page 127)

1.1.11.4.6 TServerSocket.OnClientWrite Event

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

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

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

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

See Also

ServerType (see page 39)

SendBuf (see page 68)

sendstream (see page 68)

SendText (see page 69)

GetClientThread (see page 116)

TWinSocketStream (see page 127)

1.1.11.4.7 TServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

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

See Also

TServerClientWinSocket (see page 93)

Create (see page 95)

TServerWinSocket (see page 109)

1.1.11.4.8 TServerSocket.OnGetThread Event

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

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

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

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

The Sender parameter is the TServerWinSocket object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 39)

TServerClientThread (see page 85)

TServerClientWinSocket (see page 93)

TServerWinSocket (see page 109)

1.1.11.4.9 TServerSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

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

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

See Also

Accept (see page 58)

Address (see page 12)

Port (see page 13)

1.1.11.4.10 TServerSocket.OnThreadEnd Event

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

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect event.

The Sender parameter is the TServerWinSocket that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 42)

TServerWinSocket (see page 109)

1.1.11.4.11 TServerSocket.OnThreadStart Event

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

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

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

See Also

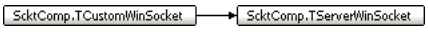
TServerClientThread (see page 85)

TServerWinSocket (see page 109)

1.1.12 TServerWinSocket Class

TServerWinSocket describes the endpoint of a listening server socket connection.

Class Hierarchy



Pascal

```
TServerWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

TServerWinSocket is used by server socket components to manage the Windows socket API calls for a TCP/IP listening connection. TServerWinSocket introduces new properties, events, and methods to manage multiple socket connections to different client sockets, and to generate or reuse TServerClientThread (see page 85) objects that allow each client connection to have its own execution thread.

See Also

TClientWinSocket (see page 25)

TServerClientThread (see page 85)

TServerClientWinSocket (see page 93)

TServerSocket (see page 97)

1.1.12.1 TServerWinSocket Members

The following tables list the members exposed by TServerWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 58)	Looks up information or forms socket connections asynchronously.
	ClientConnect (see page 114)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 114)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 114)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 114)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 115)	This is ClientWrite, a member of class TServerWinSocket.
	DoCreateThread (see page 115)	This is DoCreateThread, a member of class TServerWinSocket.
	DoListen (see page 60)	Opens a listening connection.
	DoOpen (see page 61)	Opens a connection to a remote socket.
	Error (see page 61)	Generates an OnErrorEvent (see page 79) event.
	Event (see page 62)	This is Event, a member of class TCustomWinSocket.
	GetClientSocket (see page 116)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 116)	This is GetServerThread, a member of class TServerWinSocket.
	InitSocket (see page 62)	Generates a windows Internet socket address structure from a description of the desired socket port.
	Listen (see page 116)	Opens a listening connection for a server socket.
	SetAsyncStyles (see page 70)	Sets the AsyncStyles property.
	SetServerType (see page 117)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 117)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 118)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 118)	This is ThreadStart, a member of class TServerWinSocket.

Public Events

	Name	Description
	OnClientConnect (see page 122)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 123)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 124)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 124)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 125)	Occurs when the server socket should write information to a client socket.
	OnErrorEvent (see page 79)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 126)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to form the connection to a client socket.

	OnGetThread (see page 126)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnSocketEvent (see page 79)	Occurs when the Windows socket receives notification of asynchronous events.
	OnThreadEnd (see page 127)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 127)	Occurs when the execution thread for a connection to a client socket starts up.

Public Methods

	Name	Description
V	Accept (see page 113)	Accepts a connection to a client socket.
	Close (see page 59)	Shuts down the socket connection if it is open.
V	Connect (see page 59)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 112)	Creates an instance of TServerWinSocket (see page 109) from a Windows socket handle.
V	DefaultHandler (see page 60)	Processes message records.
V	Destroy (see page 112)	Destroys an instance of TServerWinSocket (see page 109).
V	Disconnect (see page 115)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 116)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.
	Lock (see page 64)	Starts a critical section, blocking all other execution threads until the Unlock (see page 70) method is called.
	LookupName (see page 64)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 65)	Returns the port number associated with a specific service.
	Open (see page 65)	Opens a connection to a remote socket.
V	Read (see page 66)	Generates an OnSocketEvent (see page 79) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 66)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 67)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 67)	Reads a string from the socket connection.
	SendBuf (see page 68)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 68)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 69)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 69)	Writes the string S to the socket connection.
	Unlock (see page 70)	Reenables execution in other threads after a critical section begun by the Lock (see page 64) method.
V	Write (see page 70)	Generates an OnSocketEvent (see page 79) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	ActiveConnections (see page 118)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 119)	Indicates the number of TServerClientThread (see page 85) objects currently in use by the socket connections listed in the Connections (see page 119) array.
R	Addr (see page 72)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 72)	Determines which asynchronous events the socket can receive.
R	Connected (see page 73)	Indicates whether the socket connection is open and available for communication with other machines.
R	Connections (see page 119)	Lists all TServerClientWinSocket (see page 93) objects that represent open socket connections to client sockets that were accepted by this listening socket.
	Data (see page 73)	Points to application-specific data associated with the socket.
R	Handle (see page 73)	Specifies the handle of the window that receives notifications from the Windows socket.
R	IdleThreads (see page 120)	Indicates the number of unused TServerClientThread (see page 85) objects available in the cache.

	LocalAddress (see page 74)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 74)	Specifies the name of the local system.
	LocalPort (see page 75)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 75)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 76)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 76)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 77)	Specifies the name of the remote system.
	RemotePort (see page 77)	Specifies the ID number of the port used by the RemoteHost (see page 77) to form the socket connection.
	ServerType (see page 120)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	SocketHandle (see page 78)	Specifies the Windows handle for the socket.
	ThreadCacheSize (see page 121)	Indicates the number of TServerClientThread (see page 85) objects cached for reuse by new connections to client sockets.

1.1.12.2 TServerWinSocket.Create Constructor

Creates an instance of TServerWinSocket (see page 109) from a Windows socket handle.

Pascal

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

C++

```
__fastcall TServerWinSocket(TSocket ASocket);
```

Description

Call Create to create a Windows listening socket object. Windows listening socket objects are created automatically by the server socket component that uses them.

Create allocates the helper objects that manage the list of active client connections and the thread cache before calling the inherited constructor. Create also initializes the ASyncStyles (see page 72) property to [asAccept].

See Also

ASyncStyles (see page 72)

Connections (see page 119)

Create (see page 56)

1.1.12.3 TServerWinSocket.Destroy Destructor

Destroys an instance of TServerWinSocket (see page 109).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the Windows listening socket object is not nil, and only then calls Destroy. Unless the Windows socket object is not used by a server socket component, it is freed automatically by the socket component.

Destroy frees the helper objects that manage the list of active client connections and the thread cache.

See Also

System::TObject::Free (see page 3382)

TServerSocket (see page 97)

1.1.12.4 TServerWinSocket Methods

The methods of the TServerWinSocket class are listed here.

Protected Methods

	Name	Description
	ClientConnect (see page 114)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 114)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 114)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 114)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 115)	This is ClientWrite, a member of class TServerWinSocket.
	DoCreateThread (see page 115)	This is DoCreateThread, a member of class TServerWinSocket.
	GetClientSocket (see page 116)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 116)	This is GetServerThread, a member of class TServerWinSocket.
	Listen (see page 116)	Opens a listening connection for a server socket.
	SetServerType (see page 117)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 117)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 118)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 118)	This is ThreadStart, a member of class TServerWinSocket.

Public Methods

	Name	Description
	Accept (see page 113)	Accepts a connection to a client socket.
	Disconnect (see page 115)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 116)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.

1.1.12.4.1 TServerWinSocket.Accept Method

Accepts a connection to a client socket.

Pascal

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

C++

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

Description

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

Accept accepts the client connection request, obtaining the Windows socket handle for the client connection. It then generates an OnGetSocket (see page 126) event, passing this socket handle as an argument to any event handler. If a TServerClientWinSocket (see page 93) is not created by an OnGetSocket (see page 126) event handler, Accept creates a TServerClientWinSocket (see page 93) to represent the server endpoint of the accepted connection. Next, Accept obtains a running TServerClientThread (see page 85), either by restarting a thread in the cache, obtaining one from an OnGetThread (see page 126) event, or creating a new TServerClientThread (see page 85) object. This new thread handles the connection to

the client socket.

Accept is only called if the ASyncStyles (see page 72) property includes asAccept.

See Also

ASyncStyles (see page 72)

OnGetSocket (see page 126)

OnGetThread (see page 126)

1.1.12.4.2 TServerWinSocket.ClientConnect Method

Pascal

```
procedure ClientConnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientConnect(TCustomWinSOcket Socket);
```

Description

This is ClientConnect, a member of class TServerWinSocket.

1.1.12.4.3 TServerWinSocket.ClientDisconnect Method

Pascal

```
procedure ClientDisconnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientDisconnect(TCustomWinSOcket Socket);
```

Description

This is ClientDisconnect, a member of class TServerWinSocket.

1.1.12.4.4 TServerWinSocket.ClientErrorEvent Method

Pascal

```
procedure ClientErrorEvent(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode:
Integer); dynamic;
```

C++

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

Description

This is ClientErrorEvent, a member of class TServerWinSocket.

1.1.12.4.5 TServerWinSocket.ClientRead Method

Pascal

```
procedure ClientRead(Socket: TCustomWinSocket); dynamic;
```

C++

```
__fastcall ClientRead(TCustomWinSocket Socket);
```

Description

This is ClientRead, a member of class TServerWinSocket.

1.1.12.4.6 TServerWinSocket.ClientWrite Method

Pascal

```
procedure ClientWrite(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientWrite(TCustomWinSOcket Socket);
```

Description

This is ClientWrite, a member of class TServerWinSocket.

1.1.12.4.7 TServerWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

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

C++

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

Description

Disconnect is called by the Close (see page 59) method to terminate the socket connection.

Disconnect calls the Lock (see page 64) method to make sure the method is thread-safe. Then, it terminates all open connections to client sockets. Next it generates an OnSocketEvent (see page 79) of type seDisconnect. Finally, it closes the listening connection and calls Unlock (see page 70) to end the critical section started by Lock (see page 64).

See Also

Close (see page 59)

Lock (see page 64)

OnSocketEvent (see page 79)

Unlock (see page 70)

1.1.12.4.8 TServerWinSocket.DoCreateThread Method

Pascal

```
function DoCreateThread(ClientSocket: TServerClientWinSocket): TServerClientThread; virtual;
```

C++

```
virtual __fastcall TServerClientThread DoCreateThread(TServerClientWinSocket ClientSocket);
```

Description

This is DoCreateThread, a member of class TServerWinSocket.

1.1.12.4.9 TServerWinSocket.GetClientSocket Method

Pascal

```
function GetClientSocket(Socket: TSocket): TServerClientWinSocket; dynamic;
```

C++

```
__fastcall TServerClientWinSocket GetClientSocket(TSocket Socket);
```

Description

This is GetClientSocket, a member of class TServerWinSocket.

1.1.12.4.10 TServerWinSocket.GetClientThread Method

Returns the execution thread used by the client connection managed by the ClientSocket parameter.

Pascal

```
function GetClientThread(ClientSocket: TServerClientWinSocket): TServerClientThread;
```

C++

```
__fastcall TServerClientThread GetClientThread(TServerClientWinSocket ClientSocket);
```

Description

Use GetClientThread to obtain access to the TServerClientThread (see page 85) for a particular client connection. GetClientThread allows an OnClient... event handler to access thread-global information saved in the Data (see page 73) property of the thread.

See Also

OnClientConnect (see page 122)

OnClientDisconnect (see page 123)

OnClientError (see page 124)

OnClientRead (see page 124)

OnClientWrite (see page 125)

Data (see page 92)

1.1.12.4.11 TServerWinSocket.GetServerThread Method

Pascal

```
function GetServerThread(ClientSocket: TServerClientWinSocket): TServerClientThread; dynamic;
```

C++

```
__fastcall TServerClientThread GetServerThread(TServerClientWinSocket ClientSocket);
```

Description

This is GetServerThread, a member of class TServerWinSocket.

1.1.12.4.12 TServerWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(var Name: string; var Address: string; var Service: string; Port: Word;
QueueSize: Integer);
```

C++

```
__fastcall Listen(UnicodeString Name, UnicodeString Address, UnicodeString Service, Word Port,
int QueueSize);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 79) of type seListen. Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 73) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component. The QueueSize is set to 5 by TServerSocket (see page 97).

See Also

Connected (see page 73)

InitSocket (see page 62)

OnSocketEvent (see page 79)

Address (see page 12)

Host (see page 12)

Port (see page 13)

Service (see page 13)

TServerSocket (see page 97)

1.1.12.4.13 TServerWinSocket.SetServerType Method

Pascal

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

C++

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

Description

This is SetServerType, a member of class TServerWinSocket.

1.1.12.4.14 TServerWinSocket.SetThreadCacheSize Method

Pascal

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

C++

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

Description

This is SetThreadCacheSize, a member of class TServerWinSocket.

1.1.12.4.15 TServerWinSocket.ThreadEnd Method

Pascal

```
procedure ThreadEnd(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadEnd(TServerClientThread AThread);
```

Description

This is ThreadEnd, a member of class TServerWinSocket.

1.1.12.4.16 TServerWinSocket.ThreadStart Method

Pascal

```
procedure ThreadStart(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadStart(TServerClientThread AThread);
```

Description

This is ThreadStart, a member of class TServerWinSocket.

1.1.12.5 TServerWinSocket Properties

The properties of the TServerWinSocket class are listed here.

Public Properties

	Name	Description
R	ActiveConnections (see page 118)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 119)	Indicates the number of TServerClientThread (see page 85) objects currently in use by the socket connections listed in the Connections (see page 119) array.
R	Connections (see page 119)	Lists all TServerClientWinSocket (see page 93) objects that represent open socket connections to client sockets that were accepted by this listening socket.
R	IdleThreads (see page 120)	Indicates the number of unused TServerClientThread (see page 85) objects available in the cache.
	ServerType (see page 120)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 121)	Indicates the number of TServerClientThread (see page 85) objects cached for reuse by new connections to client sockets.

1.1.12.5.1 TServerWinSocket.ActiveConnections Property

Indicates the number of connections to client sockets accepted by this listening socket that are currently open.

Pascal

```
property ActiveConnections: Integer;
```

C++

```
__property int ActiveConnections;
```

Description

Use ActiveConnections to monitor the use of the server socket. ActiveConnections is the number of entries in the Connections

(see page 119) array.

See Also

Connections (see page 119)

1.1.12.5.2 TServerWinSocket.ActiveThreads Property

Indicates the number of TServerClientThread (see page 85) objects currently in use by the socket connections listed in the Connections (see page 119) array.

Pascal

```
property ActiveThreads: Integer;
```

C++

```
__property int ActiveThreads;
```

Description

Use ActiveThreads to monitor thread usage by the listening socket. If ActiveThreads is usually much higher than ThreadCacheSize (see page 121), increasing ThreadCacheSize (see page 121) should improve performance. To determine whether ThreadCacheSize (see page 121) is too high, use the IdleThreads (see page 120) property.

When the ServerType (see page 120) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache and counted by ActiveThreads.

See Also

Connections (see page 119)

GetClientThread (see page 116)

IdleThreads (see page 120)

ServerType (see page 120)

ThreadCacheSize (see page 121)

1.1.12.5.3 TServerWinSocket.Connections Property

Lists all TServerClientWinSocket (see page 93) objects that represent open socket connections to client sockets that were accepted by this listening socket.

Pascal

```
property Connections [Index: Integer]: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Connections[int Index];
```

Description

Use Connections with the ActiveConnections (see page 118) property to iterate over all open socket connections accepted by the listening socket. Each entry in the Connections array is a TServerClientWinSocket (see page 93) object, where an Index of 0 gives the first TServerClientWinSocket (see page 93), an Index of 1 returns the second TServerClientWinSocket (see page 93), and so on.

If ServerType (see page 120) is stThreadBlocking, the thread associated with each of the TServerClientWinSocket (see page 93) objects in the Connections array can be obtained using the GetClientThread (see page 116) method.

See Also

ActiveConnections (see page 118)

GetClientThread (see page 116)

ServerType (see page 120)

TServerClientWinSocket (see page 93)

1.1.12.5.4 TServerWinSocket.IdleThreads Property

Indicates the number of unused TServerClientThread (see page 85) objects available in the cache.

Pascal

```
property IdleThreads: Integer;
```

C++

```
__property int IdleThreads;
```

Description

Use IdleThreads to monitor thread cache usage by the listening socket. If IdleThreads often very high, the thread cache is tying up a lot of unused memory storing extra threads. Monitor the ActiveThreads (see page 119) property to determine an appropriate value for ThreadCacheSize (see page 121).

When the ServerType (see page 120) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache. As the threads in the cache complete their execution, they become available for reuse, and are counted by IdleThreads. When there are more active threads than ThreadCacheSize (see page 121), the extra threads are freed when they finish execution instead.

See Also

ActiveThreads (see page 119)

ServerType (see page 120)

ThreadCacheSize (see page 121)

1.1.12.5.5 TServerWinSocket.ServerType Property

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

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the listening socket connection. When ServerType is stThreadBlocking, the TServerClientThread (see page 85) object for a connection generates OnClientRead (see page 124) or OnClientWrite (see page 125) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over socket connections accepted by the listening socket asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default.

OnClientRead (see page 124) or OnClientWrite (see page 125) events occur when the server socket needs to attend to I/O over one of the connections.

Use a non-blocking socket when reading and writing must be synchronized with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient...

event handlers contain thread-safe code.

See Also

ASyncStyles (see page 72)

OnClientRead (see page 124)

OnClientWrite (see page 125)

OnGetThread (see page 126)

TServerClientThread (see page 85)

1.1.12.5.6 TServerWinSocket.ThreadCacheSize Property

Indicates the number of TServerClientThread (see page 85) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 120) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New (see page 3712) connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

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

See Also

ActiveThreads (see page 119)

IdleThreads (see page 120)

ServerType (see page 120)

1.1.12.6 TServerWinSocket Events

The events of the TServerWinSocket class are listed here.

Public Events

	Name	Description
	OnClientConnect (see page 122)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 123)	Occurs when one of the connections to a client socket is closed.

	OnClientError (see page 124)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 124)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 125)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 126)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to form the connection to a client socket.
	OnGetThread (see page 126)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnThreadEnd (see page 127)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 127)	Occurs when the execution thread for a connection to a client socket starts up.

1.1.12.6.1 TServerWinSocket.OnClientConnect Event

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

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write (see page 70) an OnClientConnect event handler to take specific action when a client socket completes the socket connection to a TServerClientWinSocket (see page 93) object. Depending on the service, this is the point when the socket should begin reading or writing.

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

1An OnSocketEvent (see page 79) event of type seListen occurs just before the Windows socket is opened for listening.

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

3The listening socket generates an OnGetSocket (see page 126) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 93) object for the Windows socket handle was not created in the OnGetSocket (see page 126) event handler, the Windows socket creates one. The Windows socket continues to listen for other clients.

4An OnSocketEvent (see page 79) event of type seAccept occurs, using the new TServerClientWinSocket (see page 93) object.

5If ServerType (see page 120) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 126) event occurs. If the OnGetThread (see page 126) event handler does not create a thread, the Windows socket object creates a TServerClientThread (see page 85).

6If ServerType (see page 120) is stThreadBlocking, an OnThreadStart (see page 127) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 93) object and an OnClientConnect event occurs.

The Sender parameter is the TServerWinClient object that listens for client requests. The Socket parameter is the TServerClientWinSocket (see page 93) object that forms a connection to the client socket.

Note: If ServerType (see page 120) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread (see page 116) method to access thread-specific information.

Note: The OnClientConnect event handler for TServerWinSocket (see page 109) is also set when setting the

OnClientConnect event handler of the associated TServerSocket (see page 97).

See Also

GetClientThread (see page 116)

OnGetSocket (see page 126)

OnGetThread (see page 126)

OnSocketEvent (see page 79)

OnThreadStart (see page 127)

ServerType (see page 120)

TServerClientWinSocket (see page 93)

OnClientConnect (see page 41)

1.1.12.6.2 TServerWinSocket.OnClientDisconnect Event

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

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write (see page 70) an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client request does not close the listening socket. The listening connection remains open for accepting new client connection requests.

The TServerClientWinSocket (see page 93) object associated with the client connection is freed after OnClientDisconnect. If ServerType (see page 120) is stThreadBlocking, OnThreadEnd (see page 127) occurs after OnClientDisconnect.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 93) object that is ending a connection with the client socket.

Note: If ServerType (see page 120) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe.Use the GetClientThread (see page 116) method to access thread-specific information.

The OnClientDisconnect event handler for TServerWinSocket (see page 109) is also set when setting the OnClientDisconnect event handler of the associated TServerSocket (see page 97).

See Also

GetClientThread (see page 116)

OnThreadEnd (see page 127)

ServerType (see page 120)

TServerClientWinSocket (see page 93)

OnClientDisconnect (see page 42)

1.1.12.6.3 TServerWinSocket.OnClientError Event

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

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write (see page 70) an OnClientError event handler to respond to errors that arise with the connection to a client socket. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition, to prevent an ESocketError (see page 3) from being raised.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 93) object that manages the connection to the client. The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. The ErrorCode parameter is the error code returned by the Windows socket API call.

Note: If ServerType (see page 120) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread (see page 116) method to access thread-specific information.

Note: The OnClientError event handler for TServerWinSocket (see page 109) is also set when setting the OnClientError event handler of the associated TServerSocket (see page 97).

See Also

ESocketError

GetClientThread (see page 116)

ServerType (see page 120)

TServerClientWinSocket (see page 93)

OnClientError (see page 42)

1.1.12.6.4 TServerWinSocket.OnClientRead Event

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

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write (see page 70) an OnClientRead event handler to read from the socket connection. Use a TWinSocketStream (see page 127) object or the methods of the Socket parameter to do the reading.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 93) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 120) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread (see page 116) method to access thread-specific information.

Note: The OnClientRead event handler for TServerWinSocket (see page 109) is also set when setting the OnClientRead event handler of the associated TServerSocket (see page 97).

See Also

GetClientThread (see page 116)

ServerType (see page 120)

receivebuf (see page 66)

ReceiveText (see page 67)

TServerClientWinSocket (see page 93)

OnClientRead (see page 43)

TWinSocketStream (see page 127)

1.1.12.6.5 TServerWinSocket.OnClientWrite Event

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

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write (see page 70) an OnClientWrite event handler to write to the socket connection. Use a TWinSocketStream (see page 127) object or the methods of the Socket parameter to do the actual writing.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 93) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 120) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread (see page 116) method to access thread-specific information.

Note: The OnClientWrite event handler for TServerWinSocket (see page 109) is also set when setting the OnClientWrite event handler of the associated TServerSocket (see page 97).

See Also

GetClientThread (see page 116)

ServerType (see page 120)

SendBuf (see page 68)

sendstream (see page 68)

SendText (see page 69)

OnClientWrite (see page 43)

1.1.12.6.6 TServerWinSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 93) object to form the connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write (see page 70) an OnGetSocket event handler to create a specialized TServerClientWinSocket (see page 93) object to use in the connection represented by the Socket parameter. Return the new TServerClientWinSocket (see page 93) object in the ClientSocket parameter.

The Sender parameter is the TServerWinSocket (see page 109) object that received the client request. The Socket parameter is the Windows socket handle for the socket connection to the client socket.

Note: The OnGetSocket event handler for TServerWinSocket (see page 109) is also set when setting the OnGetSocket event handler of the associated TServerSocket (see page 97).

See Also

Create (see page 95)

OnGetSocket (see page 44)

1.1.12.6.7 TServerWinSocket.OnGetThread Event

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

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write (see page 70) an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 85) for the connection to the client socket. Create (see page 112) the new thread with the CreateSuspended parameter set to false, and return it in the SocketThread parameter. OnGetThread only occurs if there are no idle threads in the cache.

Most applications that use thread-blocking connections will want to create a new descendant of TServerClientThread (see page 85) in an OnGetThread event handler. This is because the default behavior of TServerClientThread (see page 85) uses the OnClientRead (see page 124) and OnClientWrite (see page 125) event handlers for reading and writing. These events occur on the server socket, which is not thread-local.

The Sender parameter is the TServerWinSocket (see page 109) object that received the client request. The ClientSocket parameter is the TServerClientWinSocket (see page 93) object that will communicate with the client socket.

Note: The OnGetThread event handler for TServerWinSocket (see page 109) is also set when setting the OnGetThread event handler of the associated TServerSocket (see page 97).

See Also

IdleThreads (see page 120)

ServerType (see page 120)

OnGetThread (see page 44)

1.1.12.6.8 TServerWinSocket.OnThreadEnd Event

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

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write (see page 70) an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 123) event.

The Sender parameter is the TServerWinSocket (see page 109) that received the client request. The Thread parameter is the thread that is finishing up.

Note: The OnThreadEnd event handler for TServerWinSocket (see page 109) is also set when setting the OnThreadEnd event handler of the associated TServerSocket (see page 97).

See Also

OnClientDisconnect (see page 123)

OnThreadEnd (see page 45)

1.1.12.6.9 TServerWinSocket.OnThreadStart Event

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

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write (see page 70) an OnThreadStart to initialize the TServerClientThread (see page 85) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 109) that received the client request. The Thread parameter is the thread that is about to start execution.

Note: The OnThreadStart event handler for TServerWinSocket (see page 109) is also set when setting the OnThreadStart event handler of the associated TServerSocket (see page 97).

See Also

OnThreadStart (see page 45)

1.1.13 TWinSocketStream Class

TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Class Hierarchy

Pascal

```
TWinSocketStream = class(TStream);
```

C++

```
class TWinSocketStream : public TStream;
```

File

ScktComp

Description

Use TWinSocketStream to read or write information over a blocking socket connection. Windows socket objects include methods to read from or write to the socket connection they represent. However, these methods do not provide a mechanism for timing out when the socket connection is dropped or for waiting until the socket connection is ready before reading.

When the socket is a non-blocking socket, this lack of a time-out or waiting mechanism is not a problem, because reading and writing occur asynchronously in response to notifications from the socket connection. For blocking sockets, however, these mechanisms provided by TWinSocketStream are necessary so that the application using the socket does not hang indefinitely.

To use a Windows socket stream, create an instance of TWinSocketStream, use the methods of the stream to read or write the data, and then free the Windows socket stream.

Note: TWinSocketStream does not work with non-blocking sockets.

See Also

TClientSocket (see page 14)

TClientWinSocket (see page 25)

TServerClientWinSocket (see page 93)

TServerSocket (see page 97)

1.1.13.1 TWinSocketStream Members

The following tables list the members exposed by TWinSocketStream.

Public Methods

	Name	Description
	Create (see page 129)	Creates an instance of TWinSocketStream (see page 127) for a Windows socket object.
V	Destroy (see page 129)	Destroys an instance of TWinSocketStream (see page 127).
V	Read (see page 130)	Reads up to Count bytes from the socket connection into Buffer.
V	Seek (see page 130)	Always returns 0.
	WaitForData (see page 131)	Waits up to TimeOut (see page 132) milliseconds for the socket connection to be ready to transfer data.
V	Write (see page 131)	Writes Count bytes from Buffer to the socket connection.

Public Properties

	Name	Description
	TimeOut (see page 132)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

1.1.13.2 TWinSocketStream.Create Constructor

Creates an instance of TWinSocketStream (see page 127) for a Windows socket object.

Pascal

```
constructor Create(ASocket: TCustomWinSocket; TimeOut: Longint);
```

C++

```
__fastcall TWinSocketStream(TCustomWinSocket ASocket, Longint TimeOut);
```

Description

Call Create to obtain an instance of TWinSocketStream (see page 127) for reading from or writing to the socket connection represented by the ASocket parameter. ASocket should be the endpoint of an open blocking socket connection to another Windows socket.

Create associates the stream with the socket connection, and allocates helper objects to allow it to determine when the socket connection is ready for reading or writing. The TimeOut (see page 132) property (the number of milliseconds before a read or write operation aborts) is initialized to the value of the TimeOut (see page 132) parameter.

If ASocket represents a non-blocking socket connection, Create raises an ESocketError (see page 3) exception.

See Also

ESocketError

ClientType (see page 28)

TCustomWinSocket (see page 54)

ServerType (see page 120)

TimeOut (see page 132)

1.1.13.3 TWinSocketStream.Destroy Destructor

Destroys an instance of TWinSocketStream (see page 127).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWinSocketStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TWinSocketStream (see page 127) reference is not nil and only then calls Destroy.

Free the TWinSocketStream (see page 127) object when it is no longer needed for reading from or writing to the socket connection.

Destroy frees the helper objects allocated in the Create (see page 129) method before calling the default destructor.

See Also

Create (see page 129)

System::TObject::Free (see page 3382)

1.1.13.4 TWinSocketStream Methods

The methods of the TWinSocketStream class are listed here.

Public Methods

	Name	Description
	Read (see page 130)	Reads up to Count bytes from the socket connection into Buffer.
	Seek (see page 130)	Always returns 0.
	WaitForData (see page 131)	Waits up to TimeOut (see page 132) milliseconds for the socket connection to be ready to transfer data.
	Write (see page 131)	Writes Count bytes from Buffer to the socket connection.

1.1.13.4.1 TWinSocketStream.Read Method

Reads up to Count bytes from the socket connection into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read data from the socket connection when the number of bytes to be transferred is unknown. Buffer must have at least Count bytes allocated to hold the data that is read from the connection. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read may return 0 if the socket connection is extremely slow and the read operation has not completed after TimeOut (see page 132) milliseconds. This ensures that the Read method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Read method timing out because of a slow connection, set Count fairly low, and make several calls to Read, rather than fewer calls with a large value of Count.

To ensure that the socket connection is ready to send data before calling Read, use the WaitForData (see page 131) method.

Unlike the ReadBuffer method, Read does not raise an exception if Count bytes are not read from the socket connection.

See Also

ReadBuffer

WaitForData (see page 131)

Write (see page 131)

TimeOut (see page 132)

1.1.13.4.2 TWinSocketStream.Seek Method

Always returns 0.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Seek overrides the inherited method to always set the Position property of the TWinSocketStream (see page 127) object to 0. Socket connections do not support random access. The Position property of a Windows socket stream can only reflect the position at the beginning of all remaining data.

See Also

Position

1.1.13.4.3 TWinSocketStream.WaitForData Method

Waits up to TimeOut (see page 132) milliseconds for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(Timeout: Longint): Boolean;
```

C++

```
__fastcall Boolean WaitForData(Longint Timeout);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns true if the socket connection is ready. WaitForData returns false if the socket connection is not ready after TimeOut (see page 132) milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, Read (see page 130) or Write (see page 131) method calls may time out before any data is transferred.

See Also

Read (see page 130)

Write (see page 131)

1.1.13.4.4 TWinSocketStream.Write Method

Writes Count bytes from Buffer to the socket connection.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write data from a Buffer to the socket connection. Write returns the number of bytes transferred.

Write may return 0 if the socket connection is extremely slow and the write operation has not completed after TimeOut (see page 132) milliseconds. This ensures that the Write method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Write method timing out because of a slow connection, set Count fairly low, and make several calls to Write, rather than fewer calls with a large value of Count.

Unlike the WriteBuffer method, Write does not raise an exception if Count bytes are not written. For example, if the connection is

too slow to transfer all the data, Write will return 0, where WriteBuffer will raise an exception.

See Also

Read (see page 130)

WaitForData (see page 131)

WriteBuffer

TimeOut (see page 132)

1.1.13.5 TWinSocketStream Properties

The properties of the TWinSocketStream class are listed here.

Public Properties

	Name	Description
	TimeOut (see page 132)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

1.1.13.5.1 TWinSocketStream.TimeOut Property

Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

Pascal

```
property TimeOut: Longint;
```

C++

```
__property Longint TimeOut;
```

Description

Set TimeOut to the amount of time that should elapse before a read or write operation aborts under the assumption that the socket connection has failed.

See Also

Read (see page 130)

Write (see page 131)

1.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SetErrorProc (see page 133)	Replaces the exception handler for error messages that are received from a Windows socket connection.

1.2.1 SckttComp.SetErrorProc Function

Replaces the exception handler for error messages that are received from a Windows socket connection.

Pascal

```
function SetErrorProc(ErrorProc: TSocketErrorProc): TSocketErrorProc;
```

C++

```
TSocketErrorProc SetErrorProc(TSocketErrorProc ErrorProc);
```

File

SckttComp

Description

Call SetErrorProc to assign an exception handler for error messages from Windows socket API calls. By default, socket components have no special error handler and simply raise an ESocketError (see page 3) exception when they receive error messages. If an exception handler is assigned using SetErrorProc, the socket component does not raise this exception.

The ErrorProc (see page 3886) parameter specifies the exception handler. It is passed the Windows socket error code. Inside this exception handler an application can compensate for the error or raise an exception. Passing a value of nil (Delphi) or NULL (C++) for the ErrorProc (see page 3886) parameter restores the default behavior, where the socket component raises an ESocketError (see page 3) exception.

SetErrorProc returns the current exception handler. This value is nil (Delphi) or NULL (C++) when SetErrorProc is called for the first time, and is the value of the ErrorProc (see page 3886) parameter from the previous invocation of SetErrorProc thereafter.

See Also

ESocketError (see page 3)

1.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAsyncStyle (see page 134)	This is record ScktComp.TAsyncStyle.
	TClientType (see page 135)	This is record ScktComp.TClientType.
	TErrorEvent (see page 136)	This is record ScktComp.TErrorEvent.
	TLookupState (see page 137)	This is record ScktComp.TLookupState.
	TServerType (see page 137)	This is record ScktComp.TServerType.
	TSocketEvent (see page 137)	TSocketEvent identifies the operation that a socket is performing.

Records

	Name	Description
	TCMLookupComplete (see page 135)	This is record ScktComp.TCMLookupComplete.
	TCMSocketMessage (see page 136)	This is record ScktComp.TCMSocketMessage.

1.3.1 ScktComp.TAsyncStyle Enumeration

Pascal

```
TAsyncStyle = (
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
);
```

C++

```
enum TAsyncStyle {
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
};
```

File

ScktComp

Description

This is record ScktComp.TAsyncStyle.

1.3.2 ScktComp.TClientType Enumeration

Pascal

```
TClientType = (
  ctNonBlocking,
  ctBlocking
);
```

C++

```
enum TClientType {
  ctNonBlocking,
  ctBlocking
};
```

File

ScktComp

Description

This is record ScktComp.TClientType.

1.3.3 ScktComp.TCMLookupComplete Record

Pascal

```
TCMLookupComplete = record
  Msg: Cardinal;
  LookupHandle: THandle;
  AsyncBufLen: Word;
  AsyncError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMLookupComplete {
  unsigned Msg;
  THandle LookupHandle;
  Word AsyncBufLen;
  Word AsyncError;
  Longint Result;
};
```

File

ScktComp

Description

This is record ScktComp.TCMLookupComplete.

1.3.4 ScktComp.TCMSocketMessage Record

Pascal

```
TCMSocketMessage = record
  Msg: Cardinal;
  Socket: TSocket;
  SelectEvent: Word;
  SelectError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMSocketMessage {
  unsigned Msg;
  TSocket Socket;
  Word SelectEvent;
  Word SelectError;
  Longint Result;
};
```

File

ScktComp

Description

This is record ScktComp.TCMSocketMessage.

1.3.5 ScktComp.TErrorEvent Enumeration

Pascal

```
TErrorEvent = (
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
);
```

C++

```
enum TErrorEvent {
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
};
```

File

ScktComp

Description

This is record SckttComp.TErrorEvent.

1.3.6 SckttComp.TLookupState Enumeration

Pascal

```
TLookupState = (
  lsIdle,
  lsLookupAddress,
  lsLookupService
);
```

C++

```
enum TLookupState {
  lsIdle,
  lsLookupAddress,
  lsLookupService
};
```

File

SckttComp

Description

This is record SckttComp.TLookupState.

1.3.7 SckttComp.TServerType Enumeration

Pascal

```
TServerType = (
  stNonBlocking,
  stThreadBlocking
);
```

C++

```
enum TServerType {
  stNonBlocking,
  stThreadBlocking
};
```

File

SckttComp

Description

This is record SckttComp.TServerType.

1.3.8 SckttComp.TSocketEvent Enumeration

TSocketEvent identifies the operation that a socket is performing.

Pascal

```
TSocketEvent = (
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
);
```

C++

```
enum TSocketEvent {
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
};
```

File

ScktComp

Description

TSocketEvent is used to identify what a socket is doing when various methods or event handlers are called. It has one of the following values:

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

1.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAsyncStyles (see page 139)	This is type SocktComp.TAsyncStyles.
TGetSocketEvent (see page 139)	This is type SocktComp.TGetSocketEvent.
TGetThreadEvent (see page 140)	This is type SocktComp.TGetThreadEvent.
TSocketErrorEvent (see page 140)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 141)	This is type SocktComp.TSocketErrorProc.
TSocketEventEvent (see page 141)	This is type SocktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 141)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 142)	This is type SocktComp.TThreadNotifyEvent.

1.4.1 SocktComp.TAsyncStyles Type

Pascal

```
TAsyncStyles = set of TAsyncStyle;
```

C++

```
TAsyncStyle TAsyncStyles;
```

File

SocktComp

Description

This is type SocktComp.TAsyncStyles.

1.4.2 SocktComp.TGetSocketEvent Type

Pascal

```
TGetSocketEvent = procedure (Sender: TObject; Socket: TSocket; var ClientSocket:
TServerClientWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TSocket; var ClientSocket: TServerClientWinSocket) (
TGetSocketEvent)();
```

File

SocktComp

Description

This is type SocktComp.TGetSocketEvent.

1.4.3 ScktComp.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; ClientSocket: TServerClientWinSocket; var
SocketThread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; ClientSocket: TServerClientWinSocket; var SocketThread: TServerClientThread)
( TGetThreadEvent)();
```

File

ScktComp

Description

This is type ScktComp.TGetThreadEvent.

1.4.4 ScktComp.TSocketErrorEvent Type

TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; ErrorEvent:
TErrorEvent; var ErrorCode: Integer) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer) (
TSocketErrorEvent)();
```

File

ScktComp

Description

The Sender parameter is the socket component that encountered the error.

The Socket parameter is the TCustomWinSocket (see page 54) object that receives the error notification.

The ErrorCode parameter is the error code returned by the Windows socket API call. Changing this to 0 within an error handler prevents an exception from being raised.

The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. It is of type TErrorEvent (see page 136), which can have any of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	A connection request that was already accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.

eeAccept	A problem occurred when trying to accept a client connection request.

1.4.5 SocktComp.TSocketErrorProc Type

Pascal

```
TSocketErrorProc = procedure (ErrorCode: Integer);
```

C++

```
(ErrorCode: Integer) ( TSocketErrorProc)();
```

File

SocktComp

Description

This is type SocktComp.TSocketErrorProc.

1.4.6 SocktComp.TSocketEventEvent Type

Pascal

```
TSocketEventEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; SocketEvent:
TSocketEvent) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; SocketEvent: TSocketEvent) ( TSocketEventEvent)();
```

File

SocktComp

Description

This is type SocktComp.TSocketEventEvent.

1.4.7 SocktComp.TSocketNotifyEvent Type

The type for event handlers which respond to events during the expected course of forming or using a socket connection.

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject; Socket: TCustomWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket) ( TSocketNotifyEvent)();
```

File

SocktComp

Description

The Sender parameter is the socket component or Windows socket object to which the event occurred.

The Socket parameter is the TCustomWinSocket (see page 54) object that represents the endpoint of the connection.

When the event handler belongs to a client Windows socket object, these two parameters are the same. For server Windows socket objects, the Sender parameter represents the listening connection while the Socket parameter represents a client connection.

See Also

TCustomSocket (see page 46)

1.4.8 ScktComp.TThreadNotifyEvent Type

Pascal

```
TThreadNotifyEvent = procedure (Sender: TObject; Thread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; Thread: TServerClientThread) ( TThreadNotifyEvent)();
```

File

ScktComp

Description

This is type ScktComp.TThreadNotifyEvent.

1.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_DEFERFREE (see page 143)	This is constant SocktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 143)	This is constant SocktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 143)	This is constant SocktComp.CM_SOCKETMESSAGE.

1.5.1 SocktComp.CM_DEFERFREE Constant

Pascal

```
CM_DEFERFREE = WM_USER + $0002;
```

C++

```
CM_DEFERFREE = WM_USER + $0002;
```

File

SocktComp

Description

This is constant SocktComp.CM_DEFERFREE.

1.5.2 SocktComp.CM_LOOKUPCOMPLETE Constant

Pascal

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

C++

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

File

SocktComp

Description

This is constant SocktComp.CM_LOOKUPCOMPLETE.

1.5.3 SocktComp.CM_SOCKETMESSAGE Constant

Pascal

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

C++

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

File

ScktComp

Description

This is constant ScktComp.CM_SOCKETMESSAGE.

2 SConnect Namespace

This is namespace SConnect.

Topics

Name	Description
Classes (see page 147)	The following table lists classes in this documentation.
Interfaces (see page 221)	The following table lists interfaces in this documentation.
Functions (see page 237)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 238)	The following table lists structs, records, enums in this documentation.
Types (see page 239)	The following table lists types in this documentation.
Variables (see page 241)	The following table lists variables in this documentation.
Constants (see page 243)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 147)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 148)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 176).
	TCustomDataBlockInterpreter (see page 148)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 153)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 163)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 171)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 175)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 176)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 185)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 191)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 204)	This is class SConnect.TTransportThread.
	TWebConnection (see page 209)	TWebConnection uses HTTP to manage the connection to an application server.

Constants

Name	Description
asCreateObject (see page 243)	This is constant SConnect.asCreateObject.

asError (see page 243)	This is constant SConnect.asError.
asFreeObject (see page 244)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 244)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 244)	This is constant SConnect.asGetGUID.
asGetID (see page 245)	This is constant SConnect.asGetID.
asGetServers (see page 245)	This is constant SConnect.asGetServers.
asInvoke (see page 245)	This is constant SConnect.asInvoke.
asMask (see page 246)	This is constant SConnect.asMask.
asSoapCommand (see page 246)	This is constant SConnect.asSoapCommand.
CallSig (see page 246)	This is constant SConnect.CallSig.
ResultSig (see page 247)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 247)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 247)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 248)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 248)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 248)	This is constant SConnect.THREAD_SENDSTREAM.

Functions

	Name	Description
	GetPacketInterceptorList (see page 237)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 237)	This is function SConnect.LoadWinSock2.

Interfaces

	Name	Description
	IDataBlock (see page 221)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 229)	IDataIntercept is the interface used by TSocketConnection (see page 176) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 231)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 233)	ITransport is the interface used by TStreamedConnection (see page 191) to connect to, read from, and write to a remote application server.

Structs, Records, Enums

	Name	Description
	TVarFlag (see page 238)	This is record SConnect.TVarFlag.

Types

Name	Description
PIntArray (see page 239)	This is type SConnect.PIntArray.
PVariantArray (see page 239)	This is type SConnect.PVariantArray.
TIntArray (see page 240)	This is type SConnect.TIntArray.
TVarFlags (see page 240)	This is type SConnect.TVarFlags.
TVariantArray (see page 240)	This is type SConnect.TVariantArray.

Variables

Name	Description
WSACloseEvent (see page 241)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 241)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 242)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 242)	This is variable SConnect.WSAResetEvent.

2.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 147)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 148)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 176).
	TCustomDataBlockInterpreter (see page 148)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 153)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 163)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 171)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 175)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 176)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 185)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 191)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 204)	This is class SConnect.TTransportThread.
	TWebConnection (see page 209)	TWebConnection uses HTTP to manage the connection to an application server.

2.1.1 EInterpreterError Class

EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.

Class Hierarchy

Pascal

```
EInterpreterError = class(Exception);
```

C++

```
class EInterpreterError : public Exception;
```

File

SConnect

Description

Data block interpreters (TCustomDataBlockInterpreter (see page 148) descendants) raise EInterpreterError when they can't interpret the information in a data block. Streamed connection components such as TSocketConnection (see page 176) use data block interpreters to send interface calls to and receive interface calls from a remote application server.

See Also

TCustomDataBlockInterpreter (see page 148)

TSocketConnection (see page 176)

TSOAPConnection

TWebConnection (see page 209)

2.1.2 ESocketConnectionError Class

ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 176).

Class Hierarchy

Pascal

```
ESocketConnectionError = class(Exception);
```

C++

```
class ESocketConnectionError : public Exception;
```

File

SConnect

Description

Socket connection components raise ESocketConnectionError when the socket they use to send and receive messages reports an error, or when the information read from the socket is incomplete.

See Also

TSocketConnection (see page 176)

2.1.3 TCustomDataBlockInterpreter Class

TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.

Class Hierarchy

Pascal

```
TCustomDataBlockInterpreter = class;
```

C++

```
class TCustomDataBlockInterpreter;
```

File

SConnect

Description

TCustomDataBlockInterpreter introduces the methods needed for marshaling the messages that pass between a client application and an application server in a multi-tiered database application.

Streamed connection components use data block interpreters to marshal of messages sent to and received from an application server. An object that supports the IDataBlock (see page 221) interface stores the messages themselves.

See Also

TStreamedConnection (see page 191)

IDataBlock (see page 221)

2.1.3.1 TCustomDataBlockInterpreter Members

The following tables list the members exposed by TCustomDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 150)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 150)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 150)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 150)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 151)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 151)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 151)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 151)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 151)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 152)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 152)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 152)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 153)	This is StoreObject, a member of class TCustomDataBlockInterpreter.
	UnlockObject (see page 153)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 152)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

2.1.3.2 TCustomDataBlockInterpreter Methods

The methods of the TCustomDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 150)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 150)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 150)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 150)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 151)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 151)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 151)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 151)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 151)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 152)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 152)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 152)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 153)	This is StoreObject, a member of class TCustomDataBlockInterpreter.

	UnlockObject (see page 153)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 152)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

2.1.3.2.1 TCustomDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value) = 0;
```

Description

This is AddDispatch, a member of class TCustomDataBlockInterpreter.

2.1.3.2.2 TCustomDataBlockInterpreter.CallCreateObject Method

Pascal

```
function CallCreateObject(Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(UnicodeString Name) = 0;
```

Description

This is CallCreateObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.3 TCustomDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); virtual; abstract;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex) = 0;
```

Description

This is CallFreeObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.4 TCustomDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID,
void * Names, int NameCount, int LocaleID, void * DispIDs) = 0;
```

Description

This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.

2.1.3.2.5 TCustomDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallGetServerList() = 0;
```

Description

This is CallGetServerList, a member of class TCustomDataBlockInterpreter.

2.1.3.2.6 TCustomDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID
IID, int LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr) = 0;
```

Description

This is CallInvoke, a member of class TCustomDataBlockInterpreter.

2.1.3.2.7 TCustomDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID) = 0;
```

Description

This is CanCreateObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.8 TCustomDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CreateObject(const UnicodeString Name) = 0;
```

Description

This is CreateObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.9 TCustomDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID) = 0;
```

Description

This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.10 TCustomDataBlockInterpreter.InterpretData Method

Pascal

```
procedure InterpretData(const Data: IDataBlock); virtual; abstract;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data) = 0;
```

Description

This is InterpretData, a member of class TCustomDataBlockInterpreter.

2.1.3.2.11 TCustomDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; virtual; abstract;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID) = 0;
```

Description

This is LockObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.12 TCustomDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); virtual; abstract;
```

C++

```
virtual __fastcall ReleaseObject(int ID) = 0;
```

Description

This is ReleaseObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.13 TCustomDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value) = 0;
```

Description

This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.

2.1.3.2.14 TCustomDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value) = 0;
```

Description

This is StoreObject, a member of class TCustomDataBlockInterpreter.

2.1.3.2.15 TCustomDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); virtual; abstract;
```

C++

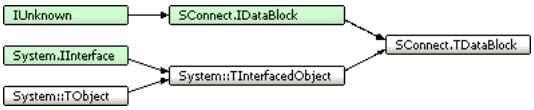
```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp) = 0;
```

Description

This is UnlockObject, a member of class TCustomDataBlockInterpreter.

2.1.4 TDataBlock Class

Class Hierarchy



Pascal

```
TDataBlock = class(TInterfacedObject, IDataBlock);
```

C++

```
class TDataBlock : public TInterfacedObject, public IDataBlock;
```

File

SConnect

Description

This is class SConnect.TDataBlock.

2.1.4.1 TDataBlock Members

The following tables list the members exposed by TDataBlock.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	Clear (see page 156)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 157)	Returns the value of the BytesReserved property.
	GetMemory (see page 157)	Returns the value of the Memory property.
	GetSignature (see page 157)	Returns the value of the Signature property.
	GetSize (see page 158)	Returns the value of the Size property.
	GetStream (see page 158)	Returns the value of the Stream property.
	IgnoreStream (see page 158)	Informs the data block when its stream is invalidated.
	InitData (see page 158)	Adds data to the data block.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	Read (see page 159)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 159)	Sets the value of the Signature property.
	SetSize (see page 160)	Sets the value of the Size property.
	Write (see page 160)	Writes a specified number of bytes from a buffer.

Protected Properties

	Name	Description
R	BytesReserved (see page 161)	Indicates the number of bytes from Memory that are reserved for internal use.
R	Memory (see page 161)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 162)	Indicates the contents of the data block.
	Size (see page 162)	Indicates the size of the data in the memory buffer.
R	Stream (see page 162)	Provides access to an internal stream that contains the data block's memory.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 155)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 155)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.

	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

2.1.4.2 TDataBlock.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataBlock();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

2.1.4.3 TDataBlock.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlock();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

2.1.4.4 TDataBlock Methods

The methods of the TDataBlock class are listed here.

Protected Methods

	Name	Description
	Clear (see page 156)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 157)	Returns the value of the BytesReserved property.
	GetMemory (see page 157)	Returns the value of the Memory property.
	GetSignature (see page 157)	Returns the value of the Signature property.
	GetSize (see page 158)	Returns the value of the Size property.
	GetStream (see page 158)	Returns the value of the Stream property.
	IgnoreStream (see page 158)	Informs the data block when its stream is invalidated.
	InitData (see page 158)	Adds data to the data block.
	Read (see page 159)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 159)	Sets the value of the Signature property.
	SetSize (see page 160)	Sets the value of the Size property.
	Write (see page 160)	Writes a specified number of bytes from a buffer.

2.1.4.4.1 TDataBlock.Clear Method

Removes all data from Memory except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory or when the information written to Memory has been used. Clear reclaims the

resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 227)

Memory (see page 227)

2.1.4.4.2 TDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved property.

See Also

BytesReserved (see page 227)

2.1.4.4.3 TDataBlock.GetMemory Method

Returns the value of the Memory property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory property.

See Also

Memory (see page 227)

2.1.4.4.4 TDataBlock.GetSignature Method

Returns the value of the Signature property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature property.

See Also

Signature (see page 228)

2.1.4.4.5 TDataBlock.GetSize Method

Returns the value of the Size property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size property.

See Also

Size (see page 228)

2.1.4.4.6 TDataBlock.GetStream Method

Returns the value of the Stream property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream property.

See Also

Stream (see page 229)

2.1.4.4.7 TDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream. After a call to IgnoreStream, the data block will not free the memory for its internal Stream component.

See Also

Stream (see page 229)

2.1.4.4.8 TDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 228)

2.1.4.4.9 TDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Write (see page 226)

Memory (see page 227)

Size (see page 228)

BytesReserved (see page 227)

2.1.4.4.10 TDataBlock.SetSignature Method

Sets the value of the Signature property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature property.

See Also

Signature (see page 228)

2.1.4.4.11 TDataBlock.SetSize Method

Sets the value of the Size property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size property.

See Also

Size (see page 228)

2.1.4.4.12 TDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory. Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Read (see page 225)

Memory (see page 227)

Size (see page 228)

BytesReserved (see page 227)

2.1.4.5 TDataBlock Properties

The properties of the TDataBlock class are listed here.

Protected Properties

	Name	Description
R	BytesReserved (see page 161)	Indicates the number of bytes from Memory that are reserved for internal use.
R	Memory (see page 161)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 162)	Indicates the contents of the data block.
	Size (see page 162)	Indicates the size of the data in the memory buffer.
R	Stream (see page 162)	Provides access to an internal stream that contains the data block's memory.

2.1.4.5.1 TDataBlock.BytesReserved Property

Indicates the number of bytes from Memory that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory can't be used for reading and writing. When interpreting the value of the Size property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 227)

Size (see page 228)

2.1.4.5.2 TDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved) for internal information such as the value of Signature,

Note: When implementing the Receive and Send methods of the ITransport interface, component writers should read directly from and write directly to Memory, rather than using the Read and Write methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 228)

ITransport (see page 233)

BytesReserved (see page 227)

Signature (see page 228)

Stream (see page 229)

2.1.4.5.3 TDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write Signature values to indicate what type of data is written to the data block. Read Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved bytes of Memory.

See Also

BytesReserved (see page 227)

Memory (see page 227)

2.1.4.5.4 TDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory, excluding the reserved bytes. Set Size to ensure that Memory is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 227)

Memory (see page 227)

Signature (see page 228)

2.1.4.5.5 TDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 227)

IgnoreStream (see page 224)

TMemoryStream

2.1.5 TDataBlockInterpreter Class

TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.

Class Hierarchy

Pascal

```
TDataBlockInterpreter = class(TCustomDataBlockInterpreter);
```

C++

```
class TDataBlockInterpreter : public TCustomDataBlockInterpreter;
```

File

SConnect

Description

Many streamed connection components use TDataBlockInterpreter to handle the marshaling of messages sent to and received from an application server. An object that supports the IDataBlock (see page 221) interface stores the messages themselves.

See Also

TStreamedConnection (see page 191)

IDataBlock (see page 221)

TSocketConnection (see page 176)

TWebConnection (see page 209)

2.1.5.1 TDataBlockInterpreter Members

The following tables list the members exposed by TDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 165)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 166)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 166)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 166)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 167)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 167)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 167)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 167)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 167)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 168)	This is DoFreeObject, a member of class TDataBlockInterpreter.

	DoGetAppServerList (see page 168)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 168)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 168)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 169)	This is DoInvoke, a member of class TDataBlockInterpreter.
V	InternalCreateObject (see page 169)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
V	LockObject (see page 169)	This is LockObject, a member of class TDataBlockInterpreter.
V	ReleaseObject (see page 170)	This is ReleaseObject, a member of class TDataBlockInterpreter.
V	RemoveDispatch (see page 170)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
V	StoreObject (see page 170)	This is StoreObject, a member of class TDataBlockInterpreter.
V	UnlockObject (see page 170)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
V	CallCreateObject (see page 166)	Creates a COM object on a remote application server given its class name.
	Create (see page 164)	Creates and initializes an instance of TDataBlockInterpreter (see page 163).
V	Destroy (see page 164)	Destroys an instance of TDataBlockInterpreter (see page 163).
V	InterpretData (see page 169)	Interprets a call received from a remote application server.

2.1.5.2 TDataBlockInterpreter.Create Constructor

Creates and initializes an instance of TDataBlockInterpreter (see page 163).

Pascal

```
constructor Create(SendDataBlock: ISendDataBlock; CheckRegValue: string);
```

C++

```
__fastcall TDataBlockInterpreter(ISendDataBlock SendDataBlock, UnicodeString CheckRegValue);
```

Description

Applications should not need to instantiate data block interpreters. Typically, they are instantiated by the streamed connection components that use them to marshal data. To access a streamed connection component's data block interpreter instance, use its Interpreter property.

Create takes two arguments:

An ISendDataBlock (see page 231) interface that the data block interpreter can use to send information in a data block to its destination object (usually the remote data module of an application server).

A string that specifies a registry key value which must appear for application servers that can be accessed using the data block interpreter.

See Also

Interpreter (see page 203)

2.1.5.3 TDataBlockInterpreter.Destroy Destructor

Destroys an instance of TDataBlockInterpreter (see page 163).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlockInterpreter();
```

Description

Applications have no need to free data block interpreters. They are freed by the streamed connection component that uses them to marshal interface calls.

See Also

System::TObject::Free (see page 3382)

2.1.5.4 TDataBlockInterpreter Methods

The methods of the TDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 165)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 166)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 166)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 166)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 167)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 167)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 167)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 167)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 167)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 168)	This is DoFreeObject, a member of class TDataBlockInterpreter.
	DoGetAppServerList (see page 168)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 168)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 168)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 169)	This is DoInvoke, a member of class TDataBlockInterpreter.
	InternalCreateObject (see page 169)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
	LockObject (see page 169)	This is LockObject, a member of class TDataBlockInterpreter.
	ReleaseObject (see page 170)	This is ReleaseObject, a member of class TDataBlockInterpreter.
	RemoveDispatch (see page 170)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
	StoreObject (see page 170)	This is StoreObject, a member of class TDataBlockInterpreter.
	UnlockObject (see page 170)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
	CallCreateObject (see page 166)	Creates a COM object on a remote application server given its class name.
	InterpretData (see page 169)	Interprets a call received from a remote application server.

2.1.5.4.1 TDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value);
```

Description

This is AddDispatch, a member of class TDataBlockInterpreter.

2.1.5.4.2 TDataBlockInterpreter.CallCreateObject Method

Creates a COM object on a remote application server given its class name.

Pascal

```
function CallCreateObject(Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(UnicodeString Name);
```

Description

CallCreateObject sends a message to the remote application server, telling it to create an object of the class specified by the Name parameter. Name is the string representation of the Class ID (CLSID). CreateObject (see page 167) returns a reference to an interface that can be used to communicate with the remote COM object.

See Also

CreateOleObject

2.1.5.4.3 TDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); override;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex);
```

Description

This is CallFreeObject, a member of class TDataBlockInterpreter.

2.1.5.4.4 TDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID, void * Names,
int NameCount, int LocaleID, void * DispIDs);
```

Description

This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.

2.1.5.4.5 TDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallGetServerList();
```

Description

This is CallGetServerList, a member of class TDataBlockInterpreter.

2.1.5.4.6 TDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; override;
```

C++

```
virtual __fastcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID IID, int
LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is CallInvoke, a member of class TDataBlockInterpreter.

2.1.5.4.7 TDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID);
```

Description

This is CanCreateObject, a member of class TDataBlockInterpreter.

2.1.5.4.8 TDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CreateObject(const UnicodeString Name);
```

Description

This is CreateObject, a member of class TDataBlockInterpreter.

2.1.5.4.9 TDataBlockInterpreter.DoCreateObject Method

Pascal

```
procedure DoCreateObject(const Data: IDataBlock);
```

C++

```
__fastcall DoCreateObject(const IDataBlock Data);
```

Description

This is DoCreateObject, a member of class TDataBlockInterpreter.

2.1.5.4.10 TDataBlockInterpreter.DoCustomAction Method

Pascal

```
function DoCustomAction(Action: Integer; const Data: IDataBlock): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomAction(int Action, const IDataBlock Data);
```

Description

This is DoCustomAction, a member of class TDataBlockInterpreter.

2.1.5.4.11 TDataBlockInterpreter.DoFreeObject Method

Pascal

```
procedure DoFreeObject(const Data: IDataBlock);
```

C++

```
__fastcall DoFreeObject(const IDataBlock Data);
```

Description

This is DoFreeObject, a member of class TDataBlockInterpreter.

2.1.5.4.12 TDataBlockInterpreter.DoGetAppServerList Method

Pascal

```
procedure DoGetAppServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetAppServerList(const IDataBlock Data);
```

Description

This is DoGetAppServerList, a member of class TDataBlockInterpreter.

2.1.5.4.13 TDataBlockInterpreter.DoGetIDsOfNames Method

Pascal

```
procedure DoGetIDsOfNames(const Data: IDataBlock);
```

C++

```
__fastcall DoGetIDsOfNames(const IDataBlock Data);
```

Description

This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.

2.1.5.4.14 TDataBlockInterpreter.DoGetServerList Method

Pascal

```
procedure DoGetServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetServerList(const IDataBlock Data);
```

Description

This is DoGetServerList, a member of class TDataBlockInterpreter.

2.1.5.4.15 TDataBlockInterpreter.DoInvoke Method

Pascal

```
procedure DoInvoke(const Data: IDataBlock);
```

C++

```
__fastcall DoInvoke(const IDataBlock Data);
```

Description

This is DoInvoke, a member of class TDataBlockInterpreter.

2.1.5.4.16 TDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID);
```

Description

This is InternalCreateObject, a member of class TDataBlockInterpreter.

2.1.5.4.17 TDataBlockInterpreter.InterpretData Method

Interprets a call received from a remote application server.

Pascal

```
procedure InterpretData(const Data: IDataBlock); override;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data);
```

Description

Call InterpretData to respond to a COM interface call that is marshaled to a data block. The Data parameter specifies an IDataBlock (see page 221) interface that can be used to read the marshaled version of the message.

InterpretData reads the call, interprets it as an IDispatch (see page 3498) call, and performs the appropriate actions (such as invoking a local COM interface).

2.1.5.4.18 TDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; override;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID);
```

Description

This is LockObject, a member of class TDataBlockInterpreter.

2.1.5.4.19 TDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); override;
```

C++

```
virtual __fastcall ReleaseObject(int ID);
```

Description

This is ReleaseObject, a member of class TDataBlockInterpreter.

2.1.5.4.20 TDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value);
```

Description

This is RemoveDispatch, a member of class TDataBlockInterpreter.

2.1.5.4.21 TDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; override;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value);
```

Description

This is StoreObject, a member of class TDataBlockInterpreter.

2.1.5.4.22 TDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); override;
```

C++

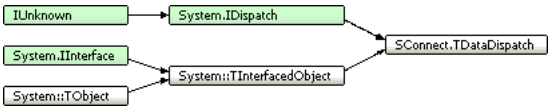
```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp);
```

Description

This is UnlockObject, a member of class TDataBlockInterpreter.

2.1.6 TDataDispatch Class

Class Hierarchy



Pascal

```
TDataDispatch = class(TInterfacedObject, IDispatch);
```

C++

```
class TDataDispatch : public TInterfacedObject, public IDispatch;
```

File

SConnect

Description

This is class SConnect.TDataDispatch.

2.1.6.1 TDataDispatch Members

The following tables list the members exposed by TDataDispatch.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetIDsOfNames (see page 173)	This is GetIDsOfNames, a member of class TDataDispatch.
	GetTypeInfo (see page 174)	This is GetTypeInfo, a member of class TDataDispatch.
	GetTypeInfoCount (see page 174)	This is GetTypeInfoCount, a member of class TDataDispatch.
	Invoke (see page 174)	This is Invoke, a member of class TDataDispatch.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	DispatchIndex (see page 174)	This is DispatchIndex, a member of class TDataDispatch.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.

	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 172)	This is Create, a member of class TDataDispatch.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 173)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

2.1.6.2 TDataDispatch.Create Constructor

Pascal

```
constructor Create(Interpreter: TCustomDataBlockInterpreter; DispatchIndex: Integer);
```

C++

```
__fastcall TDataDispatch(TCustomDataBlockInterpreter Interpreter, int DispatchIndex);
```

Description

This is Create, a member of class TDataDispatch.

2.1.6.3 TDataDispatch.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataDispatch();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

2.1.6.4 TDataDispatch Methods

The methods of the TDataDispatch class are listed here.

Protected Methods

	Name	Description
	GetIDsOfNames (see page 173)	This is GetIDsOfNames, a member of class TDataDispatch.
	GetTypeInfo (see page 174)	This is GetTypeInfo, a member of class TDataDispatch.
	GetTypeInfoCount (see page 174)	This is GetTypeInfoCount, a member of class TDataDispatch.
	Invoke (see page 174)	This is Invoke, a member of class TDataDispatch.

2.1.6.4.1 TDataDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TDataDispatch.

2.1.6.4.2 TDataDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TDataDispatch.

2.1.6.4.3 TDataDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TDataDispatch.

2.1.6.4.4 TDataDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TDataDispatch.

2.1.6.5 TDataDispatch Properties

The properties of the TDataDispatch class are listed here.

Protected Properties

	Name	Description
	DispatchIndex (see page 174)	This is DispatchIndex, a member of class TDataDispatch.

2.1.6.5.1 TDataDispatch.DispatchIndex Property

Pascal

```
property DispatchIndex: Integer;
```

C++

```
__property int DispatchIndex;
```

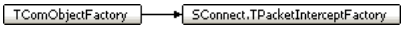
Description

This is DispatchIndex, a member of class TDataDispatch.

2.1.7 TPacketInterceptFactory Class

TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.

Class Hierarchy



Pascal

```
TPacketInterceptFactory = class(TComObjectFactory);
```

C++

```
class TPacketInterceptFactory : public TComObjectFactory;
```

File

SConnect

Description

In Delphi, use TPacketInterceptFactory as the class factory for COM objects that implement the IDataIntercept (see page 229) interface. Data packet interceptors implement the IDataIntercept (see page 229) interface, which they use in multi-tiered database applications to modify the data that passes between a socket connection component in the client application and scktsrvr.exe on the server. TPacketInterceptFactory registers the data packet interceptor so that can appear in the drop-down list of the Object Inspector when setting the socket connection's InterceptName property and in the corresponding list in the properties dialog of scktsrvr.exe.

TPacketInterceptFactory supports the IUnknown (see page 3828), IClassFactory, and IClassFactory2 interfaces. The IClassFactory interface is implemented by class objects associated with class identifiers (CLSIDs) for the classes that they instantiate. IClassFactory must be implemented for every object definition, so that objects of that type can be created. The IClassFactory2 interface is implemented by class factories for classes that need support for licensing.

See Also

ComClassManager

CreateComObject

CreateOleObject

TStreamedConnection (see page 191)

IDataIntercept (see page 229)

Specifying a Connection Using Sockets

2.1.7.1 TPacketInterceptFactory Members

The following tables list the members exposed by TPacketInterceptFactory.

Public Methods

	Name	Description
	UpdateRegistry (see page 176)	Registers or unregisters COM objects created by the factory.

2.1.7.2 TPacketInterceptFactory Methods

The methods of the TPacketInterceptFactory class are listed here.

Public Methods

	Name	Description
	UpdateRegistry (see page 176)	Registers or unregisters COM objects created by the factory.

2.1.7.2.1 TPacketInterceptFactory.UpdateRegistry Method

Registers or unregisters COM objects created by the factory.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

In addition to the standard COM registration that allows clients to instantiate the data packet interceptor that this factory creates, TPacketInterceptFactory (see page 175) also registers its associated COM object as a data packet interceptor. This allows the object's ProgID to appear in the drop-down list for a socket connection's InterceptName property, and to appear in the corresponding list on the properties dialog for scktsrvr.exe.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject

InterceptName (see page 202)

2.1.8 TSocketConnection Class

TSocketConnection uses Windows sockets to manage the connection to an application server.

Class Hierarchy

Pascal

```
TSocketConnection = class(TStreamedConnection);
```

C++

```
class TSocketConnection : public TStreamedConnection;
```

File

SConnect

Description

TSocketConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Socket connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Call into a separate COM object that encodes and decodes the content of messages.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSocketConnection establishes the initial connection between the client application and a remote application server using TCP/IP. To use TSocketConnection, the application server must be running ScktSrvr.exe.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the socket connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using HTTP, use the TWebConnection (see page 209) component instead. To connect using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TWebConnection (see page 209)

TSoapConnection (see page 1128)

Specifying a Connection Using Sockets

2.1.8.1 TSocketConnection Members

The following tables list the members exposed by TSocketConnection.

Protected Methods

	Name	Description
	_AddRef (see page 194)	Increments an internal reference counter.
	_Release (see page 194)	Decrements an internal reference counter.
V	CreateTransport (see page 179)	Returns an ITransport (see page 233) interface that can be used to connect to and communicate with the application server.
V	DoConnect (see page 179)	Connects to a remote application server.

	DoDisconnect (see page 196)	Disconnects from an application server.
	DoError (see page 196)	raises an exception for a specified exception object.
	GetInterpreter (see page 197)	Returns the value of the Interpreter (see page 203) property.
	InternalClose (see page 198)	Terminates a connection to the application server.
	InternalOpen (see page 198)	Opens a connection to the application server.
	QueryInterface (see page 199)	Implements the IUnknown (see page 3828) QueryInterface method.
	Send (see page 199)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 199)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 200)	Handles callbacks from the application server.
	WndProc (see page 200)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
	Handle (see page 201)	Provides access to an internal window handle that can receive messages from the transport layer.
	Interpreter (see page 203)	Provides access to the helper object that marshals interface calls for the streamed connection.

Public Methods

	Name	Description
	Create (see page 178)	Creates and initializes an instance of TSocketConnection (see page 176).
	Destroy (see page 193)	Disposes of a TStreamedConnection (see page 191) instance.
	GetInterceptorList (see page 196)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetServerList (see page 197)	Retrieves a list of registered data brokers from the remote server system.
	Send (see page 232)	Sends information retrieved from an IDataBlock (see page 221) interface to the application server.

Published Properties

	Name	Description
	Address (see page 180)	Specifies the IP address of the server.
	Host (see page 181)	Specifies the host name of the server.
	InterceptGUID (see page 181)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 182)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	ObjectBroker (see page 183)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Port (see page 183)	Specifies the port number of the socket dispatcher application on the server system.
	SupportCallbacks (see page 184)	Determines whether the streamed connection component can handle callbacks in the application server interface.

2.1.8.2 TSocketConnection.Create Constructor

Creates and initializes an instance of TSocketConnection (see page 176).

Pascal

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

C++

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

Description

Call Create to instantiate a TSocketConnection (see page 176) object at runtime. Socket connection components added to forms or data modules at design time are created automatically.

Create overrides the inherited constructor to initialize the Port (see page 183) property to 211.

See Also

Create (see page 193)

Port (see page 183)

2.1.8.3 TSocketConnection Methods

The methods of the TSocketConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 179)	Returns an ITransport (see page 233) interface that can be used to connect to and communicate with the application server.
	DoConnect (see page 179)	Connects to a remote application server.

2.1.8.3.1 TSocketConnection.CreateTransport Method

Returns an ITransport (see page 233) interface that can be used to connect to and communicate with the application server.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Socket connection components use an ITransport (see page 233) interface to connect to and communicate with the application server. CreateTransport instantiates the internal helper object that connects to the socket dispatcher on the server and sends or receives messages. CreateTransport passes the values of the Host (see page 181), Address (see page 180) and Port (see page 183) properties to this helper object and returns its ITransport (see page 233) interface.

See Also

Host (see page 181)

Address (see page 180)

Port (see page 183)

2.1.8.3.2 TSocketConnection.DoConnect Method

Connects to a remote application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is used internally to connect to the application server when the Connected property is set to true. If the Host (see page 181) or Address (see page 180) and Port (see page 183) properties are set, DoConnect uses these to locate the socket on the server machine. Otherwise, if the ObjectBroker (see page 183) property is set, it obtains these values from the

object broker.

Once the server machine is located, DoConnect opens a connection.

Note: When ObjectBroker (see page 183) is nil (Delphi) or NULL (C++), the values of Host (see page 181), Address (see page 180), and Port (see page 183) are saved with the socket connection component. If ObjectBroker (see page 183) is not nil (Delphi) or NULL (C++), these values are not saved from design time. However, once these values are set (either by values obtained from the Object Broker or by values specified in code at runtime), they are used every time a connection must be re-established. If these values subsequently fail, the socket connection component fetches new values from the Object Broker.

See Also

Connected

Host (see page 181)

Address (see page 180)

Port (see page 183)

ObjectBroker

2.1.8.4 TSocketConnection Properties

The properties of the TSocketConnection class are listed here.

Published Properties

	Name	Description
	Address (see page 180)	Specifies the IP address of the server.
	Host (see page 181)	Specifies the host name of the server.
	InterceptGUID (see page 181)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 182)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	ObjectBroker (see page 183)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Port (see page 183)	Specifies the port number of the socket dispatcher application on the server system.
	SupportCallbacks (see page 184)	Determines whether the streamed connection component can handle callbacks in the application server interface.

2.1.8.4.1 TSocketConnection.Address Property

Specifies the IP address of the server.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

Use Address to specify the IP address of the application server to which the client application connects. In addition to the address, you must provide a value for the Port (see page 183) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 1299) require either an IP address or a Host (see page 181) name to locate the server system. You can use the Host (see page 181) property instead of the Address property to specify the server system using a host name. The

Address and Host (see page 181) properties are mutually exclusive: Setting Address causes Host (see page 181) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Address property. Instead, the Object Broker supplies an Address for a server that is located dynamically at runtime. Explicitly setting the Address property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Host (see page 181)

Port (see page 183)

ObjectBroker

Specifying a Connection Using Sockets

2.1.8.4.2 TSocketConnection.Host Property

Specifies the host name of the server.

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

Use Host to specify the host name of the server system to which the client application should connect. In addition to the host name, you must provide a value for the Port (see page 183) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 1299) require either a Host name or an IP address to locate the server system. You can use the Address (see page 180) property instead of the Host property to specify the server system using an IP address. The Host and Address (see page 180) properties are mutually exclusive: Setting Host causes Address (see page 180) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Host property. Instead, the Object Broker supplies a host name for a server that is located dynamically at runtime. Explicitly setting the Host property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Address (see page 180)

Port (see page 183)

ObjectBroker

Specifying a Connection Using Sockets

2.1.8.4.3 TSocketConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName property instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptGUID. For servers that respond to TSocketConnection, you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 229)

InterceptName (see page 202)

GetInterceptorList (see page 196)

Specifying a Connection Using Sockets

2.1.8.4.4 TSocketConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property UnicodeString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory as a class factory.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection, you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 229)

InterceptGUID (see page 201)

GetInterceptorList (see page 196)

TPacketInterceptFactory (see page 175)

Specifying a Connection Using Sockets

2.1.8.4.5 TSocketConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker;
```

C++

```
__property ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect

ServerName

ServerGUID

TSimpleObjectBroker

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

2.1.8.4.6 TSocketConnection.Port Property

Specifies the port number of the socket dispatcher application on the server system.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Use Port to specify the port that the socket dispatcher uses to listen for client connection requests. By default, Port is 211, which is the default port number of ScktSrvr.exe (the socket dispatcher application). Change the value of Port if the socket dispatcher is configured to use a different port.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Port property. Instead, the Object Broker supplies a port number for a server that is located dynamically at runtime. Explicitly setting the Port property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Address (see page 180)

Host (see page 181)

ObjectBroker

Specifying a Connection Using Sockets

2.1.8.4.7 TSocketConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

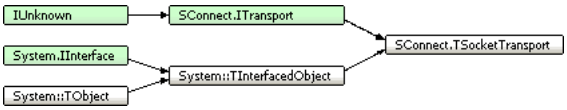
See Also

Send (see page 199)

Extending the Interface of the Application Server

2.1.9 TSocketTransport Class

Class Hierarchy



Pascal

```
TSocketTransport = class(TInterfacedObject, ITransport);
```

C++

```
class TSocketTransport : public TInterfacedObject, public ITransport;
```

File

SConnect

Description

This is class SConnect.TSocketTransport.

2.1.9.1 TSocketTransport Members

The following tables list the members exposed by TSocketTransport.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetConnected (see page 188)	Returns the value of the Connected property.
	GetWaitEvent (see page 188)	Returns the handle for an event object that is managed by the object that implements ITransport.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	Receive (see page 188)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 189)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 189)	Sets the value of the Connected property.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.

	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 186)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 187)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Address (see page 190)	This is Address, a member of class TSocketTransport.
	Connected (see page 236)	Indicates whether the connection to the application server is established.
	Host (see page 190)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 190)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 190)	This is Port, a member of class TSocketTransport.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Socket (see page 191)	This is Socket, a member of class TSocketTransport.

2.1.9.2 TSocketTransport.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSocketTransport();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

2.1.9.3 TSocketTransport.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSocketTransport();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

2.1.9.4 TSocketTransport Methods

The methods of the TSocketTransport class are listed here.

Protected Methods

	Name	Description
	GetConnected (see page 188)	Returns the value of the Connected property.
	GetWaitEvent (see page 188)	Returns the handle for an event object that is managed by the object that implements ITransport.
	Receive (see page 188)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 189)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 189)	Sets the value of the Connected property.

2.1.9.4.1 TSocketTransport.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected property.

See Also

Connected (see page 236)

2.1.9.4.2 TSocketTransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3074)

2.1.9.4.3 TSocketTransport.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 227)

Send (see page 235)

2.1.9.4.4 TSocketTransport.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 227)

Receive (see page 235)

2.1.9.4.5 TSocketTransport.SetConnected Method

Sets the value of the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected property.

2.1.9.5 TSocketTransport Properties

The properties of the TSocketTransport class are listed here.

Public Properties

	Name	Description
	Address (see page 190)	This is Address, a member of class TSocketTransport.
	Host (see page 190)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 190)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 190)	This is Port, a member of class TSocketTransport.
	Socket (see page 191)	This is Socket, a member of class TSocketTransport.

2.1.9.5.1 TSocketTransport.Address Property

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

This is Address, a member of class TSocketTransport.

2.1.9.5.2 TSocketTransport.Host Property

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

This is Host, a member of class TSocketTransport.

2.1.9.5.3 TSocketTransport.InterceptGUID Property

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

This is InterceptGUID, a member of class TSocketTransport.

2.1.9.5.4 TSocketTransport.Port Property

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

This is Port, a member of class TSocketTransport.

2.1.9.5.5 TSocketTransport.Socket Property

Pascal

```
property Socket: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Socket;
```

Description

This is Socket, a member of class TSocketTransport.

2.1.10 TStreamedConnection Class

TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.

Class Hierarchy

Pascal

```
TStreamedConnection = class(TDispatchConnection, ISendDataBlock);
```

C++

```
class TStreamedConnection : public TDispatchConnection, public ISendDataBlock;
```

File

SConnect

Description

Do not create instances of TStreamedConnection. Instead, instantiate a descendant of TStreamedConnection such as TSocketConnection (see page 176) or TWebConnection (see page 209).

TStreamedConnection handles the marshaling of all COM data types so that the streamed connection component can generate, interpret, and respond to remote COM interface calls without using DCOM. It uses an ITransport (see page 233) interface to connect to application servers and to send and receive messages.

TStreamedConnection can't actually connect to or communicate with an application server. Descendants of TStreamedConnection override the CreateTransport (see page 195) method to obtain an ITransport (see page 233) interface that enables these behaviors.

TStreamedConnection implements the ISendDataBlock (see page 231) interface.

See Also

TClientDataSet

TSocketConnection (see page 176)

TWebConnection (see page 209)

TStream

ITransport (see page 233)

TRemoteDataModule

IAppServer

2.1.10.1 TStreamedConnection Members

The following tables list the members exposed by TStreamedConnection.

Protected Methods

	Name	Description
	_AddRef (see page 194)	Increments an internal reference counter.
	_Release (see page 194)	Decrements an internal reference counter.
V	CreateTransport (see page 195)	Provides the backstop for a method that obtains an ITransport (see page 233) interface.
V	DoConnect (see page 195)	Connects to an application server.
V	DoDisconnect (see page 196)	Disconnects from an application server.
V	DoError (see page 196)	raises an exception for a specified exception object.
V	GetInterpreter (see page 197)	Returns the value of the Interpreter (see page 203) property.
V	InternalClose (see page 198)	Terminates a connection to the application server.
V	InternalOpen (see page 198)	Opens a connection to the application server.
	QueryInterface (see page 199)	Implements the IUnknown (see page 3828) QueryInterface method.
	Send (see page 199)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 199)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 200)	Handles callbacks from the application server.
	WndProc (see page 200)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
R	Handle (see page 201)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 201)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 202)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 203)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 203)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Methods

	Name	Description
V	Create (see page 193)	Creates and initializes an instance of TStreamedConnection (see page 191).
V	Destroy (see page 193)	Disposes of a TStreamedConnection (see page 191) instance.
V	GetInterceptorList (see page 196)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
V	GetServerList (see page 197)	Retrieves a list of registered data brokers from the remote server system.

2.1.10.2 TStreamedConnection.Create Constructor

Creates and initializes an instance of TStreamedConnection (see page 191).

Pascal

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

C++

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

Description

Call Create to instantiate a streamed connection component at runtime. Connection components placed in forms or data modules at design time are instantiated automatically.

After calling the inherited constructor, Create instantiates the data block interpreter accessed by the Interpreter (see page 203) property.

See Also

Interpreter (see page 203)

Create

2.1.10.3 TStreamedConnection.Destroy Destructor

Disposes of a TStreamedConnection (see page 191) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStreamedConnection();
```

Description

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

Destroy ensures that the connection to the server is closed. It then frees all resources and helper objects before calling the inherited destructor.

See Also

System::TObject::Free (see page 3382)

2.1.10.4 TStreamedConnection Methods

The methods of the TStreamedConnection class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 194)	Increments an internal reference counter.
	_Release (see page 194)	Decrements an internal reference counter.
V	CreateTransport (see page 195)	Provides the backstop for a method that obtains an ITransport (see page 233) interface.

	DoConnect (see page 195)	Connects to an application server.
	DoDisconnect (see page 196)	Disconnects from an application server.
	DoError (see page 196)	raises an exception for a specified exception object.
	GetInterpreter (see page 197)	Returns the value of the Interpreter (see page 203) property.
	InternalClose (see page 198)	Terminates a connection to the application server.
	InternalOpen (see page 198)	Opens a connection to the application server.
	QueryInterface (see page 199)	Implements the IUnknown (see page 3828) QueryInterface method.
	Send (see page 199)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 199)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 200)	Handles callbacks from the application server.
	WndProc (see page 200)	Dispatches messages from the transport layer.

Public Methods

	Name	Description
	GetInterceptorList (see page 196)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetServerList (see page 197)	Retrieves a list of registered data brokers from the remote server system.

2.1.10.4.1 TStreamedConnection._AddRef Method

Increments an internal reference counter.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IUnknown (see page 3828) interface method, _AddRef, by incrementing an internal reference counter. The reference count is decreased by calling _Release (see page 194).

See Also

_Release (see page 194)

QueryInterface (see page 199)

2.1.10.4.2 TStreamedConnection._Release Method

Decrements an internal reference counter.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IUnknown (see page 3828) interface method, _Release, by decrementing an internal reference counter. The reference count is increased by calling _AddRef (see page 194)

See Also

_AddRef (see page 194)

QueryInterface (see page 199)

2.1.10.4.3 TStreamedConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport (see page 233) interface.

Pascal

```
function CreateTransport: ITransport; virtual;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport (see page 233) interface to connect to and communicate with the application server. As implemented in TStreamedConnection (see page 191), CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection (see page 191) override CreateTransport to obtain an ITransport (see page 233) interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport (see page 233) interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 199)

DoConnect (see page 195)

DoDisconnect (see page 196)

ThreadReceivedStream (see page 200)

2.1.10.4.4 TStreamedConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport (see page 233) interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection (see page 191) override CreateTransport (see page 195) rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

CreateTransport (see page 195)

2.1.10.4.5 TStreamedConnection.DoDisconnect Method

Disconnects from an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected property is set to false. Using the ITransport (see page 233) interface, DoDisconnect drops the connection to the application server.

See Also

Connected

CreateTransport (see page 195)

2.1.10.4.6 TStreamedConnection.DoError Method

raises an exception for a specified exception object.

Pascal

```
procedure DoError(E: Exception); virtual;
```

C++

```
virtual __fastcall DoError(Exception E);
```

Description

TStreamedConnection (see page 191) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 201) property. The connection component uses DoError to raise those exceptions.

Note: DoError is called by the ThreadException (see page 199) method, which is the message handler for thread exceptions. It is also called when thread exception messages are taken from the message queue before they are received by the internal window procedure.

See Also

ThreadException (see page 199)

Handle (see page 201)

EInterpreterError (see page 147)

2.1.10.4.7 TStreamedConnection.GetInterceptorList Method

Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.

Pascal

```
function GetInterceptorList: OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetInterceptorList();
```

Description

Call GetInterceptorList to obtain a list of ProgIDs that can be assigned to the InterceptName (see page 202) property. GetInterceptorList retrieves a list of all COM objects that have been registered by TPacketInterceptFactory (see page 175) as data packet interceptors (objects that support the IDataIntercept (see page 229) interface that can be used to intercept and transform messages passed between a connection component and an application server.)

GetInterceptorList returns a Variant (see page 3430) array of strings. Any of these strings can be used as a valid setting for the InterceptName (see page 202) property.

At design time, TStreamedConnection (see page 191) uses the GetInterceptorList method to populate the InterceptName (see page 202) drop-down in the Object Inspector.

See Also

IDataIntercept (see page 229)

InterceptName (see page 202)

InterceptGUID (see page 201)

TPacketInterceptFactory (see page 175)

2.1.10.4.8 TStreamedConnection.GetInterpreter Method

Returns the value of the Interpreter (see page 203) property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; virtual;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter (see page 203) property. As implemented in TStreamedConnection (see page 191), GetInterpreter instantiates a TDataBlockInterpreter (see page 163) object.

See Also

Interpreter (see page 203)

TDataBlockInterpreter (see page 163)

2.1.10.4.9 TStreamedConnection.GetServerList Method

Retrieves a list of registered data brokers from the remote server system.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Call GetServerList to obtain a list of servers that are registered on the system to which the streamed connection communicates. GetServerList returns a Variant (see page 3430) array of strings. Any of these strings can be used as a valid setting for the ServerName property.

At design time, TStreamedConnection (see page 191) uses the GetServerList method to populate the ServerName drop-down in the Object Inspector. Note that, unlike COM-based connection components (which populate that list with servers that are registered on the client machine), this results in a list of application servers that are registered on the server machine.

See Also

ServerName

2.1.10.4.10 TStreamedConnection.InternalClose Method

Terminates a connection to the application server.

Pascal

```
procedure InternalClose; virtual;
```

C++

```
virtual __fastcall InternalClose();
```

Description

Streamed connection components use InternalClose to handle the details of closing a connection to the application server. This method is called by the DoDisconnect (see page 196) method (which implements the Connected property) and at other times when the connection component must terminate a connection.

See Also

Connected

DoDisconnect (see page 196)

2.1.10.4.11 TStreamedConnection.InternalOpen Method

Opens a connection to the application server.

Pascal

```
procedure InternalOpen; virtual;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

Streamed connection components use InternalOpen to handle the details of establishing a connection to the application server. This method is called by the DoConnect (see page 195) method (which implements the Connected property) and at other times when the connection component must establish a connection.

See Also

Connected

DoConnect (see page 195)

2.1.10.4.12 TStreamedConnection.QueryInterface Method

Implements the IUnknown (see page 3828) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall; reintroduce;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface returns a reference to the interface specified by the IID parameter as the Obj parameter. It the streamed connection supports the specified interface, QueryInterface returns 0. If the streamed connection does not support the interface, the Obj parameter is nil (Delphi) or NULL (C++) and QueryInterface returns E_NOINTERFACE (see page 3912).

See Also

_AddRef (see page 194)

ISendDataBlock

2.1.10.4.13 TStreamedConnection.Send Method

Sends an interface call to the application server and returns the response.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

The data block interpreter (specified by the Interpreter (see page 203) property) calls Send to dispatch interface messages to the application server. It supplies an IDataBlock (see page 221) interface that encapsulates the interface from the application server.

Communication with the application server occurs on a separate thread. The WaitForResult parameter indicates whether execution on the main thread should halt until the interface call returns a result (WaitForResult == true), or whether the interface call can occur asynchronously (WaitForResult == false). If WaitForResult is true, Send returns an IDataBlock (see page 221) interface (_di_IDataBlock in C++) that supplies the result.

See Also

Interpreter (see page 203)

ThreadReceivedStream (see page 200)

2.1.10.4.14 TStreamedConnection.ThreadException Method

raises an exception for the thread that communicates with the application server.

Pascal

```
procedure ThreadException(var Message: TMessage); message THREAD_EXCEPTION;
```

C++

```
__fastcall ThreadException(TMessage Message);
```

Description

TStreamedConnection (see page 191) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 201) property. ThreadException is the message handler for that Windows message. It raises an exception in the main execution thread.

See Also

Handle (see page 201)

2.1.10.4.15 TStreamedConnection.ThreadReceivedStream Method

Handles callbacks from the application server.

Pascal

```
procedure ThreadReceivedStream(var Message: TMessage); message THREAD_RECEIVEDSTREAM;
```

C++

```
__fastcall ThreadReceivedStream(TMessage Message);
```

Description

TStreamedConnection (see page 191) uses a separate thread for interface calls to and from the application server. When the application server calls a client application interface, that thread sends a message to the Window handle specified by the Handle (see page 201) property. ThreadReceivedStream is the message handler for the Windows message from the application server.

ThreadReceivedStream passes the interface call to the data block interpreter specified by the Interpreter (see page 203) property.

See Also

Handle (see page 201)

Interpreter (see page 203)

2.1.10.4.16 TStreamedConnection.WndProc Method

Dispatches messages from the transport layer.

Pascal

```
procedure WndProc(var Message: TMessage);
```

C++

```
__fastcall WndProc(TMessage Message);
```

Description

TStreamedConnection (see page 191) uses a separate thread for interface calls to and from the application server. That thread communicates with the streamed connection component by sending Windows messages to the handle specified by the Handle (see page 201) property. WndProc is the window procedure associated with that Window handle. It dispatches any messages that have message handlers such as ThreadException (see page 199) or ThreadReceivedStream (see page 200). All other messages are ignored.

See Also

Handle (see page 201)

ThreadException (see page 199)

ThreadReceivedStream (see page 200)

2.1.10.5 TStreamedConnection Properties

The properties of the TStreamedConnection class are listed here.

Protected Properties

	Name	Description
	Handle (see page 201)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 201)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 202)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	Interpreter (see page 203)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 203)	Determines whether the streamed connection component can handle callbacks in the application server interface.

2.1.10.5.1 TStreamedConnection.Handle Property

Provides access to an internal window handle that can receive messages from the transport layer.

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received.

The internal thread that handles these communication tasks uses Windows messages to coordinate events. TStreamedConnection (see page 191) uses the Handle property to provide this internal thread with a window handle that it can use.

See Also

WndProc (see page 200)

ThreadReceivedStream (see page 200)

ThreadException (see page 199)

2.1.10.5.2 TStreamedConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept (see page 229) interface. Use the IDataIntercept (see page 229) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName (see page 202) property instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptGUID. For servers that respond to TSocketConnection (see page 176), you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 229)

InterceptName (see page 202)

GetInterceptorList (see page 196)

Specifying a Connection Using Sockets

2.1.10.5.3 TStreamedConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property UnicodeString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory (see page 175) as a class factory.

The COM object must implement the IDataIntercept (see page 229) interface. Use the IDataIntercept (see page 229) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID (see page 201) property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection (see page 176), you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 229)

InterceptGUID (see page 201)

GetInterceptorList (see page 196)

TPacketInterceptFactory (see page 175)

Specifying a Connection Using Sockets

2.1.10.5.4 TStreamedConnection.Interpreter Property

Provides access to the helper object that marshals interface calls for the streamed connection.

Pascal

```
property Interpreter: TCustomDataBlockInterpreter;
```

C++

```
__property TCustomDataBlockInterpreter Interpreter;
```

Description

Streamed connection components use a data block interpreter internally to provide marshaling support.

See Also

Send (see page 199)

2.1.10.5.5 TStreamedConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection (see page 176), callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection (see page 209) can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 199)

Extending the Interface of the Application Server

2.1.11 TTransportThread Class

Class Hierarchy

Pascal

```
TTransportThread = class(TThread);
```

C++

```
class TTransportThread : public TThread;
```

File

SConnect

Description

This is class SConnect.TTransportThread.

2.1.11.1 TTransportThread Members

The following tables list the members exposed by TTransportThread.

Public Methods

	Name	Description
	Create (see page 204)	This is Create, a member of class TTransportThread.
	Destroy (see page 204)	Destroys the thread object and releases the memory allocated to it.
	Execute (see page 205)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Properties

	Name	Description
	Semaphore (see page 209)	This is Semaphore, a member of class TTransportThread.

2.1.11.2 TTransportThread.Create Constructor

Pascal

```
constructor Create(AHandle: THandle; Transport: ITransport); virtual;
```

C++

```
virtual __fastcall TTransportThread(THandle AHandle, ITransport Transport);
```

Description

This is Create, a member of class TTransportThread.

2.1.11.3 TTransportThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTransportThread();
```

Description

Do not call Destroy in an application. Instead, use Free to destroy a thread when FreeOnTerminate is False.

Destroy signals the thread to terminate and then waits for the thread to return before calling the inherited Destroy method.

See Also

Create

Terminate

WaitFor

FreeOnTerminate

System::TObject::Free (see page 3382)

2.1.11.4 TTransportThread Methods

The methods of the TTransportThread class are listed here.

Public Methods

	Name	Description
	Execute (see page 205)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

2.1.11.4.1 TTransportThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
```

```
  This example shows how to create a thread and start it
  running at a lower priority than the main execution thread.
  Set the thread's FreeOnTerminate property is True, so that
  there is no need to free the thread when it finishes. To
  create the extension choose the
  File | New | Other | Thread Object menu.
  */
  bool mythreadRunning;
  bool youthreadRunning;

  class TMyThread : public TThread
  {
  __published:    // IDE-managed Components
  private:    // User declarations
  protected:  // User declarations
      void __fastcall Execute();
  public:     // User declarations
      __fastcall TMyThread(bool suspended);
  };

  __fastcall TMyThread::TMyThread(bool suspended)
      : TThread(suspended)
  {
  }

  class TYouThread : public TThread
  {
  __published:    // IDE-managed Components
  private:    // User declarations
  protected:  // User declarations
      void __fastcall Execute();
  public:     // User declarations
      __fastcall TYouThread(bool suspended);
  };

  __fastcall TYouThread::TYouThread(bool suspended)
      : TThread(suspended)
  {
  }

  void __fastcall TMyThread::Execute()
  {
    Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
    for (int I = 0; I <= 10; I++)
    {
      Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
      Sleep(1000);
    }
    mythreadRunning = FALSE;
  }

  void __fastcall TYouThread::Execute()
  {
    Form1->Memo3->Lines->Add("Second low priority process has been running for this many
  seconds:");
    for (int I = 0; I <= 10; I++)
    {
      Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
      Sleep(1000);
    }
    youthreadRunning = FALSE;
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
```

```
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
```

```
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

2.1.11.5 TTransportThread Properties

The properties of the TTransportThread class are listed here.

Public Properties

	Name	Description
	Semaphore (see page 209)	This is Semaphore, a member of class TTransportThread.

2.1.11.5.1 TTransportThread.Semaphore Property

Pascal

```
property Semaphore: THandle;
```

C++

```
__property THandle Semaphore;
```

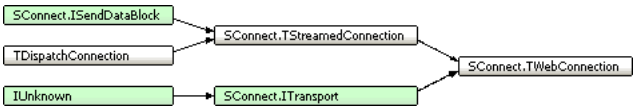
Description

This is Semaphore, a member of class TTransportThread.

2.1.12 TWebConnection Class

TWebConnection uses HTTP to manage the connection to an application server.

Class Hierarchy



Pascal

```
TWebConnection = class(TStreamedConnection, ITransport);
```

C++

```
class TWebConnection : public TStreamedConnection, public ITransport;
```

File

SConnect

Description

TWebConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Web connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TWebConnection establishes the initial connection between the client application and a remote application server using HTTP. To use TWebConnection, you must have the following:

Wininet.dll must be installed on the client system. Wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

The server system must have IIS version 4 or better or Netscape Enterprise version 3.6 or better.

HTTPsrvr.dll must be installed with the Web server to which TWebConnection forms a connection. HTTPsrvr.dll launches the application server and marshals all calls to the application server interface.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the Web connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection (see page 176) component instead. To connect to remote application servers using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TSocketConnection (see page 176)

TSoapConnection (see page 1128)

Extending the Interface of the Application Server

Specifying a Connection Using HTTP

2.1.12.1 TWebConnection Members

The following tables list the members exposed by TWebConnection.

Protected Methods

	Name	Description
	_AddRef (see page 194)	Increments an internal reference counter.
	_Release (see page 194)	Decrements an internal reference counter.
V	CreateTransport (see page 213)	Provides the backstop for a method that obtains an ITransport interface.
V	DoConnect (see page 213)	Connects to an application server.
V	DoDisconnect (see page 196)	Disconnects from an application server.
V	DoError (see page 196)	raises an exception for a specified exception object.
V	GetInterpreter (see page 214)	Returns the value of the Interpreter property.
	GetWaitEvent (see page 214)	Returns the handle for an event object that is managed by the object that implements ITransport.
V	InternalClose (see page 198)	Terminates a connection to the application server.
V	InternalOpen (see page 198)	Opens a connection to the application server.
	QueryInterface (see page 199)	Implements the IUnknown (see page 3828) QueryInterface method.
	Receive (see page 214)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 215)	Sends information retrieved from an IDataBlock interface to the application server.
V	SetConnected (see page 215)	Sets the Connected property.
	ThreadException (see page 199)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 200)	Handles callbacks from the application server.
	Transport_GetConnected (see page 216)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 216)	This is Transport_SetConnected, a member of class TWebConnection.
	WndProc (see page 200)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
R	Handle (see page 201)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 201)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 202)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 203)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 219)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Methods

	Name	Description
V	Create (see page 212)	Creates and initializes an instance of TWebConnection (see page 209).
V	Destroy (see page 212)	Destroys an instance of TWebConnection (see page 209).
	GetConnected (see page 234)	Returns the value of the Connected (see page 236) property.
V	GetInterceptorList (see page 196)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
V	GetServerList (see page 197)	Retrieves a list of registered data brokers from the remote server system.

Public Properties

	Name	Description
	Connected (see page 236)	Indicates whether the connection to the application server is established.

Published Properties

	Name	Description
	Agent (see page 217)	Provides the value of the User-Agent header for HTTP messages to the server.
	ObjectBroker (see page 217)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Password (see page 218)	Specifies the password that allows the client application to log on to the application server.

	Proxy (see page 218)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 219)	Lists the host names of locally registered servers that do not require a proxy.
	URL (see page 220)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 220)	Specifies the user name that allows the client application to log on to the application server.

2.1.12.2 TWebConnection.Create Constructor

Creates and initializes an instance of TWebConnection (see page 209).

Pascal

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

C++

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

Description

Call Create to instantiate a TWebConnection (see page 209) object at runtime. Web connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the Web connection component.

See Also

Create (see page 193)

Owner

2.1.12.3 TWebConnection.Destroy Destructor

Destroys an instance of TWebConnection (see page 209).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the Web connection is not nil before calling Destroy.

2.1.12.4 TWebConnection Methods

The methods of the TWebConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 213)	Provides the backstop for a method that obtains an ITransport interface.
	DoConnect (see page 213)	Connects to an application server.
	GetInterpreter (see page 214)	Returns the value of the Interpreter property.
	GetWaitEvent (see page 214)	Returns the handle for an event object that is managed by the object that implements ITransport.

	Receive (see page 214)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 215)	Sends information retrieved from an IDataBlock interface to the application server.
V	SetConnected (see page 215)	Sets the Connected property.
	Transport_GetConnected (see page 216)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 216)	This is Transport_SetConnected, a member of class TWebConnection.

2.1.12.4.1 TWebConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport interface.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport interface to connect to and communicate with the application server. As implemented in TStreamedConnection, CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection override CreateTransport to obtain an ITransport interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 199)

DoConnect (see page 195)

DoDisconnect (see page 196)

ThreadReceivedStream (see page 200)

2.1.12.4.2 TWebConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection override CreateTransport rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

CreateTransport (see page 195)

2.1.12.4.3 TWebConnection.GetInterpreter Method

Returns the value of the Interpreter property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; override;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter property. As implemented in TStreamedConnection, GetInterpreter instantiates a TDataBlockInterpreter object.

See Also

Interpreter (see page 203)

TDataBlockInterpreter (see page 163)

2.1.12.4.4 TWebConnection.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3074)

2.1.12.4.5 TWebConnection.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 227)

Send (see page 235)

2.1.12.4.6 TWebConnection.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 227)

Receive (see page 235)

2.1.12.4.7 TWebConnection.SetConnected Method

Sets the Connected property.

Pascal

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

C++

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

Description

SetConnected is the protected implementation of the Connected property. It establishes a connection to the Application server if Value is true, and disconnects from the application server if Value is false.

If Value is false, SetConnected generates a BeforeDisconnect event, closes the connection, and generates an AfterDisconnect event.

If Value is true, SetConnected performs the following actions:

1. If LoginPrompt is true, and the connection is not already open, SetConnected generates an OnGetUsername event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
2. It generates a BeforeConnect event.
3. It opens the connection.
4. It generates an AfterConnect event.
5. If LoginPrompt is true, and the connection opened successfully, SetConnected generates an OnLogin event.

See Also

Connected

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

OnGetUserName

OnLogin

2.1.12.4.8 TWebConnection.Transport_GetConnected Method

Pascal

```
function Transport_GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean Transport_GetConnected();
```

Description

This is Transport_GetConnected, a member of class TWebConnection.

2.1.12.4.9 TWebConnection.Transport_SetConnected Method

Pascal

```
procedure Transport_SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall Transport_SetConnected(Boolean Value);
```

Description

This is Transport_SetConnected, a member of class TWebConnection.

2.1.12.5 TWebConnection Properties

The properties of the TWebConnection class are listed here.

Protected Properties

	Name	Description
	SupportCallbacks (see page 219)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Published Properties

	Name	Description
	Agent (see page 217)	Provides the value of the User-Agent header for HTTP messages to the server.
	ObjectBroker (see page 217)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Password (see page 218)	Specifies the password that allows the client application to log on to the application server.
	Proxy (see page 218)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 219)	Lists the host names of locally registered servers that do not require a proxy.
	URL (see page 220)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 220)	Specifies the user name that allows the client application to log on to the application server.

2.1.12.5.1 TWebConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply HTTP messages generated by the Web connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title

2.1.12.5.2 TWebConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker;
```

C++

```
__property ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect

ServerName

ServerGUID

TSimpleObjectBroker

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

2.1.12.5.3 TWebConnection.Password Property

Specifies the password that allows the client application to log on to the application server.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When the Web connection component connects to the application server, it supplies the values of the UserName (see page 220) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 220)

Specifying a Connection Using HTTP

2.1.12.5.4 TWebConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the Web connection component can't resolve the value of Host locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the

intended server machine.

See Also

ProxyByPass (see page 219)

2.1.12.5.5 TWebConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the Web connection component can make a direct connection. When the Host property specifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 218) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and bypasses the proxy server if Host refers to a registered local server.

See Also

Proxy (see page 218)

2.1.12.5.6 TWebConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 199)

Extending the Interface of the Application Server

2.1.12.5.7 TWebConnection.URL Property

Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the Web connection component uses to locate httpsrvr.dll. This DLL, which ships in the BIN directory, is deployed on the same machine as the Web server so that it can accept HTTP request messages and convert them into COM calls that communicate with the application server.

URL includes the protocol (http or https), host name, and scriptname for httpsrvr.dll. Typically, it has the form

`http://DataHost.org/scripts/httpsrvr.dll`

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

Specifying a Connection Using HTTP

2.1.12.5.8 TWebConnection.UserName Property

Specifies the user name that allows the client application to log on to the application server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When the Web connection component connects to the application server on a local host or to the proxy server, it supplies the values of the UserName and Password (see page 218) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

See Also

Password (see page 218)

Specifying a Connection Using HTTP

2.2 Interfaces

The following table lists interfaces in this documentation.

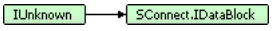
Interfaces

	Name	Description
	IDataBlock (see page 221)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 229)	IDataIntercept is the interface used by TSocketConnection (see page 176) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 231)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 233)	ITransport is the interface used by TStreamedConnection (see page 191) to connect to, read from, and write to a remote application server.

2.2.1 IDataBlock Interface

IDataBlock is the interface used by data block interpreters to access memory buffers.

Class Hierarchy



Pascal

```
IDataBlock = interface(IUnknown);
```

C++

```
__interface IDataBlock : public IUnknown;
```

File

SConnect

Description

Data block interpreters uses IDataBlock when marshaling COM interface calls. Streamed connection components send data obtained from IDataBlock to an application server once the data block interpreter has filled its memory by marshaling interface calls. They also read data from the application server into IDataBlock and pass it to a data block interpreter for marshaling.

IDataBlock is an interface, and as such can't be directly instantiated. Instead, objects that implement the IDataBlock properties and methods include IDataBlock in their class definition (Delphi) or be derived from IDataBlock (C++). Such objects can be cast to an IDataBlock so that they can be used by data block interpreters.

Note: C++ method declarations that use IDataBlock use the _di_IDataBlock type instead. This is a DelphiInterface (see page 3263) wrapper around the IDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDataBlock >  _di_IDataBlock;
```

See Also

TCustomDataBlockInterpreter (see page 148)

TStreamedConnection (see page 191)

2.2.1.1 IDataBlock Members

The following tables list the members exposed by IDataBlock.

Public Methods

	Name	Description
	Clear (see page 222)	Removes all data from Memory (see page 227) except for the reserved bytes.
	GetBytesReserved (see page 223)	Returns the value of the BytesReserved (see page 227) property.
	GetMemory (see page 223)	Returns the value of the Memory (see page 227) property.
	GetSignature (see page 223)	Returns the value of the Signature (see page 228) property.
	GetSize (see page 224)	Returns the value of the Size (see page 228) property.
	GetStream (see page 224)	Returns the value of the Stream (see page 229) property.
	IgnoreStream (see page 224)	Informs the data block when its stream is invalidated.
	InitData (see page 225)	Adds data to the data block.
	Read (see page 225)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 226)	Sets the value of the Signature (see page 228) property.
	SetSize (see page 226)	Sets the value of the Size (see page 228) property.
	Write (see page 226)	Writes a specified number of bytes from a buffer.

Public Properties

	Name	Description
R	BytesReserved (see page 227)	Indicates the number of bytes from Memory (see page 227) that are reserved for internal use.
R	Memory (see page 227)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 228)	Indicates the contents of the data block.
	Size (see page 228)	Indicates the size of the data in the memory buffer.
R	Stream (see page 229)	Provides access to an internal stream that contains the data block's memory.

2.2.1.2 IDataBlock Methods

The methods of the IDataBlock class are listed here.

Public Methods

	Name	Description
	Clear (see page 222)	Removes all data from Memory (see page 227) except for the reserved bytes.
	GetBytesReserved (see page 223)	Returns the value of the BytesReserved (see page 227) property.
	GetMemory (see page 223)	Returns the value of the Memory (see page 227) property.
	GetSignature (see page 223)	Returns the value of the Signature (see page 228) property.
	GetSize (see page 224)	Returns the value of the Size (see page 228) property.
	GetStream (see page 224)	Returns the value of the Stream (see page 229) property.
	IgnoreStream (see page 224)	Informs the data block when its stream is invalidated.
	InitData (see page 225)	Adds data to the data block.
	Read (see page 225)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 226)	Sets the value of the Signature (see page 228) property.
	SetSize (see page 226)	Sets the value of the Size (see page 228) property.
	Write (see page 226)	Writes a specified number of bytes from a buffer.

2.2.1.2.1 IDataBlock.Clear Method

Removes all data from Memory (see page 227) except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory (see page 227) or when the information written to Memory (see page 227) has been used. Clear reclaims the resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 227)

Memory (see page 227)

2.2.1.2.2 IDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved (see page 227) property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved (see page 227) property.

See Also

BytesReserved (see page 227)

2.2.1.2.3 IDataBlock.GetMemory Method

Returns the value of the Memory (see page 227) property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory (see page 227) property.

See Also

Memory (see page 227)

2.2.1.2.4 IDataBlock.GetSignature Method

Returns the value of the Signature (see page 228) property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature (see page 228) property.

See Also

Signature (see page 228)

2.2.1.2.5 IDataBlock.GetSize Method

Returns the value of the Size (see page 228) property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size (see page 228) property.

See Also

Size (see page 228)

2.2.1.2.6 IDataBlock.GetStream Method

Returns the value of the Stream (see page 229) property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream (see page 229) property.

See Also

Stream (see page 229)

2.2.1.2.7 IDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream (see page 229). After a call to

IgnoreStream, the data block will not free the memory for its internal Stream (see page 229) component.

See Also

Stream (see page 229)

2.2.1.2.8 IDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 228)

2.2.1.2.9 IDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory (see page 227) at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 228) - BytesReserved (see page 227).

See Also

Write (see page 226)

Memory (see page 227)

Size (see page 228)

BytesReserved (see page 227)

2.2.1.2.10 IDataBlock.SetSignature Method

Sets the value of the Signature (see page 228) property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature (see page 228) property.

See Also

Signature (see page 228)

2.2.1.2.11 IDataBlock.SetSize Method

Sets the value of the Size (see page 228) property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size (see page 228) property.

See Also

Size (see page 228)

2.2.1.2.12 IDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory (see page 227). Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory (see page 227) at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 228) - BytesReserved (see page 227).

See Also

Read (see page 225)

Memory (see page 227)

Size (see page 228)

BytesReserved (see page 227)

2.2.1.3 IDataBlock Properties

The properties of the IDataBlock class are listed here.

Public Properties

	Name	Description
R	BytesReserved (see page 227)	Indicates the number of bytes from Memory (see page 227) that are reserved for internal use.
R	Memory (see page 227)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 228)	Indicates the contents of the data block.
	Size (see page 228)	Indicates the size of the data in the memory buffer.
R	Stream (see page 229)	Provides access to an internal stream that contains the data block's memory.

2.2.1.3.1 IDataBlock.BytesReserved Property

Indicates the number of bytes from Memory (see page 227) that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory (see page 227) can't be used for reading and writing. When interpreting the value of the Size (see page 228) property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 227)

Size (see page 228)

2.2.1.3.2 IDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved (see page 227)) for internal information such as the value of Signature (see page 228),

Note: When implementing the Receive and Send methods of the ITransport (see page 233) interface, component writers should read directly from and write directly to Memory, rather than using the Read (see page 225) and Write (see page 226)

methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 228)

ITransport (see page 233)

BytesReserved (see page 227)

Signature (see page 228)

Stream (see page 229)

2.2.1.3.3 IDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write (see page 226) Signature values to indicate what type of data is written to the data block. Read (see page 225) Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved (see page 227) bytes of Memory (see page 227).

See Also

BytesReserved (see page 227)

Memory (see page 227)

2.2.1.3.4 IDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory (see page 227), excluding the reserved bytes. Set Size to ensure that Memory (see page 227) is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 227)

Memory (see page 227)

Signature (see page 228)

2.2.1.3.5 IDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 227)

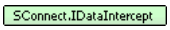
IgnoreStream (see page 224)

TMemoryStream

2.2.2 IDataIntercept Interface

IDataIntercept is the interface used by TSocketConnection (see page 176) components to transform data that is communicated over the connection.

Class Hierarchy



Pascal

```
IDataIntercept = interface;
```

C++

```
__interface IDataIntercept;
```

File

SConnect

Description

IDataIntercept is the interface implemented by data packet interceptors. A data packet interceptor is a COM object that implements IDataIntercept to transform data that is accessed through an IDataBlock (see page 221) interface. TSocketConnection (see page 176) can use a data packet interceptor before messages are sent and after responses are received.

Developers can create a data packet interceptor to encrypt and decrypt data or to compress and decompress messages. The transformation must leave the reserved bytes of the data blocks unchanged or the socket connection component won't be able to communicate properly.

Note: When creating a data packet interceptor in Delphi, use TPacketInterceptFactory (see page 175) as its class factory.

Once you have created and registered a data packet interceptor, Set the InterceptName or InterceptGUID property of a socket connection component to identify the data packet interceptor. This tells the socket connection to instantiate the interceptor and use it to transform messages before they are sent and after they are received. On the server side, right click on the Tray Icon for

scktsrvr.exe and assign the interceptor using the properties dialog.

See Also

IDataBlock (see page 221)

TStreamedConnection (see page 191)

TStreamedConnection (see page 191)

TPacketInterceptFactory (see page 175)

Specifying a Connection Using Sockets

2.2.2.1 IDataIntercept Members

The following tables list the members exposed by IDataIntercept.

Public Methods

	Name	Description
	DataIn (see page 230)	Undoes the transformation applied by the DataOut (see page 231) method.
	DataOut (see page 231)	Returns the value of the BytesReserved property.

2.2.2.2 IDataIntercept Methods

The methods of the IDataIntercept class are listed here.

Public Methods

	Name	Description
	DataIn (see page 230)	Undoes the transformation applied by the DataOut (see page 231) method.
	DataOut (see page 231)	Returns the value of the BytesReserved property.

2.2.2.2.1 IDataIntercept.DataIn Method

Undoes the transformation applied by the DataOut (see page 231) method.

Pascal

```
procedure DataIn(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataIn(const IDataBlock Data);
```

Description

Call DataIn to undo any transformations that were applied to information received by the connection. For example, use this method to decrypt or decompress data.

The Data parameter is an IDataBlock (see page 221) interface to the data that was received by the socket connection component. Use the properties and methods of IDataBlock (see page 221) to reverse the transformation that was applied by the DataOut (see page 231) method.

Warning: Do not transform the reserved bytes in the data block.

See Also

DataOut (see page 231)

BytesReserved (see page 227)

2.2.2.2.2 IDataIntercept.DataOut Method

Returns the value of the BytesReserved property.

Pascal

```
procedure DataOut(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataOut(const IDataBlock Data);
```

Description

Call DataOut to transform data before it is sent using a connection component. For example, use this method to encrypt or compress data.

The Data parameter is an IDataBlock (see page 221) interface to the data that is about to be sent by the socket connection component. Use the properties and methods of IDataBlock (see page 221) to transform the data.

Warning: Do not transform the reserved bytes in the data block.

See Also

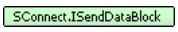
DataIn (see page 230)

BytesReserved (see page 227)

2.2.3 ISendDataBlock Interface

ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.

Class Hierarchy



Pascal

```
ISendDataBlock = interface;
```

C++

```
__interface ISendDataBlock;
```

File

SConnect

Description

Streamed connection components implement ISendDataBlock to allow a data block interpreter (which handles marshaling of COM interface calls) to send interface calls to the application server.

ISendDataBlock is an interface, and as such can't be directly instantiated. Instead, streamed connection components (which implement ISendDataBlock) can be cast to an ISendDataBlock (Delphi) or have a conversion operator to obtain this interface (C++).

Note: TDataBlockInterpreter (see page 163) uses _di_ISendDataBlock in place of ISendDataBlock. This is a DelphiInterface (see page 3263) wrapper around the ISendDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ISendDataBlock > _di_ISendDataBlock;
```

See Also

TStreamedConnection (see page 191)

TCustomDataBlockInterpreter (see page 148)

IDataBlock (see page 221)

2.2.3.1 ISendDataBlock Members

The following tables list the members exposed by ISendDataBlock.

Public Methods

	Name	Description
	Send (see page 232)	Sends information retrieved from an IDataBlock (see page 221) interface to the application server.

2.2.3.2 ISendDataBlock Methods

The methods of the ISendDataBlock class are listed here.

Public Methods

	Name	Description
	Send (see page 232)	Sends information retrieved from an IDataBlock (see page 221) interface to the application server.

2.2.3.2.1 ISendDataBlock.Send Method

Sends information retrieved from an IDataBlock (see page 221) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

ISendDataBlock (see page 231) dispatches information to an application server. The information must be available as the Memory property of an IDataBlock (see page 221) interface.

The WaitForResult parameter indicates whether the caller requires a response when the information is sent. If WaitForResult is true, Send does not return until a response is received from the application server. This response is returned as an IDataBlock (see page 221) interface (_di_IDataBlock). If WaitForResult is false, Send may still return a response, but only if the application server responds immediately.

See Also

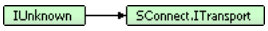
Memory (see page 227)

Send (see page 199)

2.2.4 ITransport Interface

ITransport is the interface used by TStreamedConnection (see page 191) to connect to, read from, and write to a remote application server.

Class Hierarchy



Pascal

```
ITransport = interface(IUnknown);
```

C++

```
__interface ITransport : public IUnknown;
```

File

SConnect

Description

Streamed connection components use ITransport to connect to an application server and to send or receive data over that connection. ITransport is responsible only for managing the communication protocols. It does not provide any support for marshaling or storing information.

ITransport is an interface, and as such can't be directly instantiated. Instead, objects that implement the ITransport properties and methods include ITransport in their class definition (Delphi) or are derived from ITransport (C++). Such objects can be cast to an ITransport so that they can be used by streamed connection components.

Note: C++ method declarations that use ITransport use the _di_ITransport type instead. This is a DelphiInterface (see page 3263) wrapper around the ITransport interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ITransport> _di_ITransport;
```

See Also

TStreamedConnection (see page 191)

2.2.4.1 ITransport Members

The following tables list the members exposed by ITransport.

Public Methods

	Name	Description
	GetConnected (see page 234)	Returns the value of the Connected (see page 236) property.
	GetWaitEvent (see page 234)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 233).
	Receive (see page 235)	Returns an IDataBlock (see page 221) interface that can be used to read information received from the application server.
	Send (see page 235)	Sends information retrieved from an IDataBlock (see page 221) interface to the application server.
	SetConnected (see page 235)	Sets the value of the Connected (see page 236) property.

Public Properties

	Name	Description
	Connected (see page 236)	Indicates whether the connection to the application server is established.

2.2.4.2 ITransport Methods

The methods of the ITransport class are listed here.

Public Methods

	Name	Description
	GetConnected (see page 234)	Returns the value of the Connected (see page 236) property.
	GetWaitEvent (see page 234)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 233).
	Receive (see page 235)	Returns an IDataBlock (see page 221) interface that can be used to read information received from the application server.
	Send (see page 235)	Sends information retrieved from an IDataBlock (see page 221) interface to the application server.
	SetConnected (see page 235)	Sets the value of the Connected (see page 236) property.

2.2.4.2.1 ITransport.GetConnected Method

Returns the value of the Connected (see page 236) property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected (see page 236) property.

See Also

Connected (see page 236)

2.2.4.2.2 ITransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport (see page 233).

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport (see page 233).

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport (see page 233),

create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3074)

2.2.4.2.3 ITransport.Receive Method

Returns an IDataBlock (see page 221) interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock (see page 221) interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport (see page 233). Its value is the return value of the Send (see page 235) method.

See Also

Memory (see page 227)

Send (see page 235)

2.2.4.2.4 ITransport.Send Method

Sends information retrieved from an IDataBlock (see page 221) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive (see page 235) method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 227)

Receive (see page 235)

2.2.4.2.5 ITransport.SetConnected Method

Sets the value of the Connected (see page 236) property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected (see page 236) property.

2.2.4.3 ITransport Properties

The properties of the ITransport class are listed here.

Public Properties

	Name	Description
	Connected (see page 236)	Indicates whether the connection to the application server is established.

2.2.4.3.1 ITransport.Connected Property

Indicates whether the connection to the application server is established.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to connect to the application server. Set Connected to false to close a connection.

Read (see page 3727) the Connected property to verify that the connection is open before reading from or writing to the application server.

See Also

Receive (see page 235)

Send (see page 235)

2.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetPacketInterceptorList (see page 237)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 237)	This is function SConnect.LoadWinSock2.

2.3.1 SConnect.GetPacketInterceptorList Function

Pascal

```
procedure GetPacketInterceptorList(List: TStringList);
```

C++

```
GetPacketInterceptorList(TStringList List);
```

File

SConnect

Description

This is function SConnect.GetPacketInterceptorList.

2.3.2 SConnect.LoadWinSock2 Function

Pascal

```
function LoadWinSock2: Boolean;
```

C++

```
Boolean LoadWinSock2();
```

File

SConnect

Description

This is function SConnect.LoadWinSock2.

2.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TVarFlag (see page 238)	This is record SConnect.TVarFlag.

2.4.1 SConnect.TVarFlag Enumeration

Pascal

```
TVarFlag = (
  vfByRef,
  vfVariant
);
```

C++

```
enum TVarFlag {
  vfByRef,
  vfVariant
};
```

File

SConnect

Description

This is record SConnect.TVarFlag.

2.5 Types

The following table lists types in this documentation.

Types

Name	Description
PIntArray (see page 239)	This is type SConnect.PIntArray.
PVariantArray (see page 239)	This is type SConnect.PVariantArray.
TIntArray (see page 240)	This is type SConnect.TIntArray.
TVarFlags (see page 240)	This is type SConnect.TVarFlags.
TVariantArray (see page 240)	This is type SConnect.TVariantArray.

2.5.1 SConnect.PIntArray Type

Pascal

```
PIntArray = ^TIntArray;
```

C++

```
^TIntArray PIntArray;
```

File

SConnect

Description

This is type SConnect.PIntArray.

2.5.2 SConnect.PVariantArray Type

Pascal

```
PVariantArray = ^TVariantArray;
```

C++

```
^TVariantArray PVariantArray;
```

File

SConnect

Description

This is type SConnect.PVariantArray.

2.5.3 SConnect.TIntArray Type

Pascal

```
TIntArray = array[0..0] of Integer;
```

C++

```
array[0..0] of Integer TIntArray;
```

File

SConnect

Description

This is type SConnect.TIntArray.

2.5.4 SConnect.TVarFlags Type

Pascal

```
TVarFlags = set of TVarFlag;
```

C++

```
TVarFlag TVarFlags;
```

File

SConnect

Description

This is type SConnect.TVarFlags.

2.5.5 SConnect.TVariantArray Type

Pascal

```
TVariantArray = array[0..0] of OleVariant;
```

C++

```
array[0..0] of OleVariant TVariantArray;
```

File

SConnect

Description

This is type SConnect.TVariantArray.

2.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WSACloseEvent (see page 241)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 241)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 242)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 242)	This is variable SConnect.WSAResetEvent.

2.6.1 SConnect.WSACloseEvent Variable

Pascal

```
WSACloseEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSACloseEvent;
```

File

SConnect

Description

This is variable SConnect.WSACloseEvent.

2.6.2 SConnect.WSACreateEvent Variable

Pascal

```
WSACreateEvent: function: THandle stdcall;
```

C++

```
function: THandle stdcall WSACreateEvent;
```

File

SConnect

Description

This is variable SConnect.WSACreateEvent.

2.6.3 SConnect.WSAEventSelect Variable

Pascal

```
WSAEventSelect: function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer
stdcall;
```

C++

```
function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer stdcall
WSAEventSelect;
```

File

SConnect

Description

This is variable SConnect.WSAEventSelect.

2.6.4 SConnect.WSAResetEvent Variable

Pascal

```
WSAResetEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSAResetEvent;
```

File

SConnect

Description

This is variable SConnect.WSAResetEvent.

2.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
asCreateObject (see page 243)	This is constant SConnect.asCreateObject.
asError (see page 243)	This is constant SConnect.asError.
asFreeObject (see page 244)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 244)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 244)	This is constant SConnect.asGetGUID.
asGetID (see page 245)	This is constant SConnect.asGetID.
asGetServers (see page 245)	This is constant SConnect.asGetServers.
asInvoke (see page 245)	This is constant SConnect.asInvoke.
asMask (see page 246)	This is constant SConnect.asMask.
asSoapCommand (see page 246)	This is constant SConnect.asSoapCommand.
CallSig (see page 246)	This is constant SConnect.CallSig.
ResultSig (see page 247)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 247)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 247)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 248)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 248)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 248)	This is constant SConnect.THREAD_SENDSTREAM.

2.7.1 SConnect.asCreateObject Constant

Pascal

```
asCreateObject = $04;
```

C++

```
asCreateObject = 0x04;
```

File

SConnect

Description

This is constant SConnect.asCreateObject.

2.7.2 SConnect.asError Constant

Pascal

```
asError = $01;
```

C++

```
asError = 0x01;
```

File

SConnect

Description

This is constant SConnect.asError.

2.7.3 SConnect.asFreeObject Constant

Pascal

```
asFreeObject = $05;
```

C++

```
asFreeObject = 0x05;
```

File

SConnect

Description

This is constant SConnect.asFreeObject.

2.7.4 SConnect.asGetAppServers Constant

Pascal

```
asGetAppServers = $12;
```

C++

```
asGetAppServers = 0x12;
```

File

SConnect

Description

This is constant SConnect.asGetAppServers.

2.7.5 SConnect.asGetGUID Constant

Pascal

```
asGetGUID = $11;
```

C++

```
asGetGUID = 0x11;
```

File

SConnect

Description

This is constant SConnect.asGetGUID.

2.7.6 SConnect.asGetID Constant

Pascal

```
asGetID = $03;
```

C++

```
asGetID = 0x03;
```

File

SConnect

Description

This is constant SConnect.asGetID.

2.7.7 SConnect.asGetServers Constant

Pascal

```
asGetServers = $10;
```

C++

```
asGetServers = 0x10;
```

File

SConnect

Description

This is constant SConnect.asGetServers.

2.7.8 SConnect.asInvoke Constant

Pascal

```
asInvoke = $02;
```

C++

```
asInvoke = 0x02;
```

File

SConnect

Description

This is constant SConnect.asInvoke.

2.7.9 SConnect.asMask Constant

Pascal

```
asMask = $FF;
```

C++

```
asMask = 0xFF;
```

File

SConnect

Description

This is constant SConnect.asMask.

2.7.10 SConnect.asSoapCommand Constant

Pascal

```
asSoapCommand = $14;
```

C++

```
asSoapCommand = 0x14;
```

File

SConnect

Description

This is constant SConnect.asSoapCommand.

2.7.11 SConnect.CallSig Constant

Pascal

```
CallSig = $DA00;
```

C++

```
CallSig = 0xDA00;
```

File

SConnect

Description

This is constant SConnect.CallSig.

2.7.12 SConnect.ResultSig Constant

Pascal

```
ResultSig = $DB00;
```

C++

```
ResultSig = 0xDB00;
```

File

SConnect

Description

This is constant SConnect.ResultSig.

2.7.13 SConnect.THREAD_EXCEPTION Constant

Pascal

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

C++

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_EXCEPTION.

2.7.14 SConnect.THREAD_RECEIVEDSTREAM Constant

Pascal

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

C++

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_RECEIVEDSTREAM.

2.7.15 SConnect.THREAD_REPLACETRANSPORT Constant

Pascal

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

C++

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_REPLACETRANSPORT.

2.7.16 SConnect.THREAD_SENDNOTIFY Constant

Pascal

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

C++

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDNOTIFY.

2.7.17 SConnect.THREAD_SENDSTREAM Constant

Pascal

```
THREAD_SENDSTREAM = WM_USER + 1;
```

C++

```
THREAD_SENDSTREAM = WM_USER + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDSTREAM.

3 ScreenTips Namespace

This is namespace ScreenTips.

Topics

Name	Description
Classes (see page 251)	The following table lists classes in this documentation.
Interfaces (see page 315)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 317)	The following table lists structs, records, enums in this documentation.
Types (see page 320)	The following table lists types in this documentation.
Constants (see page 323)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAllocatedActionsException (see page 251)	This is class ScreenTips.EAllocatedActionsException.
	TAllocatedActions (see page 251)	This is class ScreenTips.TAllocatedActions.
	TAssociateItem (see page 255)	This is class ScreenTips.TAssociateItem.
	TAssociateItemList (see page 256)	This is class ScreenTips.TAssociateItemList.
	TCustomScreenTipsManager (see page 259)	This is class ScreenTips.TCustomScreenTipsManager.
	TCustomScreenTipsPopup (see page 270)	This is class ScreenTips.TCustomScreenTipsPopup.
	TDisplayOffset (see page 278)	This is class ScreenTips.TDisplayOffset.
	TScreenTipBorderStyle (see page 280)	This is class ScreenTips.TScreenTipBorderStyle.
	TScreenTipItem (see page 282)	This is class ScreenTips.TScreenTipItem.
	TScreenTipsCollection (see page 288)	This is class ScreenTips.TScreenTipsCollection.
	TScreenTipsManager (see page 291)	This is class ScreenTips.TScreenTipsManager.
	TScreenTipsPopup (see page 297)	This is class ScreenTips.TScreenTipsPopup.
	TScreenTipsWindow (see page 305)	This is class ScreenTips.TScreenTipsWindow.

Constants

Name	Description
cDefaultBorderColor (see page 323)	This is constant ScreenTips.cDefaultBorderColor.
cDefaultEndColor (see page 323)	This is constant ScreenTips.cDefaultEndColor.
cDefaultStartColor (see page 324)	This is constant ScreenTips.cDefaultStartColor.
cScreenTipHeaderSpacing (see page 324)	This is constant ScreenTips.cScreenTipHeaderSpacing.
cScreenTipMinimumHeight (see page 324)	This is constant ScreenTips.cScreenTipMinimumHeight.
cScreenTipTextAndImageWidth (see page 325)	This is constant ScreenTips.cScreenTipTextAndImageWidth.
cScreenTipTextOnlyWidth (see page 325)	This is constant ScreenTips.cScreenTipTextOnlyWidth.

Interfaces

	Name	Description
	IScreenTipItem (see page 315)	This is class ScreenTips.IScreenTipItem.

Structs, Records, Enums

	Name	Description
	TDescriptionType (see page 317)	This is record ScreenTips.TDescriptionType.
	TImageAlign (see page 317)	This is record ScreenTips.TImageAlign.
	TItemImageAlign (see page 318)	This is record ScreenTips.TItemImageAlign.
	TScreenTipOption (see page 318)	This is record ScreenTips.TScreenTipOption.
	TScreenTipPopupType (see page 319)	This is record ScreenTips.TScreenTipPopupType.
	TTipBorderStyle (see page 319)	This is record ScreenTips.TTipBorderStyle.

Types

Name	Description
TCreateScreenTipEvent (see page 320)	This is type ScreenTips.TCreateScreenTipEvent.
TDuplicateCreateErrorEvent (see page 320)	This is type ScreenTips.TDuplicateCreateErrorEvent.
TScreenTipOptions (see page 321)	This is type ScreenTips.TScreenTipOptions.
TScreenTipsWindowClass (see page 321)	This is type ScreenTips.TScreenTipsWindowClass.
TScreenTipWinClass (see page 321)	This is type ScreenTips.TScreenTipWinClass.
TShowScreenTipEvent (see page 322)	This is type ScreenTips.TShowScreenTipEvent.

3.1 Classes

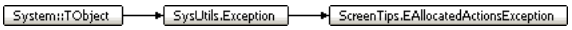
The following table lists classes in this documentation.

Classes

	Name	Description
	EAllocatedActionsException (see page 251)	This is class ScreenTips.EAllocatedActionsException.
	TAllocatedActions (see page 251)	This is class ScreenTips.TAllocatedActions.
	TAssociateItem (see page 255)	This is class ScreenTips.TAssociateItem.
	TAssociateItemList (see page 256)	This is class ScreenTips.TAssociateItemList.
	TCustomScreenTipsManager (see page 259)	This is class ScreenTips.TCustomScreenTipsManager.
	TCustomScreenTipsPopup (see page 270)	This is class ScreenTips.TCustomScreenTipsPopup.
	TDisplayOffset (see page 278)	This is class ScreenTips.TDisplayOffset.
	TScreenTipBorderStyle (see page 280)	This is class ScreenTips.TScreenTipBorderStyle.
	TScreenTipItem (see page 282)	This is class ScreenTips.TScreenTipItem.
	TScreenTipsCollection (see page 288)	This is class ScreenTips.TScreenTipsCollection.
	TScreenTipsManager (see page 291)	This is class ScreenTips.TScreenTipsManager.
	TScreenTipsPopup (see page 297)	This is class ScreenTips.TScreenTipsPopup.
	TScreenTipsWindow (see page 305)	This is class ScreenTips.TScreenTipsWindow.

3.1.1 EAllocatedActionsException Class

Class Hierarchy



Pascal

```
EAllocatedActionsException = class(Exception);
```

C++

```
class EAllocatedActionsException : public Exception;
```

File

ScreenTips

Description

This is class ScreenTips.EAllocatedActionsException.

3.1.2 TAllocatedActions Class

Class Hierarchy

Pascal

```
TAllocatedActions = class;
```

C++

```
class TAllocatedActions;
```

File

ScreenTips

Description

This is class ScreenTips.TAllocatedActions.

3.1.2.1 TAllocatedActions Members

The following tables list the members exposed by TAllocatedActions.

Public Methods

	Name	Description
	AddAction (see page 253)	This is AddAction, a member of class TAllocatedActions.
	Clear (see page 253)	This is Clear, a member of class TAllocatedActions.
	Create (see page 252)	This is Create, a member of class TAllocatedActions.
	Delete (see page 253)	This is Delete, a member of class TAllocatedActions.
	Destroy (see page 252)	This is Destroy, a member of class TAllocatedActions.
	Exists (see page 253)	This is Exists, a member of class TAllocatedActions.
	Find (see page 254)	This is Find, a member of class TAllocatedActions.
	GetItem (see page 254)	This is GetItem, a member of class TAllocatedActions.

Public Properties

	Name	Description
	Count (see page 254)	This is Count, a member of class TAllocatedActions.

3.1.2.2 TAllocatedActions.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TAllocatedActions();
```

Description

This is Create, a member of class TAllocatedActions.

3.1.2.3 TAllocatedActions.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAllocatedActions();
```

Description

This is Destroy, a member of class TAllocatedActions.

3.1.2.4 TAllocatedActions Methods

The methods of the TAllocatedActions class are listed here.

Public Methods

	Name	Description
	AddAction (see page 253)	This is AddAction, a member of class TAllocatedActions.
	Clear (see page 253)	This is Clear, a member of class TAllocatedActions.
	Delete (see page 253)	This is Delete, a member of class TAllocatedActions.
	Exists (see page 253)	This is Exists, a member of class TAllocatedActions.
	Find (see page 254)	This is Find, a member of class TAllocatedActions.
	GetItem (see page 254)	This is GetItem, a member of class TAllocatedActions.

3.1.2.4.1 TAllocatedActions.AddAction Method

Pascal

```
procedure AddAction(const aName: string; aScreenTipItem: TObject);
```

C++

```
__fastcall AddAction(const UnicodeString aName, TObject * aScreenTipItem);
```

Description

This is AddAction, a member of class TAllocatedActions.

3.1.2.4.2 TAllocatedActions.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TAllocatedActions.

3.1.2.4.3 TAllocatedActions.Delete Method

Pascal

```
function Delete(const aName: string; out aIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean Delete(const UnicodeString aName, int aIndex);
```

Description

This is Delete, a member of class TAllocatedActions.

3.1.2.4.4 TAllocatedActions.Exists Method

Pascal

```
function Exists(const aName: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const UnicodeString aName);
```

Description

This is Exists, a member of class TAllocatedActions.

3.1.2.4.5 TAllocatedActions.Find Method

Pascal

```
function Find(const aName: string; out aIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(const UnicodeString aName, int aIndex);
```

Description

This is Find, a member of class TAllocatedActions.

3.1.2.4.6 TAllocatedActions.GetItem Method

Pascal

```
function GetItem(const aName: string): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetItem(const UnicodeString aName);
```

Description

This is GetItem, a member of class TAllocatedActions.

3.1.2.5 TAllocatedActions Properties

The properties of the TAllocatedActions class are listed here.

Public Properties

	Name	Description
	Count (see page 254)	This is Count, a member of class TAllocatedActions.

3.1.2.5.1 TAllocatedActions.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAllocatedActions.

3.1.3 TAssociateItem Class

Class Hierarchy

Pascal

```
TAssociateItem = class;
```

C++

```
class TAssociateItem;
```

File

ScreenTips

Description

This is class ScreenTips.TAssociateItem.

3.1.3.1 TAssociateItem Members

The following tables list the members exposed by TAssociateItem.

Public Methods

	Name	Description
	Create (see page 255)	This is Create, a member of class TAssociateItem.
	Destroy (see page 255)	This is Destroy, a member of class TAssociateItem.

Public Properties

	Name	Description
	Associate (see page 256)	This is Associate, a member of class TAssociateItem.
	PopupTip (see page 256)	This is PopupTip, a member of class TAssociateItem.

3.1.3.2 TAssociateItem.Create Constructor

Pascal

```
constructor Create(APopupTip: TControl; AAssociate: TControl);
```

C++

```
__fastcall TAssociateItem(TControl * APopupTip, TControl * AAssociate);
```

Description

This is Create, a member of class TAssociateItem.

3.1.3.3 TAssociateItem.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAssociateItem();
```

Description

This is Destroy, a member of class TAssociateItem.

3.1.3.4 TAssociateItem Properties

The properties of the TAssociateItem class are listed here.

Public Properties

	Name	Description
	Associate (see page 256)	This is Associate, a member of class TAssociateItem.
	PopupTip (see page 256)	This is PopupTip, a member of class TAssociateItem.

3.1.3.4.1 TAssociateItem.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TAssociateItem.

3.1.3.4.2 TAssociateItem.PopupTip Property

Pascal

```
property PopupTip: TControl;
```

C++

```
__property TControl * PopupTip;
```

Description

This is PopupTip, a member of class TAssociateItem.

3.1.4 TAssociateItemList Class

Class Hierarchy

Pascal

```
TAssociateItemList = class;
```

C++

```
class TAssociateItemList;
```

File

ScreenTips

Description

This is class ScreenTips.TAssociateItemList.

3.1.4.1 TAssociateItemList Members

The following tables list the members exposed by TAssociateItemList.

Public Methods

	Name	Description
	Add (see page 258)	This is Add, a member of class TAssociateItemList.
	Create (see page 257)	This is Create, a member of class TAssociateItemList.
	Destroy (see page 257)	This is Destroy, a member of class TAssociateItemList.
	Find (see page 258)	This is Find, a member of class TAssociateItemList.
	Get (see page 258)	This is Get, a member of class TAssociateItemList.
	Remove (see page 258)	This is Remove, a member of class TAssociateItemList.

3.1.4.2 TAssociateItemList.Create Constructor

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TAssociateItemList();
```

Description

This is Create, a member of class TAssociateItemList.

3.1.4.3 TAssociateItemList.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAssociateItemList();
```

Description

This is Destroy, a member of class TAssociateItemList.

3.1.4.4 TAssociateItemList Methods

The methods of the TAssociateItemList class are listed here.

Public Methods

	Name	Description
	Add (see page 258)	This is Add, a member of class TAssociateItemList.

	Find (see page 258)	This is Find, a member of class TAssociateItemList.
	Get (see page 258)	This is Get, a member of class TAssociateItemList.
	Remove (see page 258)	This is Remove, a member of class TAssociateItemList.

3.1.4.4.1 TAssociateItemList.Add Method

Pascal

```
procedure Add(Popup: TControl; Associate: TControl);
```

C++

```
__fastcall Add(TControl * Popup, TControl * Associate);
```

Description

This is Add, a member of class TAssociateItemList.

3.1.4.4.2 TAssociateItemList.Find Method

Pascal

```
function Find(Associate: TControl): Integer;
```

C++

```
__fastcall int Find(TControl * Associate);
```

Description

This is Find, a member of class TAssociateItemList.

3.1.4.4.3 TAssociateItemList.Get Method

Pascal

```
function Get(const Idx: Integer): TAssociateItem;
```

C++

```
__fastcall TAssociateItem Get(const int Idx);
```

Description

This is Get, a member of class TAssociateItemList.

3.1.4.4.4 TAssociateItemList.Remove Method

Pascal

```
procedure Remove(Associate: TControl);
```

C++

```
__fastcall Remove(TControl * Associate);
```

Description

This is Remove, a member of class TAssociateItemList.

3.1.5 TCustomScreenTipsManager Class

Class Hierarchy

Pascal

```
TCustomScreenTipsManager = class(TComponent);
```

C++

```
class TCustomScreenTipsManager : public TComponent;
```

File

ScreenTips

Description

This is class ScreenTips.TCustomScreenTipsManager.

3.1.5.1 TCustomScreenTipsManager Members

The following tables list the members exposed by TCustomScreenTipsManager.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 263)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.
	DoDuplicateCreateError (see page 263)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 263)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 263)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 263)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 264)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 264)	Initializes the component after the form file has been read into memory.
	Notification (see page 265)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 266)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Events

	Name	Description
	OnCreateScreenTip (see page 269)	This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.
	OnDuplicateCreateError (see page 269)	This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.
	OnGetScreenTipWinClass (see page 270)	This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.
	OnScreenTipsLoaded (see page 270)	This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	OnShowScreenTip (see page 270)	This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 262)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 262)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	Create (see page 260)	Allocates memory and constructs a safely initialized instance of a component.
	CreateScreenTip (see page 262)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	Destroy (see page 261)	Disposes of the component and its owned components.
	GenerateScreenTips (see page 264)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.

	GetScreenTip (see page 264)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 265)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 266)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 266)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

Public Properties

	Name	Description
	BorderStyle (see page 267)	This is BorderStyle, a member of class TCustomScreenTipsManager.
	DisplayOffset (see page 267)	This is DisplayOffset, a member of class TCustomScreenTipsManager.
	Font (see page 267)	This is Font, a member of class TCustomScreenTipsManager.
	Footer (see page 267)	This is Footer, a member of class TCustomScreenTipsManager.
	FooterImage (see page 268)	This is FooterImage, a member of class TCustomScreenTipsManager.
	GradientEndColor (see page 268)	This is GradientEndColor, a member of class TCustomScreenTipsManager.
	GradientStartColor (see page 268)	This is GradientStartColor, a member of class TCustomScreenTipsManager.
	LinkedActionLists (see page 268)	This is LinkedActionLists, a member of class TCustomScreenTipsManager.
	Options (see page 268)	This is Options, a member of class TCustomScreenTipsManager.
	ScreenTips (see page 269)	This is ScreenTips, a member of class TCustomScreenTipsManager.

3.1.5.2 TCustomScreenTipsManager.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3366)

Destroy

System::TObject::Free (see page 3382)

Owner

3.1.5.3 TCustomScreenTipsManager.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomScreenTipsManager();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3366)

DestroyComponents

Destroying

System::TObject::Free (see page 3382)

Notification

Owner

Release

Components

3.1.5.4 TCustomScreenTipsManager Methods

The methods of the TCustomScreenTipsManager class are listed here.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 263)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.

	DoDuplicateCreateError (see page 263)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 263)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 263)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 263)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 264)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 264)	Initializes the component after the form file has been read into memory.
	Notification (see page 265)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 266)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 262)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 262)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	CreateScreenTip (see page 262)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	GenerateScreenTips (see page 264)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.
	GetScreenTip (see page 264)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 265)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 266)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 266)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

3.1.5.4.1 TCustomScreenTipsManager.AddPopup Method

Pascal

```
procedure AddPopup(AScreenTipPopup: TCustomControl);
```

C++

```
__fastcall AddPopup(TCustomControl AScreenTipPopup);
```

Description

This is AddPopup, a member of class TCustomScreenTipsManager.

3.1.5.4.2 TCustomScreenTipsManager.ClearScreenTips Method

Pascal

```
procedure ClearScreenTips;
```

C++

```
__fastcall ClearScreenTips();
```

Description

This is ClearScreenTips, a member of class TCustomScreenTipsManager.

3.1.5.4.3 TCustomScreenTipsManager.CreateScreenTip Method

Pascal

```
function CreateScreenTip(aAction: TBasicAction): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem CreateScreenTip(TBasicAction aAction);
```

Description

This is CreateScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.4.4 TCustomScreenTipsManager.DoCreateScreenTip Method

Pascal

```
procedure DoCreateScreenTip(aItem: TScreenTipItem); virtual;
```

C++

```
virtual __fastcall DoCreateScreenTip(TScreenTipItem aItem);
```

Description

This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.4.5 TCustomScreenTipsManager.DoDuplicateCreateError Method

Pascal

```
procedure DoDuplicateCreateError(aAction: TBasicAction); virtual;
```

C++

```
virtual __fastcall DoDuplicateCreateError(TBasicAction aAction);
```

Description

This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.

3.1.5.4.6 TCustomScreenTipsManager.DoGetScreenTipClass Method

Pascal

```
procedure DoGetScreenTipClass(var WindowClass: TScreenTipsWindowClass); virtual;
```

C++

```
virtual __fastcall DoGetScreenTipClass(TScreenTipsWindowClass WindowClass);
```

Description

This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.

3.1.5.4.7 TCustomScreenTipsManager.DoScreenTipsLoaded Method

Pascal

```
procedure DoScreenTipsLoaded; virtual;
```

C++

```
virtual __fastcall DoScreenTipsLoaded();
```

Description

This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.

3.1.5.4.8 TCustomScreenTipsManager.DoShow Method

Pascal

```
procedure DoShow(Item: TScreenTipItem; var HintInfo: THintInfo); virtual;
```

C++

```
virtual __fastcall DoShow(TScreenTipItem Item, THintInfo HintInfo);
```

Description

This is DoShow, a member of class TCustomScreenTipsManager.

3.1.5.4.9 TCustomScreenTipsManager.DoShowScreenTip Method

Pascal

```
procedure DoShowScreenTip(Action: TBasicAction; Item: TScreenTipItem; var HintInfo:
THintInfo); virtual;
```

C++

```
virtual __fastcall DoShowScreenTip(TBasicAction Action, TScreenTipItem Item, THintInfo
HintInfo);
```

Description

This is DoShowScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.4.10 TCustomScreenTipsManager.GenerateScreenTips Method

Pascal

```
function GenerateScreenTips: Boolean;
```

C++

```
__fastcall Boolean GenerateScreenTips();
```

Description

This is GenerateScreenTips, a member of class TCustomScreenTipsManager.

3.1.5.4.11 TCustomScreenTipsManager.GetScreenTip Method

Pascal

```
function GetScreenTip(aAction: TBasicAction): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTip(TBasicAction aAction);
```

Description

This is GetScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.4.12 TCustomScreenTipsManager.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

3.1.5.4.13 TCustomScreenTipsManager.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

3.1.5.4.14 TCustomScreenTipsManager.RemovePopup Method

Pascal

```
procedure RemovePopup(AScreenTipPopup: TCustomControl);
```

C++

```
__fastcall RemovePopup(TCustomControl AScreenTipPopup);
```

Description

This is RemovePopup, a member of class TCustomScreenTipsManager.

3.1.5.4.15 TCustomScreenTipsManager.RemoveScreenTip Method

Pascal

```
function RemoveScreenTip(aScreenTip: TScreenTipItem): Boolean;
```

C++

```
__fastcall Boolean RemoveScreenTip(TScreenTipItem aScreenTip);
```

Description

This is RemoveScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.4.16 TCustomScreenTipsManager.UpdateActionInfo Method

Pascal

```
procedure UpdateActionInfo(aAction: TContainedAction; aTooltip: TScreenTipItem);
```

C++

```
__fastcall UpdateActionInfo(TContainedAction aAction, TScreenTipItem aTooltip);
```

Description

This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

3.1.5.4.17 TCustomScreenTipsManager.UpdateOptions Method

Pascal

```
procedure UpdateOptions;
```

C++

```
__fastcall UpdateOptions();
```

Description

This is UpdateOptions, a member of class TCustomScreenTipsManager.

3.1.5.5 TCustomScreenTipsManager Properties

The properties of the TCustomScreenTipsManager class are listed here.

Public Properties

	Name	Description
	BorderStyle (see page 267)	This is BorderStyle, a member of class TCustomScreenTipsManager.
	DisplayOffset (see page 267)	This is DisplayOffset, a member of class TCustomScreenTipsManager.
	Font (see page 267)	This is Font, a member of class TCustomScreenTipsManager.
	Footer (see page 267)	This is Footer, a member of class TCustomScreenTipsManager.
	FooterImage (see page 268)	This is FooterImage, a member of class TCustomScreenTipsManager.

	GradientEndColor (see page 268)	This is GradientEndColor, a member of class TCustomScreenTipsManager.
	GradientStartColor (see page 268)	This is GradientStartColor, a member of class TCustomScreenTipsManager.
	LinkedActionLists (see page 268)	This is LinkedActionLists, a member of class TCustomScreenTipsManager.
	Options (see page 268)	This is Options, a member of class TCustomScreenTipsManager.
	ScreenTips (see page 269)	This is ScreenTips, a member of class TCustomScreenTipsManager.

3.1.5.5.1 TCustomScreenTipsManager.BorderStyle Property

Pascal

```
property BorderStyle: TScreenTipBorderStyle;
```

C++

```
__property TScreenTipBorderStyle BorderStyle;
```

Description

This is BorderStyle, a member of class TCustomScreenTipsManager.

3.1.5.5.2 TCustomScreenTipsManager.DisplayOffset Property

Pascal

```
property DisplayOffset: TDisplayOffset;
```

C++

```
__property TDisplayOffset DisplayOffset;
```

Description

This is DisplayOffset, a member of class TCustomScreenTipsManager.

3.1.5.5.3 TCustomScreenTipsManager.Font Property

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

This is Font, a member of class TCustomScreenTipsManager.

3.1.5.5.4 TCustomScreenTipsManager.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TCustomScreenTipsManager.

3.1.5.5.5 TCustomScreenTipsManager.FooterImage Property

Pascal

```
property FooterImage: TPicture;
```

C++

```
__property TPicture FooterImage;
```

Description

This is FooterImage, a member of class TCustomScreenTipsManager.

3.1.5.5.6 TCustomScreenTipsManager.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TCustomScreenTipsManager.

3.1.5.5.7 TCustomScreenTipsManager.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TCustomScreenTipsManager.

3.1.5.5.8 TCustomScreenTipsManager.LinkedActionLists Property

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

This is LinkedActionLists, a member of class TCustomScreenTipsManager.

3.1.5.5.9 TCustomScreenTipsManager.Options Property

Pascal

```
property Options: TScreenTipOptions;
```

C++

```
__property TScreenTipOptions Options;
```

Description

This is Options, a member of class TCustomScreenTipsManager.

3.1.5.5.10 TCustomScreenTipsManager.ScreenTips Property

Pascal

```
property ScreenTips: TScreenTipsCollection;
```

C++

```
__property TScreenTipsCollection ScreenTips;
```

Description

This is ScreenTips, a member of class TCustomScreenTipsManager.

3.1.5.6 TCustomScreenTipsManager Events

The events of the TCustomScreenTipsManager class are listed here.

Public Events

	Name	Description
	OnCreateScreenTip (see page 269)	This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.
	OnDuplicateCreateError (see page 269)	This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.
	OnGetScreenTipWinClass (see page 270)	This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.
	OnScreenTipsLoaded (see page 270)	This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	OnShowScreenTip (see page 270)	This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.6.1 TCustomScreenTipsManager.OnCreateScreenTip Event

Pascal

```
property OnCreateScreenTip: TCreateScreenTipEvent;
```

C++

```
__property TCreateScreenTipEvent OnCreateScreenTip;
```

Description

This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.

3.1.5.6.2 TCustomScreenTipsManager.OnDuplicateCreateError Event

Pascal

```
property OnDuplicateCreateError: TDuplicateCreateErrorEvent;
```

C++

```
__property TDuplicateCreateErrorEvent OnDuplicateCreateError;
```

Description

This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.

3.1.5.6.3 TCustomScreenTipsManager.OnGetScreenTipWinClass Event

Pascal

```
property OnGetScreenTipWinClass: TScreenTipWinClass;
```

C++

```
__property TScreenTipWinClass OnGetScreenTipWinClass;
```

Description

This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.

3.1.5.6.4 TCustomScreenTipsManager.OnScreenTipsLoaded Event

Pascal

```
property OnScreenTipsLoaded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnScreenTipsLoaded;
```

Description

This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.

3.1.5.6.5 TCustomScreenTipsManager.OnShowScreenTip Event

Pascal

```
property OnShowScreenTip: TShowScreenTipEvent;
```

C++

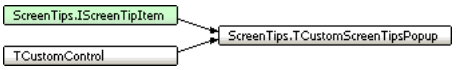
```
__property TShowScreenTipEvent OnShowScreenTip;
```

Description

This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

3.1.6 TCustomScreenTipsPopup Class

Class Hierarchy



Pascal

```
TCustomScreenTipsPopup = class(TCustomControl, IScreenTipItem);
```

C++

```
class TCustomScreenTipsPopup : public TCustomControl, public IScreenTipItem;
```

File

ScreenTips

Description

This is class ScreenTips.TCustomScreenTipsPopup.

3.1.6.1 TCustomScreenTipsPopup Members

The following tables list the members exposed by TCustomScreenTipsPopup.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 273)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 273)	Responds to notifications that components are being created or destroyed.
	Paint (see page 274)	Renders the image of a custom control.
	RemoveLinkToManager (see page 274)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 274)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	Create (see page 271)	Creates an instance of TCustomControl.
	Destroy (see page 272)	Destroys an instance of TCustomControl.
	GetScreenTipItem (see page 273)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

Public Properties

	Name	Description
	Associate (see page 275)	This is Associate, a member of class TCustomScreenTipsPopup.
	Glyph (see page 275)	This is Glyph, a member of class TCustomScreenTipsPopup.
	PopupType (see page 275)	This is PopupType, a member of class TCustomScreenTipsPopup.
	ScreenTip (see page 276)	This is ScreenTip, a member of class TCustomScreenTipsPopup.
	ScreenTipManager (see page 276)	This is ScreenTipManager, a member of class TCustomScreenTipsPopup.
	Transparent (see page 278)	This is Transparent, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	ShowHint (see page 276)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

3.1.6.2 TCustomScreenTipsPopup.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

3.1.6.3 TCustomScreenTipsPopup.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomScreenTipsPopup();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3382)

delete

Create

3.1.6.4 TCustomScreenTipsPopup Methods

The methods of the TCustomScreenTipsPopup class are listed here.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 273)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 273)	Responds to notifications that components are being created or destroyed.
	Paint (see page 274)	Renders the image of a custom control.
	RemoveLinkToManager (see page 274)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 274)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	GetScreenTipItem (see page 273)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

3.1.6.4.1 TCustomScreenTipsPopup.GetScreenTipItem Method

Pascal

```
function GetScreenTipItem: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTipItem();
```

Description

This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

3.1.6.4.2 TCustomScreenTipsPopup.LinkPopupToManager Method

Pascal

```
procedure LinkPopupToManager; virtual;
```

C++

```
virtual __fastcall LinkPopupToManager();
```

Description

This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.

3.1.6.4.3 TCustomScreenTipsPopup.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware

controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

3.1.6.4.4 TCustomScreenTipsPopup.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

3.1.6.4.5 TCustomScreenTipsPopup.RemoveLinkToManager Method

Pascal

```
procedure RemoveLinkToManager; virtual;
```

C++

```
virtual __fastcall RemoveLinkToManager();
```

Description

This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.

3.1.6.4.6 TCustomScreenTipsPopup.UpdateAssociateToManager Method

Pascal

```
procedure UpdateAssociateToManager; virtual;
```

C++

```
virtual __fastcall UpdateAssociateToManager();
```

Description

This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

3.1.6.5 TCustomScreenTipsPopup Properties

The properties of the TCustomScreenTipsPopup class are listed here.

Public Properties

	Name	Description
	Associate (see page 275)	This is Associate, a member of class TCustomScreenTipsPopup.
	Glyph (see page 275)	This is Glyph, a member of class TCustomScreenTipsPopup.
	PopupType (see page 275)	This is PopupType, a member of class TCustomScreenTipsPopup.
	ScreenTip (see page 276)	This is ScreenTip, a member of class TCustomScreenTipsPopup.
	ScreenTipManager (see page 276)	This is ScreenTipManager, a member of class TCustomScreenTipsPopup.
	Transparent (see page 278)	This is Transparent, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	ShowHint (see page 276)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

3.1.6.5.1 TCustomScreenTipsPopup.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TCustomScreenTipsPopup.

3.1.6.5.2 TCustomScreenTipsPopup.Glyph Property

Pascal

```
property Glyph: TPicture;
```

C++

```
__property TPicture Glyph;
```

Description

This is Glyph, a member of class TCustomScreenTipsPopup.

3.1.6.5.3 TCustomScreenTipsPopup.PopupType Property

Pascal

```
property PopupType: TScreenTipPopupType;
```

C++

```
__property TScreenTipPopupType PopupType;
```

Description

This is PopupType, a member of class TCustomScreenTipsPopup.

3.1.6.5.4 TCustomScreenTipsPopup.ScreenTip Property

Pascal

```
property ScreenTip: TScreenTipItem;
```

C++

```
__property TScreenTipItem ScreenTip;
```

Description

This is ScreenTip, a member of class TCustomScreenTipsPopup.

3.1.6.5.5 TCustomScreenTipsPopup.ScreenTipManager Property

Pascal

```
property ScreenTipManager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTipManager;
```

Description

This is ScreenTipManager, a member of class TCustomScreenTipsPopup.

3.1.6.5.6 TCustomScreenTipsPopup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
```

```
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

3.1.6.5.7 TCustomScreenTipsPopup.Transparent Property

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

This is Transparent, a member of class TCustomScreenTipsPopup.

3.1.7 TDisplayOffset Class

Class Hierarchy

Pascal

```
TDisplayOffset = class(TPersistent);
```

C++

```
class TDisplayOffset : public TPersistent;
```

File

ScreenTips

Description

This is class ScreenTips.TDisplayOffset.

3.1.7.1 TDisplayOffset Members

The following tables list the members exposed by TDisplayOffset.

Public Methods

	Name	Description
	Create (see page 278)	Constructs an object and initializes its data before the object is first used.

Published Properties

	Name	Description
	Control (see page 279)	This is Control, a member of class TDisplayOffset.
	X (see page 279)	This is X, a member of class TDisplayOffset.
	Y (see page 279)	This is Y, a member of class TDisplayOffset.

3.1.7.2 TDisplayOffset.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDisplayOffset();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

3.1.7.3 TDisplayOffset Properties

The properties of the TDisplayOffset class are listed here.

Published Properties

	Name	Description
	Control (see page 279)	This is Control, a member of class TDisplayOffset.
	X (see page 279)	This is X, a member of class TDisplayOffset.
	Y (see page 279)	This is Y, a member of class TDisplayOffset.

3.1.7.3.1 TDisplayOffset.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TDisplayOffset.

3.1.7.3.2 TDisplayOffset.X Property

Pascal

```
property X: Integer;
```

C++

```
__property int X;
```

Description

This is X, a member of class TDisplayOffset.

3.1.7.3.3 TDisplayOffset.Y Property

Pascal

```
property Y: Integer;
```

C++

```
__property int Y;
```

Description

This is Y, a member of class TDisplayOffset.

3.1.8 TScreenTipBorderStyle Class

Class Hierarchy

Pascal

```
TScreenTipBorderStyle = class(TPersistent);
```

C++

```
class TScreenTipBorderStyle : public TPersistent;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipBorderStyle.

3.1.8.1 TScreenTipBorderStyle Members

The following tables list the members exposed by TScreenTipBorderStyle.

Public Methods

	Name	Description
	Create (see page 280)	Constructs an object and initializes its data before the object is first used.

Published Properties

	Name	Description
	Color (see page 281)	This is Color, a member of class TScreenTipBorderStyle.
	CornerSize (see page 281)	This is CornerSize, a member of class TScreenTipBorderStyle.
	Style (see page 281)	This is Style, a member of class TScreenTipBorderStyle.

3.1.8.2 TScreenTipBorderStyle.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScreenTipBorderStyle();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The

System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

3.1.8.3 TScreenTipBorderStyle Properties

The properties of the TScreenTipBorderStyle class are listed here.

Published Properties

	Name	Description
	Color (see page 281)	This is Color, a member of class TScreenTipBorderStyle.
	CornerSize (see page 281)	This is CornerSize, a member of class TScreenTipBorderStyle.
	Style (see page 281)	This is Style, a member of class TScreenTipBorderStyle.

3.1.8.3.1 TScreenTipBorderStyle.Color Property

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

This is Color, a member of class TScreenTipBorderStyle.

3.1.8.3.2 TScreenTipBorderStyle.CornerSize Property

Pascal

```
property CornerSize: Integer;
```

C++

```
__property int CornerSize;
```

Description

This is CornerSize, a member of class TScreenTipBorderStyle.

3.1.8.3.3 TScreenTipBorderStyle.Style Property

Pascal

```
property Style: TTipBorderStyle;
```

C++

```
__property TTipBorderStyle Style;
```

Description

This is Style, a member of class TScreenTipBorderStyle.

3.1.9 TScreenTipItem Class

Class Hierarchy

Pascal

```
TScreenTipItem = class(TCollectionItem);
```

C++

```
class TScreenTipItem : public TCollectionItem;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipItem.

3.1.9.1 TScreenTipItem Members

The following tables list the members exposed by TScreenTipItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 284)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 284)	Copies the contents of another, similar object.
	Create (see page 283)	Creates and initializes a TCollectionItem instance.
	Destroy (see page 283)	Destroys the TCollectionItem instance and frees its memory.
	IsEnabled (see page 285)	This is IsEnabled, a member of class TScreenTipItem.
	SetManager (see page 285)	This is SetManager, a member of class TScreenTipItem.

Public Properties

	Name	Description
	Height (see page 287)	This is Height, a member of class TScreenTipItem.
	Width (see page 288)	This is Width, a member of class TScreenTipItem.

Published Properties

	Name	Description
	Action (see page 286)	This is Action, a member of class TScreenTipItem.
	Description (see page 286)	This is Description, a member of class TScreenTipItem.
	DisabledDescription (see page 286)	This is DisabledDescription, a member of class TScreenTipItem.
	DisabledHeader (see page 286)	This is DisabledHeader, a member of class TScreenTipItem.
	Header (see page 286)	This is Header, a member of class TScreenTipItem.
	Image (see page 287)	This is Image, a member of class TScreenTipItem.
	Manager (see page 287)	This is Manager, a member of class TScreenTipItem.
	ShowFooter (see page 287)	This is ShowFooter, a member of class TScreenTipItem.
	ShowImage (see page 288)	This is ShowImage, a member of class TScreenTipItem.

3.1.9.2 TScreenTipItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TScreenTipItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

3.1.9.3 TScreenTipItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreenTipItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free (see page 3382)

3.1.9.4 TScreenTipItem Methods

The methods of the TScreenTipItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 284)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 284)	Copies the contents of another, similar object.
	IsEnabled (see page 285)	This is IsEnabled, a member of class TScreenTipItem.
	SetManager (see page 285)	This is SetManager, a member of class TScreenTipItem.

3.1.9.4.1 TScreenTipItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError (see page 3985)

3.1.9.4.2 TScreenTipItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

3.1.9.4.3 TScreenTipItem.IsEnabled Method

Pascal

```
function IsEnabled: Boolean;
```

C++

```
__fastcall Boolean IsEnabled();
```

Description

This is IsEnabled, a member of class TScreenTipItem.

3.1.9.4.4 TScreenTipItem.SetManager Method

Pascal

```
procedure SetManager(AManager: TCustomScreenTipsManager);
```

C++

```
__fastcall SetManager(TCustomScreenTipsManager AManager);
```

Description

This is SetManager, a member of class TScreenTipItem.

3.1.9.5 TScreenTipItem Properties

The properties of the TScreenTipItem class are listed here.

Public Properties

	Name	Description
	Height (see page 287)	This is Height, a member of class TScreenTipItem.
	Width (see page 288)	This is Width, a member of class TScreenTipItem.

Published Properties

	Name	Description
	Action (see page 286)	This is Action, a member of class TScreenTipItem.
	Description (see page 286)	This is Description, a member of class TScreenTipItem.
	DisabledDescription (see page 286)	This is DisabledDescription, a member of class TScreenTipItem.
	DisabledHeader (see page 286)	This is DisabledHeader, a member of class TScreenTipItem.
	Header (see page 286)	This is Header, a member of class TScreenTipItem.
	Image (see page 287)	This is Image, a member of class TScreenTipItem.
R	Manager (see page 287)	This is Manager, a member of class TScreenTipItem.
	ShowFooter (see page 287)	This is ShowFooter, a member of class TScreenTipItem.
	ShowImage (see page 288)	This is ShowImage, a member of class TScreenTipItem.

3.1.9.5.1 TScreenTipItem.Action Property

Pascal

```
property Action: TContainedAction;
```

C++

```
__property TContainedAction Action;
```

Description

This is Action, a member of class TScreenTipItem.

3.1.9.5.2 TScreenTipItem.Description Property

Pascal

```
property Description: TStrings;
```

C++

```
__property TStrings Description;
```

Description

This is Description, a member of class TScreenTipItem.

3.1.9.5.3 TScreenTipItem.DisabledDescription Property

Pascal

```
property DisabledDescription: TStrings;
```

C++

```
__property TStrings DisabledDescription;
```

Description

This is DisabledDescription, a member of class TScreenTipItem.

3.1.9.5.4 TScreenTipItem.DisabledHeader Property

Pascal

```
property DisabledHeader: string;
```

C++

```
__property UnicodeString DisabledHeader;
```

Description

This is DisabledHeader, a member of class TScreenTipItem.

3.1.9.5.5 TScreenTipItem.Header Property

Pascal

```
property Header: string;
```

C++

```
__property UnicodeString Header;
```

Description

This is Header, a member of class TScreenTipItem.

3.1.9.5.6 TScreenTipItem.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TScreenTipItem.

3.1.9.5.7 TScreenTipItem.Image Property

Pascal

```
property Image: TPicture;
```

C++

```
__property TPicture Image;
```

Description

This is Image, a member of class TScreenTipItem.

3.1.9.5.8 TScreenTipItem.Manager Property

Pascal

```
property Manager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager Manager;
```

Description

This is Manager, a member of class TScreenTipItem.

3.1.9.5.9 TScreenTipItem.ShowFooter Property

Pascal

```
property ShowFooter: Boolean;
```

C++

```
__property Boolean ShowFooter;
```

Description

This is ShowFooter, a member of class TScreenTipItem.

3.1.9.5.10 TScreenTipItem.ShowImage Property

Pascal

```
property ShowImage: Boolean;
```

C++

```
__property Boolean ShowImage;
```

Description

This is ShowImage, a member of class TScreenTipItem.

3.1.9.5.11 TScreenTipItem.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TScreenTipItem.

3.1.10 TScreenTipsCollection Class

Class Hierarchy

Pascal

```
TScreenTipsCollection = class(TCollection);
```

C++

```
class TScreenTipsCollection : public TCollection;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsCollection.

3.1.10.1 TScreenTipsCollection Members

The following tables list the members exposed by TScreenTipsCollection.

Protected Methods

	Name	Description
	GetOwner (see page 290)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 289)	Creates a new TCollectionItem instance and adds it to the Items array.
	Create (see page 289)	This is Create, a member of class TScreenTipsCollection.
	Insert (see page 290)	Creates a new TCollectionItem instance and adds it to the Items array.

Public Properties

	Name	Description
	Items (see page 291)	Lists the items in the collection.

3.1.10.2 TScreenTipsCollection.Create Constructor

Pascal

```
constructor Create(Manager: TCustomScreenTipsManager);
```

C++

```
__fastcall TScreenTipsCollection(TCustomScreenTipsManager Manager);
```

Description

This is Create, a member of class TScreenTipsCollection.

3.1.10.3 TScreenTipsCollection Methods

The methods of the TScreenTipsCollection class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 290)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 289)	Creates a new TCollectionItem instance and adds it to the Items array.
	Insert (see page 290)	Creates a new TCollectionItem instance and adds it to the Items array.

3.1.10.3.1 TScreenTipsCollection.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items

Insert

Create

Delete

3.1.10.3.2 TScreenTipsCollection.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

3.1.10.3.3 TScreenTipsCollection.Insert Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Insert(Index: Integer): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem Insert(int Index);
```

Description

Call Insert to add a new item at a specified position in the collection. Existing items (starting from the specified position) are moved up in the Items array.

Insert returns the new collection item.

See Also

Add

Items

3.1.10.4 TScreenTipsCollection Properties

The properties of the TScreenTipsCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 291)	Lists the items in the collection.

3.1.10.4.1 TScreenTipsCollection.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TScreenTipItem;
```

C++

```
__property TScreenTipItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count

ItemClass

Index

Add

Insert

3.1.11 TScreenTipsManager Class

Class Hierarchy

Pascal

```
TScreenTipsManager = class(TCustomScreenTipsManager);
```

C++

```
class TScreenTipsManager : public TCustomScreenTipsManager;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsManager.

3.1.11.1 TScreenTipsManager Members

The following tables list the members exposed by TScreenTipsManager.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 263)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.
	DoDuplicateCreateError (see page 263)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 263)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 263)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 263)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 264)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 264)	Initializes the component after the form file has been read into memory.
	Notification (see page 265)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 266)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 262)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 262)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	Create (see page 260)	Allocates memory and constructs a safely initialized instance of a component.
	CreateScreenTip (see page 262)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	Destroy (see page 261)	Disposes of the component and its owned components.
	GenerateScreenTips (see page 264)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.
	GetScreenTip (see page 264)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 265)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 266)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 266)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

Published Events

	Name	Description
	OnCreateScreenTip (see page 295)	This is OnCreateScreenTip, a member of class TScreenTipsManager.
	OnDuplicateCreateError (see page 296)	This is OnDuplicateCreateError, a member of class TScreenTipsManager.
	OnGetScreenTipWinClass (see page 296)	This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.
	OnScreenTipsLoaded (see page 296)	This is OnScreenTipsLoaded, a member of class TScreenTipsManager.
	OnShowScreenTip (see page 296)	This is OnShowScreenTip, a member of class TScreenTipsManager.

Published Properties

	Name	Description
	BorderStyle (see page 293)	This is BorderStyle, a member of class TScreenTipsManager.
	DisplayOffset (see page 293)	This is DisplayOffset, a member of class TScreenTipsManager.
	Font (see page 293)	This is Font, a member of class TScreenTipsManager.
	Footer (see page 294)	This is Footer, a member of class TScreenTipsManager.
	FooterImage (see page 294)	This is FooterImage, a member of class TScreenTipsManager.
	GradientEndColor (see page 294)	This is GradientEndColor, a member of class TScreenTipsManager.
	GradientStartColor (see page 294)	This is GradientStartColor, a member of class TScreenTipsManager.
	LinkedActionLists (see page 294)	This is LinkedActionLists, a member of class TScreenTipsManager.
	Options (see page 295)	This is Options, a member of class TScreenTipsManager.
	ScreenTips (see page 295)	This is ScreenTips, a member of class TScreenTipsManager.

3.1.11.2 TScreenTipsManager Properties

The properties of the TScreenTipsManager class are listed here.

Published Properties

	Name	Description
	BorderStyle (see page 293)	This is BorderStyle, a member of class TScreenTipsManager.
	DisplayOffset (see page 293)	This is DisplayOffset, a member of class TScreenTipsManager.
	Font (see page 293)	This is Font, a member of class TScreenTipsManager.
	Footer (see page 294)	This is Footer, a member of class TScreenTipsManager.
	FooterImage (see page 294)	This is FooterImage, a member of class TScreenTipsManager.
	GradientEndColor (see page 294)	This is GradientEndColor, a member of class TScreenTipsManager.
	GradientStartColor (see page 294)	This is GradientStartColor, a member of class TScreenTipsManager.
	LinkedActionLists (see page 294)	This is LinkedActionLists, a member of class TScreenTipsManager.
	Options (see page 295)	This is Options, a member of class TScreenTipsManager.
	ScreenTips (see page 295)	This is ScreenTips, a member of class TScreenTipsManager.

3.1.11.2.1 TScreenTipsManager.BorderStyle Property

Pascal

```
property BorderStyle: TScreenTipBorderStyle;
```

C++

```
__property TScreenTipBorderStyle BorderStyle;
```

Description

This is BorderStyle, a member of class TScreenTipsManager.

3.1.11.2.2 TScreenTipsManager.DisplayOffset Property

Pascal

```
property DisplayOffset: TDisplayOffset;
```

C++

```
__property TDisplayOffset DisplayOffset;
```

Description

This is DisplayOffset, a member of class TScreenTipsManager.

3.1.11.2.3 TScreenTipsManager.Font Property

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

This is Font, a member of class TScreenTipsManager.

3.1.11.2.4 TScreenTipsManager.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TScreenTipsManager.

3.1.11.2.5 TScreenTipsManager.FooterImage Property

Pascal

```
property FooterImage: TPicture;
```

C++

```
__property TPicture FooterImage;
```

Description

This is FooterImage, a member of class TScreenTipsManager.

3.1.11.2.6 TScreenTipsManager.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TScreenTipsManager.

3.1.11.2.7 TScreenTipsManager.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TScreenTipsManager.

3.1.11.2.8 TScreenTipsManager.LinkedActionLists Property

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

This is LinkedActionLists, a member of class TScreenTipsManager.

3.1.11.2.9 TScreenTipsManager.Options Property

Pascal

```
property Options: TScreenTipOptions;
```

C++

```
__property TScreenTipOptions Options;
```

Description

This is Options, a member of class TScreenTipsManager.

3.1.11.2.10 TScreenTipsManager.ScreenTips Property

Pascal

```
property ScreenTips: TScreenTipsCollection;
```

C++

```
__property TScreenTipsCollection ScreenTips;
```

Description

This is ScreenTips, a member of class TScreenTipsManager.

3.1.11.3 TScreenTipsManager Events

The events of the TScreenTipsManager class are listed here.

Published Events

	Name	Description
	OnCreateScreenTip (see page 295)	This is OnCreateScreenTip, a member of class TScreenTipsManager.
	OnDuplicateCreateError (see page 296)	This is OnDuplicateCreateError, a member of class TScreenTipsManager.
	OnGetScreenTipWinClass (see page 296)	This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.
	OnScreenTipsLoaded (see page 296)	This is OnScreenTipsLoaded, a member of class TScreenTipsManager.
	OnShowScreenTip (see page 296)	This is OnShowScreenTip, a member of class TScreenTipsManager.

3.1.11.3.1 TScreenTipsManager.OnCreateScreenTip Event

Pascal

```
property OnCreateScreenTip: TCreateScreenTipEvent;
```

C++

```
__property TCreateScreenTipEvent OnCreateScreenTip;
```

Description

This is OnCreateScreenTip, a member of class TScreenTipsManager.

3.1.11.3.2 TScreenTipsManager.OnDuplicateCreateError Event

Pascal

```
property OnDuplicateCreateError: TDuplicateCreateErrorEvent;
```

C++

```
__property TDuplicateCreateErrorEvent OnDuplicateCreateError;
```

Description

This is OnDuplicateCreateError, a member of class TScreenTipsManager.

3.1.11.3.3 TScreenTipsManager.OnGetScreenTipWinClass Event

Pascal

```
property OnGetScreenTipWinClass: TScreenTipWinClass;
```

C++

```
__property TScreenTipWinClass OnGetScreenTipWinClass;
```

Description

This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.

3.1.11.3.4 TScreenTipsManager.OnScreenTipsLoaded Event

Pascal

```
property OnScreenTipsLoaded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnScreenTipsLoaded;
```

Description

This is OnScreenTipsLoaded, a member of class TScreenTipsManager.

3.1.11.3.5 TScreenTipsManager.OnShowScreenTip Event

Pascal

```
property OnShowScreenTip: TShowScreenTipEvent;
```

C++

```
__property TShowScreenTipEvent OnShowScreenTip;
```

Description

This is OnShowScreenTip, a member of class TScreenTipsManager.

3.1.12 TScreenTipsPopup Class

Class Hierarchy

Pascal

```
TScreenTipsPopup = class(TCustomScreenTipsPopup);
```

C++

```
class TScreenTipsPopup : public TCustomScreenTipsPopup;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsPopup.

3.1.12.1 TScreenTipsPopup Members

The following tables list the members exposed by TScreenTipsPopup.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 273)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 273)	Responds to notifications that components are being created or destroyed.
	Paint (see page 274)	Renders the image of a custom control.
	RemoveLinkToManager (see page 274)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 274)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	Create (see page 271)	Creates an instance of TCustomControl.
	Destroy (see page 272)	Destroys an instance of TCustomControl.
	GetScreenTipItem (see page 316)	This is GetScreenTipItem, a member of class IScreenTipItem.
	GetScreenTipItem (see page 273)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	Align (see page 298)	Determines how the control aligns within its container (parent control).
	Anchors (see page 301)	Specifies how the control is anchored to its parent.
	Associate (see page 302)	This is Associate, a member of class TScreenTipsPopup.
	Glyph (see page 302)	This is Glyph, a member of class TScreenTipsPopup.
	PopupType (see page 302)	This is PopupType, a member of class TScreenTipsPopup.
	ScreenTip (see page 303)	This is ScreenTip, a member of class TScreenTipsPopup.
	ScreenTipManager (see page 303)	This is ScreenTipManager, a member of class TScreenTipsPopup.
	ShowHint (see page 276)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 303)	This is Transparent, a member of class TScreenTipsPopup.

	Visible (see page 303)	Determines whether the component appears onscreen.

3.1.12.2 TScreenTipsPopup Properties

The properties of the TScreenTipsPopup class are listed here.

Published Properties

	Name	Description
	Align (see page 298)	Determines how the control aligns within its container (parent control).
	Anchors (see page 301)	Specifies how the control is anchored to its parent.
	Associate (see page 302)	This is Associate, a member of class TScreenTipsPopup.
	Glyph (see page 302)	This is Glyph, a member of class TScreenTipsPopup.
	PopupType (see page 302)	This is PopupType, a member of class TScreenTipsPopup.
	ScreenTip (see page 303)	This is ScreenTip, a member of class TScreenTipsPopup.
	ScreenTipManager (see page 303)	This is ScreenTipManager, a member of class TScreenTipsPopup.
	Transparent (see page 303)	This is Transparent, a member of class TScreenTipsPopup.
	Visible (see page 303)	Determines whether the component appears onscreen.

3.1.12.2.1 TScreenTipsPopup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
```

```
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
```

```
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
```

```
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

3.1.12.2.2 TScreenTipsPopup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

3.1.12.2.3 TScreenTipsPopup.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TScreenTipsPopup.

3.1.12.2.4 TScreenTipsPopup.Glyph Property

Pascal

```
property Glyph: TPicture;
```

C++

```
__property TPicture Glyph;
```

Description

This is Glyph, a member of class TScreenTipsPopup.

3.1.12.2.5 TScreenTipsPopup.PopupType Property

Pascal

```
property PopupType: TScreenTipPopupType;
```

C++

```
__property TScreenTipPopupType PopupType;
```

Description

This is PopupType, a member of class TScreenTipsPopup.

3.1.12.2.6 TScreenTipsPopup.ScreenTip Property

Pascal

```
property ScreenTip: TScreenTipItem;
```

C++

```
__property TScreenTipItem ScreenTip;
```

Description

This is ScreenTip, a member of class TScreenTipsPopup.

3.1.12.2.7 TScreenTipsPopup.ScreenTipManager Property

Pascal

```
property ScreenTipManager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTipManager;
```

Description

This is ScreenTipManager, a member of class TScreenTipsPopup.

3.1.12.2.8 TScreenTipsPopup.Transparent Property

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

This is Transparent, a member of class TScreenTipsPopup.

3.1.12.2.9 TScreenTipsPopup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

3.1.13 TScreenTipsWindow Class

Class Hierarchy

Pascal

```
TScreenTipsWindow = class(THintWindow);
```

C++

```
class TScreenTipsWindow : public THintWindow;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsWindow.

3.1.13.1 TScreenTipsWindow Members

The following tables list the members exposed by TScreenTipsWindow.

Protected Methods

	Name	Description
	CMMouseenter (see page 307)	This is CMMouseenter, a member of class TScreenTipsWindow.
	CMMouseLeave (see page 308)	This is CMMouseLeave, a member of class TScreenTipsWindow.
V	CreateParams (see page 308)	Initializes a window-creation parameter data structure.
V	CreateRoundedStyle (see page 308)	This is CreateRoundedStyle, a member of class TScreenTipsWindow.
V	CreateTipBorderStyle (see page 309)	This is CreateTipBorderStyle, a member of class TScreenTipsWindow.
V	CreateWnd (see page 309)	Creates the underlying window.
V	DrawBackground (see page 309)	This is DrawBackground, a member of class TScreenTipsWindow.
V	DrawBorder (see page 310)	This is DrawBorder, a member of class TScreenTipsWindow.
V	DrawFooter (see page 310)	This is DrawFooter, a member of class TScreenTipsWindow.
V	DrawHeader (see page 310)	This is DrawHeader, a member of class TScreenTipsWindow.
V	Paint (see page 310)	Renders the image of a custom control.
	WMNCHitTest (see page 311)	This is WMNCHitTest, a member of class TScreenTipsWindow.

Public Methods

	Name	Description
V	ActivateToolTip (see page 307)	This is ActivateToolTip, a member of class TScreenTipsWindow.
V	Create (see page 306)	Creates and initializes an instance of THintWindow.
V	Destroy (see page 306)	Destroys an instance of TCustomControl.

	HideHint (see page 310)	This is HideHint, a member of class TScreenTipsWindow.

Public Properties

	Name	Description
R	ActiveItem (see page 311)	This is ActiveItem, a member of class TScreenTipsWindow.
	Body (see page 312)	This is Body, a member of class TScreenTipsWindow.
	CornerSize (see page 312)	This is CornerSize, a member of class TScreenTipsWindow.
	Footer (see page 312)	This is Footer, a member of class TScreenTipsWindow.
	FooterImage (see page 312)	This is FooterImage, a member of class TScreenTipsWindow.
	GradientEndColor (see page 313)	This is GradientEndColor, a member of class TScreenTipsWindow.
	GradientStartColor (see page 313)	This is GradientStartColor, a member of class TScreenTipsWindow.
	HintControl (see page 313)	This is HintControl, a member of class TScreenTipsWindow.
	Image (see page 313)	This is Image, a member of class TScreenTipsWindow.
	TipBorderStyle (see page 313)	This is TipBorderStyle, a member of class TScreenTipsWindow.

3.1.13.2 TScreenTipsWindow.Create Constructor

Creates and initializes an instance of THintWindow.

Pascal

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

C++

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

Description

Call Create to instantiate a hint window at runtime. Hint windows activated by the application in response to the ShowHint property of a control are created automatically.

See Also

ActivateHint

ShowHint

3.1.13.3 TScreenTipsWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreenTipsWindow();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a

descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3382)

delete

Create

3.1.13.4 TScreenTipsWindow Methods

The methods of the TScreenTipsWindow class are listed here.

Protected Methods

	Name	Description
	CMMouseenter (see page 307)	This is CMMouseenter, a member of class TScreenTipsWindow.
	CMMouseLeave (see page 308)	This is CMMouseLeave, a member of class TScreenTipsWindow.
V	CreateParams (see page 308)	Initializes a window-creation parameter data structure.
V	CreateRoundedStyle (see page 308)	This is CreateRoundedStyle, a member of class TScreenTipsWindow.
V	CreateTipBorderStyle (see page 309)	This is CreateTipBorderStyle, a member of class TScreenTipsWindow.
V	CreateWnd (see page 309)	Creates the underlying window.
V	DrawBackground (see page 309)	This is DrawBackground, a member of class TScreenTipsWindow.
V	DrawBorder (see page 310)	This is DrawBorder, a member of class TScreenTipsWindow.
V	DrawFooter (see page 310)	This is DrawFooter, a member of class TScreenTipsWindow.
V	DrawHeader (see page 310)	This is DrawHeader, a member of class TScreenTipsWindow.
V	Paint (see page 310)	Renders the image of a custom control.
	WMNCHitTest (see page 311)	This is WMNCHitTest, a member of class TScreenTipsWindow.

Public Methods

	Name	Description
V	ActivateToolTip (see page 307)	This is ActivateToolTip, a member of class TScreenTipsWindow.
	HideHint (see page 310)	This is HideHint, a member of class TScreenTipsWindow.

3.1.13.4.1 TScreenTipsWindow.ActivateToolTip Method

Pascal

```
procedure ActivateToolTip(Rect: TRect; Item: TScreenTipItem); virtual;
```

C++

```
virtual __fastcall ActivateToolTip(TRect Rect, TScreenTipItem Item);
```

Description

This is ActivateToolTip, a member of class TScreenTipsWindow.

3.1.13.4.2 TScreenTipsWindow.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TScreenTipsWindow.

3.1.13.4.3 TScreenTipsWindow.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TScreenTipsWindow.

3.1.13.4.4 TScreenTipsWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

3.1.13.4.5 TScreenTipsWindow.CreateRoundedStyle Method

Pascal

```
procedure CreateRoundedStyle; virtual;
```

C++

```
virtual __fastcall CreateRoundedStyle();
```

Description

This is CreateRoundedStyle, a member of class TScreenTipsWindow.

3.1.13.4.6 TScreenTipsWindow.CreateTipBorderStyle Method

Pascal

```
procedure CreateTipBorderStyle; virtual;
```

C++

```
virtual __fastcall CreateTipBorderStyle();
```

Description

This is CreateTipBorderStyle, a member of class TScreenTipsWindow.

3.1.13.4.7 TScreenTipsWindow.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

3.1.13.4.8 TScreenTipsWindow.DrawBackground Method

Pascal

```
procedure DrawBackground(const aRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawBackground(const TRect aRect);
```

Description

This is DrawBackground, a member of class TScreenTipsWindow.

3.1.13.4.9 TScreenTipsWindow.DrawBorder Method

Pascal

```
procedure DrawBorder(const aRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawBorder(const TRect aRect);
```

Description

This is DrawBorder, a member of class TScreenTipsWindow.

3.1.13.4.10 TScreenTipsWindow.DrawFooter Method

Pascal

```
procedure DrawFooter; virtual;
```

C++

```
virtual __fastcall DrawFooter();
```

Description

This is DrawFooter, a member of class TScreenTipsWindow.

3.1.13.4.11 TScreenTipsWindow.DrawHeader Method

Pascal

```
function DrawHeader: Integer; virtual;
```

C++

```
virtual __fastcall int DrawHeader();
```

Description

This is DrawHeader, a member of class TScreenTipsWindow.

3.1.13.4.12 TScreenTipsWindow.HideHint Method

Pascal

```
procedure HideHint;
```

C++

```
__fastcall HideHint();
```

Description

This is HideHint, a member of class TScreenTipsWindow.

3.1.13.4.13 TScreenTipsWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

3.1.13.4.14 TScreenTipsWindow.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TScreenTipsWindow.

3.1.13.5 TScreenTipsWindow Properties

The properties of the TScreenTipsWindow class are listed here.

Public Properties

	Name	Description
R	ActiveItem (see page 311)	This is ActiveItem, a member of class TScreenTipsWindow.
	Body (see page 312)	This is Body, a member of class TScreenTipsWindow.
	CornerSize (see page 312)	This is CornerSize, a member of class TScreenTipsWindow.
	Footer (see page 312)	This is Footer, a member of class TScreenTipsWindow.
	FooterImage (see page 312)	This is FooterImage, a member of class TScreenTipsWindow.
	GradientEndColor (see page 313)	This is GradientEndColor, a member of class TScreenTipsWindow.
	GradientStartColor (see page 313)	This is GradientStartColor, a member of class TScreenTipsWindow.
	HintControl (see page 313)	This is HintControl, a member of class TScreenTipsWindow.
	Image (see page 313)	This is Image, a member of class TScreenTipsWindow.
	TipBorderStyle (see page 313)	This is TipBorderStyle, a member of class TScreenTipsWindow.

3.1.13.5.1 TScreenTipsWindow.ActiveItem Property

Pascal

```
property ActiveItem: TScreenTipItem;
```

C++

```
__property TScreenTipItem ActiveItem;
```

Description

This is ActiveItem, a member of class TScreenTipsWindow.

3.1.13.5.2 TScreenTipsWindow.Body Property

Pascal

```
property Body: string;
```

C++

```
__property UnicodeString Body;
```

Description

This is Body, a member of class TScreenTipsWindow.

3.1.13.5.3 TScreenTipsWindow.CornerSize Property

Pascal

```
property CornerSize: Integer;
```

C++

```
__property int CornerSize;
```

Description

This is CornerSize, a member of class TScreenTipsWindow.

3.1.13.5.4 TScreenTipsWindow.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TScreenTipsWindow.

3.1.13.5.5 TScreenTipsWindow.FooterImage Property

Pascal

```
property FooterImage: TBitmap;
```

C++

```
__property TBitmap FooterImage;
```

Description

This is FooterImage, a member of class TScreenTipsWindow.

3.1.13.5.6 TScreenTipsWindow.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TScreenTipsWindow.

3.1.13.5.7 TScreenTipsWindow.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TScreenTipsWindow.

3.1.13.5.8 TScreenTipsWindow.HintControl Property

Pascal

```
property HintControl: TControl;
```

C++

```
__property TControl * HintControl;
```

Description

This is HintControl, a member of class TScreenTipsWindow.

3.1.13.5.9 TScreenTipsWindow.Image Property

Pascal

```
property Image: TPicture;
```

C++

```
__property TPicture Image;
```

Description

This is Image, a member of class TScreenTipsWindow.

3.1.13.5.10 TScreenTipsWindow.TipBorderStyle Property

Pascal

```
property TipBorderStyle: TTipBorderStyle;
```

C++

```
__property TTipBorderStyle TipBorderStyle;
```

Description

This is TipBorderStyle, a member of class TScreenTipsWindow.

3.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IScreenTipItem (see page 315)	This is class ScreenTips.IScreenTipItem.

3.2.1 IScreenTipItem Interface

Class Hierarchy

ScreenTips.IScreenTipItem

Pascal

```
IScreenTipItem = interface;
```

C++

```
__interface IScreenTipItem;
```

File

ScreenTips

Description

This is class ScreenTips.IScreenTipItem.

3.2.1.1 IScreenTipItem Members

The following tables list the members exposed by IScreenTipItem.

Public Methods

	Name	Description
	GetScreenTipItem (see page 316)	This is GetScreenTipItem, a member of class IScreenTipItem.

3.2.1.2 IScreenTipItem Methods

The methods of the IScreenTipItem class are listed here.

Public Methods

	Name	Description
	GetScreenTipItem (see page 316)	This is GetScreenTipItem, a member of class IScreenTipItem.

3.2.1.2.1 IScreenTipItem.GetScreenTipItem Method

Pascal

```
function GetScreenTipItem: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTipItem();
```

Description

This is GetScreenTipItem, a member of class IScreenTipItem.

3.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDescriptionType (see page 317)	This is record ScreenTips.TDescriptionType.
	TImageAlign (see page 317)	This is record ScreenTips.TImageAlign.
	TItemImageAlign (see page 318)	This is record ScreenTips.TItemImageAlign.
	TScreenTipOption (see page 318)	This is record ScreenTips.TScreenTipOption.
	TScreenTipPopupType (see page 319)	This is record ScreenTips.TScreenTipPopupType.
	TTipBorderStyle (see page 319)	This is record ScreenTips.TTipBorderStyle.

3.3.1 ScreenTips.TDescriptionType Enumeration

Pascal

```
TDescriptionType = (
  dtManual,
  dtHelp,
  dtResource
);
```

C++

```
enum TDescriptionType {
  dtManual,
  dtHelp,
  dtResource
};
```

File

ScreenTips

Description

This is record ScreenTips.TDescriptionType.

3.3.2 ScreenTips.TImageAlign Enumeration

Pascal

```
TImageAlign = (
  iaLeft,
  iaRight
);
```

C++

```
enum TImageAlign {
  iaLeft,
  iaRight
```

```
};
```

File

ScreenTips

Description

This is record ScreenTips.TImageAlign.

3.3.3 ScreenTips.TItemImageAlign Enumeration

Pascal

```
TItemImageAlign = (
  iiaLeft,
  iiaCenter,
  iiaRight
);
```

C++

```
enum TItemImageAlign {
  iiaLeft,
  iiaCenter,
  iiaRight
};
```

File

ScreenTips

Description

This is record ScreenTips.TItemImageAlign.

3.3.4 ScreenTips.TScreenTipOption Enumeration

Pascal

```
TScreenTipOption = (
  soShowHeader,
  soShowShortCut
);
```

C++

```
enum TScreenTipOption {
  soShowHeader,
  soShowShortCut
};
```

File

ScreenTips

Description

This is record ScreenTips.TScreenTipOption.

3.3.5 ScreenTips.TScreenTipPopupType Enumeration

Pascal

```
TScreenTipPopupType = (
  ptCustom,
  ptHelp,
  ptInfo
);
```

C++

```
enum TScreenTipPopupType {
  ptCustom,
  ptHelp,
  ptInfo
};
```

File

ScreenTips

Description

This is record ScreenTips.TScreenTipPopupType.

3.3.6 ScreenTips.TTipBorderStyle Enumeration

Pascal

```
TTipBorderStyle = (
  hbsNormal,
  hbsRounded
);
```

C++

```
enum TTipBorderStyle {
  hbsNormal,
  hbsRounded
};
```

File

ScreenTips

Description

This is record ScreenTips.TTipBorderStyle.

3.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCreateScreenTipEvent (see page 320)	This is type ScreenTips.TCreateScreenTipEvent.
TDuplicateCreateErrorEvent (see page 320)	This is type ScreenTips.TDuplicateCreateErrorEvent.
TScreenTipOptions (see page 321)	This is type ScreenTips.TScreenTipOptions.
TScreenTipsWindowClass (see page 321)	This is type ScreenTips.TScreenTipsWindowClass.
TScreenTipWinClass (see page 321)	This is type ScreenTips.TScreenTipWinClass.
TShowScreenTipEvent (see page 322)	This is type ScreenTips.TShowScreenTipEvent.

3.4.1 ScreenTips.TCreateScreenTipEvent Type

Pascal

```
TCreateScreenTipEvent = procedure (Manager: TObject; Item: TScreenTipItem) of object;
```

C++

```
(Manager: TObject; Item: TScreenTipItem) ( TCreateScreenTipEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TCreateScreenTipEvent.

3.4.2 ScreenTips.TDuplicateCreateErrorEvent Type

Pascal

```
TDuplicateCreateErrorEvent = procedure (Manager: TObject; Action: TBasicAction) of object;
```

C++

```
(Manager: TObject; Action: TBasicAction) ( TDuplicateCreateErrorEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TDuplicateCreateErrorEvent.

3.4.3 ScreenTips.TScreenTipOptions Type

Pascal

```
TScreenTipOptions = set of TScreenTipOption;
```

C++

```
TScreenTipOption TScreenTipOptions;
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipOptions.

3.4.4 ScreenTips.TScreenTipsWindowClass Type

Pascal

```
TScreenTipsWindowClass = class of TScreenTipsWindow;
```

C++

```
TScreenTipsWindow TScreenTipsWindowClass;
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipsWindowClass.

3.4.5 ScreenTips.TScreenTipWinClass Type

Pascal

```
TScreenTipWinClass = procedure (Manager: TObject; var ScreenTipWinClass:
TScreenTipsWindowClass) of object;
```

C++

```
(Manager: TObject; var ScreenTipWinClass: TScreenTipsWindowClass) ( TScreenTipWinClass)();
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipWinClass.

3.4.6 ScreenTips.TShowScreenTipEvent Type

Pascal

```
TShowScreenTipEvent = procedure (Manager: TObject; Action: TBasicAction; var ShowScreenTip:
Boolean) of object;
```

C++

```
(Manager: TObject; Action: TBasicAction; var ShowScreenTip: Boolean) ( TShowScreenTipEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TShowScreenTipEvent.

3.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cDefaultBorderColor (see page 323)	This is constant ScreenTips.cDefaultBorderColor.
cDefaultEndColor (see page 323)	This is constant ScreenTips.cDefaultEndColor.
cDefaultStartColor (see page 324)	This is constant ScreenTips.cDefaultStartColor.
cScreenTipHeaderSpacing (see page 324)	This is constant ScreenTips.cScreenTipHeaderSpacing.
cScreenTipMinimumHeight (see page 324)	This is constant ScreenTips.cScreenTipMinimumHeight.
cScreenTipTextAndImageWidth (see page 325)	This is constant ScreenTips.cScreenTipTextAndImageWidth.
cScreenTipTextOnlyWidth (see page 325)	This is constant ScreenTips.cScreenTipTextOnlyWidth.

3.5.1 ScreenTips.cDefaultBorderColor Constant

Pascal

```
cDefaultBorderColor = $00767676;
```

C++

```
cDefaultBorderColor = 0x00767676;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultBorderColor.

3.5.2 ScreenTips.cDefaultEndColor Constant

Pascal

```
cDefaultEndColor = clBtnFace;
```

C++

```
cDefaultEndColor = clBtnFace;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultEndColor.

3.5.3 ScreenTips.cDefaultStartColor Constant

Pascal

```
cDefaultStartColor = clWhite;
```

C++

```
cDefaultStartColor = clWhite;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultStartColor.

3.5.4 ScreenTips.cScreenTipHeaderSpacing Constant

Pascal

```
cScreenTipHeaderSpacing = 14;
```

C++

```
cScreenTipHeaderSpacing = 14;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipHeaderSpacing.

3.5.5 ScreenTips.cScreenTipMinimumHeight Constant

Pascal

```
cScreenTipMinimumHeight = 50;
```

C++

```
cScreenTipMinimumHeight = 50;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipMinimumHeight.

3.5.6 ScreenTips.cScreenTipTextAndImageWidth Constant

Pascal

```
cScreenTipTextAndImageWidth = 318;
```

C++

```
cScreenTipTextAndImageWidth = 318;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipTextAndImageWidth.

3.5.7 ScreenTips.cScreenTipTextOnlyWidth Constant

Pascal

```
cScreenTipTextOnlyWidth = 210;
```

C++

```
cScreenTipTextOnlyWidth = 210;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipTextOnlyWidth.

4 SessColn Namespace

This is namespace SessColn.

Topics

Name	Description
Classes (see page 328)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 348)	The following table lists structs, records, enums in this documentation.
Types (see page 349)	The following table lists types in this documentation.
Variables (see page 350)	The following table lists variables in this documentation.

Classes

	Name	Description
	ESessionsException (see page 328)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 328)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 329)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 331)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 332)	This is class SessColn.TSessionItem.
	TSessions (see page 337)	This is class SessColn.TSessions.

Structs, Records, Enums

	Name	Description
	TSessionStatusFlag (see page 348)	This is record SessColn.TSessionStatusFlag.

Types

Name	Description
TSessionStatusFlags (see page 349)	This is type SessColn.TSessionStatusFlags.

Variables

Name	Description
Sessions (see page 350)	This is variable SessColn.Sessions.

4.1 Classes

The following table lists classes in this documentation.

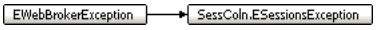
Classes

	Name	Description
	ESessionsException (see page 328)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 328)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 329)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 331)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 332)	This is class SessColn.TSessionItem.
	TSessions (see page 337)	This is class SessColn.TSessions.

4.1.1 ESessionsException Class

ESessionsException is the exception that is raised when TSessionsService encounters a problem.

Class Hierarchy



Pascal

```
ESessionsException = class(EWebBrokerException);
```

C++

```
class ESessionsException : public EWebBrokerException;
```

File

SessColn

Description

ESessionsException represents exceptions that occur when managing the Web sessions in a WebSnap application. For example, this exception is raised when the application tries to create a new session and the sessions service is already storing the maximum allowed number of sessions.

See Also

TSessionsService (see page 6722)

4.1.2 TAbstractSessionIDGen Class

Class Hierarchy

Pascal

```
TAbstractSessionIDGen = class;
```

C++

```
class TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionIDGen.

4.1.2.1 TAbstractSessionIDGen Members

The following tables list the members exposed by TAbstractSessionIDGen.

Public Methods

	Name	Description
	NextID (see page 329)	This is NextID, a member of class TAbstractSessionIDGen.

4.1.2.2 TAbstractSessionIDGen Methods

The methods of the TAbstractSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 329)	This is NextID, a member of class TAbstractSessionIDGen.

4.1.2.2.1 TAbstractSessionIDGen.NextID Method

Pascal

```
function NextID: string; virtual; abstract;
```

C++

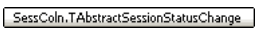
```
virtual __fastcall UnicodeString NextID() = 0;
```

Description

This is NextID, a member of class TAbstractSessionIDGen.

4.1.3 TAbstractSessionStatusChange Class

Class Hierarchy



Pascal

```
TAbstractSessionStatusChange = class;
```

C++

```
class TAbstractSessionStatusChange;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionStatusChange.

4.1.3.1 TAbstractSessionStatusChange Members

The following tables list the members exposed by TAbstractSessionStatusChange.

Protected Methods

	Name	Description
	SessionStarted (see page 330)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 330)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 331)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

4.1.3.2 TAbstractSessionStatusChange Methods

The methods of the TAbstractSessionStatusChange class are listed here.

Protected Methods

	Name	Description
	SessionStarted (see page 330)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 330)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 331)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

4.1.3.2.1 TAbstractSessionStatusChange.SessionStarted Method

Pascal

```
procedure SessionStarted(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionStarted(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionStarted, a member of class TAbstractSessionStatusChange.

4.1.3.2.2 TAbstractSessionStatusChange.SessionTerminated Method

Pascal

```
procedure SessionTerminated(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTerminated(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTerminated, a member of class TAbstractSessionStatusChange.

4.1.3.2.3 TAbstractSessionStatusChange.SessionTimeout Method

Pascal

```
procedure SessionTimeout(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTimeout(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTimeout, a member of class TAbstractSessionStatusChange.

4.1.4 TSessionIDGen Class

Class Hierarchy

Pascal

```
TSessionIDGen = class(TAbstractSessionIDGen);
```

C++

```
class TSessionIDGen : public TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TSessionIDGen.

4.1.4.1 TSessionIDGen Members

The following tables list the members exposed by TSessionIDGen.

Public Methods

	Name	Description
	Create (see page 331)	This is Create, a member of class TSessionIDGen.
V	NextID (see page 332)	This is NextID, a member of class TSessionIDGen.

4.1.4.2 TSessionIDGen.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessionIDGen();
```

Description

This is Create, a member of class TSessionIDGen.

4.1.4.3 TSessionIDGen Methods

The methods of the TSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 332)	This is NextID, a member of class TSessionIDGen.

4.1.4.3.1 TSessionIDGen.NextID Method

Pascal

```
function NextID: string; override;
```

C++

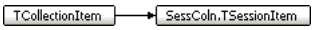
```
virtual __fastcall UnicodeString NextID();
```

Description

This is NextID, a member of class TSessionIDGen.

4.1.5 TSessionItem Class

Class Hierarchy



Pascal

```
TSessionItem = class(TCollectionItem);
```

C++

```
class TSessionItem : public TCollectionItem;
```

File

SessColn

Description

This is class SessColn.TSessionItem.

4.1.5.1 TSessionItem Members

The following tables list the members exposed by TSessionItem.

Public Methods

	Name	Description
	Assign (see page 334)	Copies the contents of another, similar object.
	Create (see page 333)	Creates and initializes a TCollectionItem instance.
	Destroy (see page 333)	Destroys the TCollectionItem instance and frees its memory.
	IsActive (see page 335)	This is IsActive, a member of class TSessionItem.

Public Properties

	Name	Description
	ExpirationTime (see page 335)	This is ExpirationTime, a member of class TSessionItem.

Published Properties

	Name	Description
	Expired (see page 335)	This is Expired, a member of class TSessionItem.
	ID (see page 336)	A unique, permanent index for the item.
	Items (see page 336)	This is Items, a member of class TSessionItem.
	StartTime (see page 336)	This is StartTime, a member of class TSessionItem.
	StatusFlags (see page 336)	This is StatusFlags, a member of class TSessionItem.
	Terminated (see page 337)	This is Terminated, a member of class TSessionItem.
	TimeoutMinutes (see page 337)	This is TimeoutMinutes, a member of class TSessionItem.
	TouchTime (see page 337)	This is TouchTime, a member of class TSessionItem.

4.1.5.2 TSessionItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TSessionItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

4.1.5.3 TSessionItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessionItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free (see page 3382)

4.1.5.4 TSessionItem Methods

The methods of the TSessionItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 334)	Copies the contents of another, similar object.
	IsActive (see page 335)	This is IsActive, a member of class TSessionItem.

4.1.5.4.1 TSessionItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError (see page 3985)

4.1.5.4.2 TSessionItem.IsActive Method

Pascal

```
function IsActive: Boolean;
```

C++

```
__fastcall Boolean IsActive();
```

Description

This is IsActive, a member of class TSessionItem.

4.1.5.5 TSessionItem Properties

The properties of the TSessionItem class are listed here.

Public Properties

	Name	Description
	ExpirationTime (see page 335)	This is ExpirationTime, a member of class TSessionItem.

Published Properties

	Name	Description
	Expired (see page 335)	This is Expired, a member of class TSessionItem.
	ID (see page 336)	A unique, permanent index for the item.
	Items (see page 336)	This is Items, a member of class TSessionItem.
	StartTime (see page 336)	This is StartTime, a member of class TSessionItem.
	StatusFlags (see page 336)	This is StatusFlags, a member of class TSessionItem.
	Terminated (see page 337)	This is Terminated, a member of class TSessionItem.
	TimeoutMinutes (see page 337)	This is TimeoutMinutes, a member of class TSessionItem.
	TouchTime (see page 337)	This is TouchTime, a member of class TSessionItem.

4.1.5.5.1 TSessionItem.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class TSessionItem.

4.1.5.5.2 TSessionItem.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class TSessionItem.

4.1.5.5.3 TSessionItem.ID Property

A unique, permanent index for the item.

Pascal

```
property ID: string;
```

C++

```
__property UnicodeString ID;
```

Description

In most cases, ID is the value of the item's Index property when it was first added to the collection. But, unlike Index values, ID values are never changed or reassigned. If an item is removed from a collection, its ID not used again.

An item's Index may change when other items are deleted from the collection or when items are rearranged. ID provides a unique identifier for each item that is unchanged for the life of the collection.

See Also

Index

4.1.5.5.4 TSessionItem.Items Property

Pascal

```
property Items: TAbstractNamedVariants;
```

C++

```
__property TAbstractNamedVariants Items;
```

Description

This is Items, a member of class TSessionItem.

4.1.5.5.5 TSessionItem.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class TSessionItem.

4.1.5.5.6 TSessionItem.StatusFlags Property

Pascal

```
property StatusFlags: TSessionStatusFlags;
```

C++

```
__property TSessionStatusFlags StatusFlags;
```

Description

This is StatusFlags, a member of class TSessionItem.

4.1.5.5.7 TSessionItem.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class TSessionItem.

4.1.5.5.8 TSessionItem.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class TSessionItem.

4.1.5.5.9 TSessionItem.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class TSessionItem.

4.1.6 TSessions Class

Class Hierarchy

Pascal

```
TSessions = class(TCollection);
```

C++

```
class TSessions : public TCollection;
```

File

SessColn

Description

This is class SessColn.TSessions.

4.1.6.1 TSessions Members

The following tables list the members exposed by TSessions.

Public Methods

	Name	Description
	CheckStatusChange (see page 340)	This is CheckStatusChange, a member of class TSessions.
	ConnectActiveSession (see page 340)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 340)	This is ConnectSession, a member of class TSessions.
	Create (see page 338)	Constructs an object and initializes its data before the object is first used.
V	CreateSessionIDGen (see page 341)	This is CreateSessionIDGen, a member of class TSessions.
V	Destroy (see page 339)	Destroys the collection and each item in it.
	DisconnectSession (see page 341)	This is DisconnectSession, a member of class TSessions.
	GetExpirationTime (see page 341)	This is GetExpirationTime, a member of class TSessions.
	GetExpired (see page 341)	This is GetExpired, a member of class TSessions.
	GetIDCount (see page 341)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 342)	This is GetIDs, a member of class TSessions.
	GetIsActive (see page 342)	This is GetIsActive, a member of class TSessions.
	GetItemCount (see page 342)	This is GetItemCount, a member of class TSessions.
	GetItems (see page 342)	This is GetItems, a member of class TSessions.
	GetSession (see page 343)	This is GetSession, a member of class TSessions.
	GetSessionIDGen (see page 343)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 343)	This is GetStartTime, a member of class TSessions.
	GetTerminated (see page 343)	This is GetTerminated, a member of class TSessions.
	GetTimeoutMinutes (see page 343)	This is GetTimeoutMinutes, a member of class TSessions.
	GetTouchTime (see page 344)	This is GetTouchTime, a member of class TSessions.
	ItemValue (see page 344)	This is ItemValue, a member of class TSessions.
	LoadFromFile (see page 344)	This is LoadFromFile, a member of class TSessions.
	LoadFromStream (see page 344)	This is LoadFromStream, a member of class TSessions.
	NotifyStatusChange (see page 345)	This is NotifyStatusChange, a member of class TSessions.
	SaveToFile (see page 345)	This is SaveToFile, a member of class TSessions.
	SaveToStream (see page 345)	This is SaveToStream, a member of class TSessions.
	SessionExists (see page 345)	This is SessionExists, a member of class TSessions.
	SetItemValue (see page 345)	This is SetItemValue, a member of class TSessions.
	SetTimeoutMinutes (see page 346)	This is SetTimeoutMinutes, a member of class TSessions.
	StartSession (see page 346)	This is StartSession, a member of class TSessions.
	TerminateSession (see page 346)	This is TerminateSession, a member of class TSessions.
	TouchSession (see page 346)	This is TouchSession, a member of class TSessions.

Public Properties

	Name	Description
R	ActiveCount (see page 347)	This is ActiveCount, a member of class TSessions.
	FileName (see page 347)	This is FileName, a member of class TSessions.
	MaxSessions (see page 347)	This is MaxSessions, a member of class TSessions.

4.1.6.2 TSessions.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessions();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

4.1.6.3 TSessions.Destroy Destructor

Destroys the collection and each item in it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessions();
```

Description

Destroy calls the Clear method to free each item referenced in the Items array, then destroys the collection itself.

See Also

Clear

System::TObject::Free (see page 3382)

4.1.6.4 TSessions Methods

The methods of the TSessions class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 340)	This is CheckStatusChange, a member of class TSessions.
	ConnectActiveSession (see page 340)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 340)	This is ConnectSession, a member of class TSessions.
	CreateSessionIDGen (see page 341)	This is CreateSessionIDGen, a member of class TSessions.
	DisconnectSession (see page 341)	This is DisconnectSession, a member of class TSessions.
	GetExpirationTime (see page 341)	This is GetExpirationTime, a member of class TSessions.
	GetExpired (see page 341)	This is GetExpired, a member of class TSessions.
	GetIDCount (see page 341)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 342)	This is GetIDs, a member of class TSessions.
	GetIsActive (see page 342)	This is GetIsActive, a member of class TSessions.
	GetItemCount (see page 342)	This is GetItemCount, a member of class TSessions.
	GetItems (see page 342)	This is GetItems, a member of class TSessions.
	GetSession (see page 343)	This is GetSession, a member of class TSessions.

	GetSessionIDGen (see page 343)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 343)	This is GetStartTime, a member of class TSessions.
	GetTerminated (see page 343)	This is GetTerminated, a member of class TSessions.
	GetTimeoutMinutes (see page 343)	This is GetTimeoutMinutes, a member of class TSessions.
	GetTouchTime (see page 344)	This is GetTouchTime, a member of class TSessions.
	ItemValue (see page 344)	This is ItemValue, a member of class TSessions.
	LoadFromFile (see page 344)	This is LoadFromFile, a member of class TSessions.
	LoadFromStream (see page 344)	This is LoadFromStream, a member of class TSessions.
	NotifyStatusChange (see page 345)	This is NotifyStatusChange, a member of class TSessions.
	SaveToFile (see page 345)	This is SaveToFile, a member of class TSessions.
	SaveToStream (see page 345)	This is SaveToStream, a member of class TSessions.
	SessionExists (see page 345)	This is SessionExists, a member of class TSessions.
	SetItemValue (see page 345)	This is SetItemValue, a member of class TSessions.
	SetTimeoutMinutes (see page 346)	This is SetTimeoutMinutes, a member of class TSessions.
	StartSession (see page 346)	This is StartSession, a member of class TSessions.
	TerminateSession (see page 346)	This is TerminateSession, a member of class TSessions.
	TouchSession (see page 346)	This is TouchSession, a member of class TSessions.

4.1.6.4.1 TSessions.CheckStatusChange Method

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

This is CheckStatusChange, a member of class TSessions.

4.1.6.4.2 TSessions.ConnectActiveSession Method

Pascal

```
function ConnectActiveSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectActiveSession(TSessionID AID);
```

Description

This is ConnectActiveSession, a member of class TSessions.

4.1.6.4.3 TSessions.ConnectSession Method

Pascal

```
function ConnectSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectSession(TSessionID AID);
```

Description

This is ConnectSession, a member of class TSessions.

4.1.6.4.4 TSessions.CreateSessionIDGen Method

Pascal

```
function CreateSessionIDGen: TAbstractSessionIDGen; virtual;
```

C++

```
virtual __fastcall TAbstractSessionIDGen CreateSessionIDGen();
```

Description

This is CreateSessionIDGen, a member of class TSessions.

4.1.6.4.5 TSessions.DisconnectSession Method

Pascal

```
procedure DisconnectSession(AID: TSessionID);
```

C++

```
__fastcall DisconnectSession(TSessionID AID);
```

Description

This is DisconnectSession, a member of class TSessions.

4.1.6.4.6 TSessions.GetExpirationTime Method

Pascal

```
function GetExpirationTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime(TSessionID AID);
```

Description

This is GetExpirationTime, a member of class TSessions.

4.1.6.4.7 TSessions.GetExpired Method

Pascal

```
function GetExpired(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetExpired(TSessionID AID);
```

Description

This is GetExpired, a member of class TSessions.

4.1.6.4.8 TSessions.GetIDCount Method

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

This is GetIDCount, a member of class TSessions.

4.1.6.4.9 TSessions.GetIDs Method

Pascal

```
procedure GetIDs(var AList: TList);
```

C++

```
__fastcall GetIDs(TList AList);
```

Description

This is GetIDs, a member of class TSessions.

4.1.6.4.10 TSessions.GetIsActive Method

Pascal

```
function GetIsActive(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetIsActive(TSessionID AID);
```

Description

This is GetIsActive, a member of class TSessions.

4.1.6.4.11 TSessions.GetItemCount Method

Pascal

```
function GetItemCount(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetItemCount(TSessionID AID);
```

Description

This is GetItemCount, a member of class TSessions.

4.1.6.4.12 TSessions.GetItems Method

Pascal

```
procedure GetItems(AID: TSessionID; var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TSessionID AID, TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class TSessions.

4.1.6.4.13 TSessions.GetSession Method

Pascal

```
function GetSession(AID: TSessionID; var AItem: TSessionItem): Boolean;
```

C++

```
__fastcall Boolean GetSession(TSessionID AID, TSessionItem AItem);
```

Description

This is GetSession, a member of class TSessions.

4.1.6.4.14 TSessions.GetSessionIDGen Method

Pascal

```
function GetSessionIDGen: TAbstractSessionIDGen;
```

C++

```
__fastcall TAbstractSessionIDGen GetSessionIDGen();
```

Description

This is GetSessionIDGen, a member of class TSessions.

4.1.6.4.15 TSessions.GetStartTime Method

Pascal

```
function GetStartTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime(TSessionID AID);
```

Description

This is GetStartTime, a member of class TSessions.

4.1.6.4.16 TSessions.GetTerminated Method

Pascal

```
function GetTerminated(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetTerminated(TSessionID AID);
```

Description

This is GetTerminated, a member of class TSessions.

4.1.6.4.17 TSessions.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetTimeoutMinutes(TSessionID AID);
```

Description

This is GetTimeoutMinutes, a member of class TSessions.

4.1.6.4.18 TSessions.GetTouchTime Method

Pascal

```
function GetTouchTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime(TSessionID AID);
```

Description

This is GetTouchTime, a member of class TSessions.

4.1.6.4.19 TSessions.ItemValue Method

Pascal

```
function ItemValue(AID: TSessionID; const AName: string): Variant;
```

C++

```
__fastcall Variant ItemValue(TSessionID AID, const UnicodeString AName);
```

Description

This is ItemValue, a member of class TSessions.

4.1.6.4.20 TSessions.LoadFromFile Method

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

This is LoadFromFile, a member of class TSessions.

4.1.6.4.21 TSessions.LoadFromStream Method

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

This is LoadFromStream, a member of class TSessions.

4.1.6.4.22 TSessions.NotifyStatusChange Method

Pascal

```
procedure NotifyStatusChange(AStatusChange: TAbstractSessionStatusChange);
```

C++

```
__fastcall NotifyStatusChange(TAbstractSessionStatusChange AStatusChange);
```

Description

This is NotifyStatusChange, a member of class TSessions.

4.1.6.4.23 TSessions.SaveToFile Method

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

This is SaveToFile, a member of class TSessions.

4.1.6.4.24 TSessions.SaveToStream Method

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

This is SaveToStream, a member of class TSessions.

4.1.6.4.25 TSessions.SessionExists Method

Pascal

```
function SessionExists(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean SessionExists(TSessionID AID);
```

Description

This is SessionExists, a member of class TSessions.

4.1.6.4.26 TSessions.SetItemValue Method

Pascal

```
procedure SetItemValue(AID: TSessionID; const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetItemValue(TSessionID AID, const UnicodeString AName, const Variant AValue);
```

Description

This is SetItemValue, a member of class TSessions.

4.1.6.4.27 TSessions.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AID: TSessionID; AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(TSessionID AID, int AValue);
```

Description

This is SetTimeoutMinutes, a member of class TSessions.

4.1.6.4.28 TSessions.StartSession Method

Pascal

```
function StartSession: TSessionID;
```

C++

```
__fastcall TSessionID StartSession();
```

Description

This is StartSession, a member of class TSessions.

4.1.6.4.29 TSessions.TerminateSession Method

Pascal

```
function TerminateSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TerminateSession(TSessionID AID);
```

Description

This is TerminateSession, a member of class TSessions.

4.1.6.4.30 TSessions.TouchSession Method

Pascal

```
function TouchSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TouchSession(TSessionID AID);
```

Description

This is TouchSession, a member of class TSessions.

4.1.6.5 TSessions Properties

The properties of the TSessions class are listed here.

Public Properties

	Name	Description
	ActiveCount (see page 347)	This is ActiveCount, a member of class TSessions.
	FileName (see page 347)	This is FileName, a member of class TSessions.
	MaxSessions (see page 347)	This is MaxSessions, a member of class TSessions.

4.1.6.5.1 TSessions.ActiveCount Property

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

This is ActiveCount, a member of class TSessions.

4.1.6.5.2 TSessions.FileName Property

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

This is FileName, a member of class TSessions.

4.1.6.5.3 TSessions.MaxSessions Property

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

This is MaxSessions, a member of class TSessions.

4.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSessionStatusFlag (see page 348)	This is record SessColn.TSessionStatusFlag.

4.2.1 SessColn.TSessionStatusFlag Enumeration

Pascal

```
TSessionStatusFlag = (
  ssStarted,
  ssTerminated,
  ssTimeOut
);
```

C++

```
enum TSessionStatusFlag {
  ssStarted,
  ssTerminated,
  ssTimeOut
};
```

File

SessColn

Description

This is record SessColn.TSessionStatusFlag.

4.3 Types

The following table lists types in this documentation.

Types

Name	Description
TSessionStatusFlags (see page 349)	This is type SessColn.TSessionStatusFlags.

4.3.1 SessColn.TSessionStatusFlags Type

Pascal

```
TSessionStatusFlags = set of TSessionStatusFlag;
```

C++

```
TSessionStatusFlag TSessionStatusFlags;
```

File

SessColn

Description

This is type SessColn.TSessionStatusFlags.

4.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Sessions (see page 350)	This is variable SessColn.Sessions.

4.4.1 SessColn.Sessions Variable

Pascal

```
Sessions: TSessions;
```

C++

```
TSessions Sessions;
```

File

SessColn

Description

This is variable SessColn.Sessions.

5 ShadowWnd Namespace

This is namespace ShadowWnd.

Topics

Name	Description
Classes (see page 352)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 358)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TShadowWindow (see page 352)	This is class ShadowWnd.TShadowWindow.

Structs, Records, Enums

	Name	Description
	TControlSide (see page 358)	This is record ShadowWnd.TControlSide.

5.1 Classes

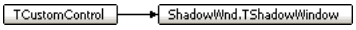
The following table lists classes in this documentation.

Classes

	Name	Description
	TShadowWindow (see page 352)	This is class ShadowWnd.TShadowWindow.

5.1.1 TShadowWindow Class

Class Hierarchy



Pascal

```
TShadowWindow = class(TCustomControl);
```

C++

```
class TShadowWindow : public TCustomControl;
```

File

ShadowWnd

Description

This is class ShadowWnd.TShadowWindow.

5.1.1.1 TShadowWindow Members

The following tables list the members exposed by TShadowWindow.

Protected Methods

	Name	Description
	CreateParams (see page 354)	Initializes a window-creation parameter data structure.
	Notification (see page 355)	Responds to notifications that components are being created or destroyed.
	Paint (see page 355)	Renders the image of a custom control.
	WMEraseBkgnd (see page 356)	This is WMEraseBkgnd, a member of class TShadowWindow.

Public Methods

	Name	Description
	Create (see page 353)	Creates an instance of TCustomControl.
	CreateShadow (see page 353)	This is CreateShadow, a member of class TShadowWindow.
	Destroy (see page 353)	Destroys an instance of TCustomControl.
	SetBounds (see page 356)	Sets the windowed control's boundary properties all at once.

Public Properties

	Name	Description
	Control (see page 357)	This is Control, a member of class TShadowWindow.

	Side (see page 357)	This is Side, a member of class TShadowWindow.

5.1.1.2 TShadowWindow.Create Constructor

Creates an instance of TCustomControl.

Pascal

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

C++

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

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

5.1.1.3 TShadowWindow.CreateShadow Constructor

Pascal

```
constructor CreateShadow(AOwner: TComponent; ControlSide: TControlSide); virtual;
```

C++

```
virtual __fastcall TShadowWindow(TComponent * AOwner, TControlSide ControlSide);
```

Description

This is CreateShadow, a member of class TShadowWindow.

5.1.1.4 TShadowWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShadowWindow();
```

Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3382)

delete

Create

5.1.1.5 TShadowWindow Methods

The methods of the TShadowWindow class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 354)	Initializes a window-creation parameter data structure.
	Notification (see page 355)	Responds to notifications that components are being created or destroyed.
	Paint (see page 355)	Renders the image of a custom control.
	WMEraseBkgnd (see page 356)	This is WMEraseBkgnd, a member of class TShadowWindow.

Public Methods

	Name	Description
	SetBounds (see page 356)	Sets the windowed control's boundary properties all at once.

5.1.1.5.1 TShadowWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override

CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

5.1.1.5.2 TShadowWindow.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.1.5.3 TShadowWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT

messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

5.1.1.5.4 TShadowWindow.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

5.1.1.5.5 TShadowWindow.WMEraseBkgnd Method

Pascal

```
procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
```

C++

```
__fastcall WMEraseBkgnd(TWMEraseBkgnd Message);
```

Description

This is WMEraseBkgnd, a member of class TShadowWindow.

5.1.1.6 TShadowWindow Properties

The properties of the TShadowWindow class are listed here.

Public Properties

	Name	Description
	Control (see page 357)	This is Control, a member of class TShadowWindow.
	Side (see page 357)	This is Side, a member of class TShadowWindow.

5.1.1.6.1 TShadowWindow.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TShadowWindow.

5.1.1.6.2 TShadowWindow.Side Property

Pascal

```
property Side: TControlSide;
```

C++

```
__property TControlSide Side;
```

Description

This is Side, a member of class TShadowWindow.

5.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TControlSide (see page 358)	This is record ShadowWnd.TControlSide.

5.2.1 ShadowWnd.TControlSide Enumeration

Pascal

```
TControlSide = (
  csRight,
  csBottom
);
```

C++

```
enum TControlSide {
  csRight,
  csBottom
};
```

File

ShadowWnd

Description

This is record ShadowWnd.TControlSide.

6 ShareMem Namespace

This is namespace ShareMem.

Topics

Name	Description
Functions (see page 360)	The following table lists functions in this documentation.

Functions

	Name	Description
	DumpBlocks (see page 360)	This is function ShareMem.DumpBlocks.
	GetAllocMemCount (see page 360)	This is function ShareMem.GetAllocMemCount.
	GetAllocMemSize (see page 361)	This is function ShareMem.GetAllocMemSize.
	GetHeapStatus (see page 361)	This is function ShareMem.GetHeapStatus.
	HeapAddRef (see page 361)	This is function ShareMem.HeapAddRef.
	HeapRelease (see page 362)	This is function ShareMem.HeapRelease.
	SysAllocMem (see page 362)	This is function ShareMem.SysAllocMem.
	SysFreeMem (see page 362)	This is function ShareMem.SysFreeMem.
	SysGetMem (see page 363)	This is function ShareMem.SysGetMem.
	SysReallocMem (see page 363)	This is function ShareMem.SysReallocMem.
	SysRegisterExpectedMemoryLeak (see page 363)	This is function ShareMem.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 364)	This is function ShareMem.SysUnregisterExpectedMemoryLeak.

6.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DumpBlocks (see page 360)	This is function ShareMem.DumpBlocks.
	GetAllocMemCount (see page 360)	This is function ShareMem.GetAllocMemCount.
	GetAllocMemSize (see page 361)	This is function ShareMem.GetAllocMemSize.
	GetHeapStatus (see page 361)	This is function ShareMem.GetHeapStatus.
	HeapAddRef (see page 361)	This is function ShareMem.HeapAddRef.
	HeapRelease (see page 362)	This is function ShareMem.HeapRelease.
	SysAllocMem (see page 362)	This is function ShareMem.SysAllocMem.
	SysFreeMem (see page 362)	This is function ShareMem.SysFreeMem.
	SysGetMem (see page 363)	This is function ShareMem.SysGetMem.
	SysReallocMem (see page 363)	This is function ShareMem.SysReallocMem.
	SysRegisterExpectedMemoryLeak (see page 363)	This is function ShareMem.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 364)	This is function ShareMem.SysUnregisterExpectedMemoryLeak.

6.1.1 ShareMem.DumpBlocks Function

Pascal

```
procedure DumpBlocks;
```

C++

```
DumpBlocks();
```

File

sharemem

Description

This is function ShareMem.DumpBlocks.

6.1.2 ShareMem.GetAllocMemCount Function

Pascal

```
function GetAllocMemCount: Integer;
```

C++

```
int GetAllocMemCount();
```

File

sharemem

Description

This is function ShareMem.GetAllocMemCount.

6.1.3 ShareMem.GetAllocMemSize Function

Pascal

```
function GetAllocMemSize: Integer;
```

C++

```
int GetAllocMemSize();
```

File

sharemem

Description

This is function ShareMem.GetAllocMemSize.

6.1.4 ShareMem.GetHeapStatus Function

Pascal

```
function GetHeapStatus: THeapStatus;
```

C++

```
THeapStatus GetHeapStatus();
```

File

sharemem

Description

This is function ShareMem.GetHeapStatus.

6.1.5 ShareMem.HeapAddRef Function

Pascal

```
procedure HeapAddRef;
```

C++

```
HeapAddRef();
```

File

sharemem

Description

This is function ShareMem.HeapAddRef.

6.1.6 ShareMem.HeapRelease Function

Pascal

```
procedure HeapRelease;
```

C++

```
HeapRelease();
```

File

sharemem

Description

This is function ShareMem.HeapRelease.

6.1.7 ShareMem.SysAllocMem Function

Pascal

```
function SysAllocMem(Size: Cardinal): Pointer;
```

C++

```
void * SysAllocMem(unsigned Size);
```

File

sharemem

Description

This is function ShareMem.SysAllocMem.

6.1.8 ShareMem.SysFreeMem Function

Pascal

```
function SysFreeMem(P: Pointer): Integer;
```

C++

```
int SysFreeMem(void * P);
```

File

sharemem

Description

This is function ShareMem.SysFreeMem.

6.1.9 ShareMem.SysGetMem Function

Pascal

```
function SysGetMem(Size: Integer): Pointer;
```

C++

```
void * SysGetMem(int Size);
```

File

sharemem

Description

This is function ShareMem.SysGetMem.

6.1.10 ShareMem.SysReallocMem Function

Pascal

```
function SysReallocMem(P: Pointer; Size: Integer): Pointer;
```

C++

```
void * SysReallocMem(void * P, int Size);
```

File

sharemem

Description

This is function ShareMem.SysReallocMem.

6.1.11 ShareMem.SysRegisterExpectedMemoryLeak Function

Pascal

```
function SysRegisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysRegisterExpectedMemoryLeak(void * P);
```

File

sharemem

Description

This is function ShareMem.SysRegisterExpectedMemoryLeak.

6.1.12 ShareMem.SysUnregisterExpectedMemoryLeak Function

Pascal

```
function SysUnregisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysUnregisterExpectedMemoryLeak(void * P);
```

File

sharemem

Description

This is function ShareMem.SysUnregisterExpectedMemoryLeak.

7 SHDocVw Namespace

This is namespace SHDocVw.

Topics

Name	Description
Classes (see page 371)	The following table lists classes in this documentation.
Interfaces (see page 571)	The following table lists interfaces in this documentation.
Types (see page 746)	The following table lists types in this documentation.
Constants (see page 780)	The following table lists constants in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 371)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 372)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 373)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 375)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 376)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 377)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 378)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 379)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 424)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 441)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 452)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 462)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 526)	This is class SHDocVw.TWebBrowser_V1.

Constants

Name	Description
CGID_IWebBrowser (see page 782)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 783)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 783)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 783)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 784)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 784)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 784)	This is constant SHDocVw.CLASS_ShellUIHelper.

CLASS_ShellWindows (see page 785)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 785)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 785)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 786)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 786)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 786)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 787)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 787)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 787)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 788)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 788)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 788)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 789)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 789)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 789)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 790)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 790)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 790)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 792)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 792)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 792)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 793)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 793)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 793)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 794)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 794)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 794)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 795)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 795)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 795)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 796)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 796)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 796)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.
OLECMDEXECOPT_DONTPROMPTUSER (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 798)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 798)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 798)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 799)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 799)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 799)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 800)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 800)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 800)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 801)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 801)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 801)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 802)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 802)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 802)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

OLECMDID_FIND (see page 803)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 803)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 803)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 804)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 804)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 804)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 805)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 805)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 805)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 806)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 806)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 806)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 807)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 807)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 807)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 808)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 808)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 808)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 809)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 809)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 809)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 810)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 810)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 810)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 811)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 811)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 811)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 812)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 812)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 812)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 813)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 813)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 813)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 814)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 814)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 814)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 815)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.
OLECMDID_SPELL (see page 815)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 815)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 816)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 816)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 816)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 817)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 817)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 817)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 818)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 818)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 818)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 819)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 819)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 819)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 820)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 820)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 820)	This is constant SHDocVw.secureLockIconMixed.

secureLockIconSecure128Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 822)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 822)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 822)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 823)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 823)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 823)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 824)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 824)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 824)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 825)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 825)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 825)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 571)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 573)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 574)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 576)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 581)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 590)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 594)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 598)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 601)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 611)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 613)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 621)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 624)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 632)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 639)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 641)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 643)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 645)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 650)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 654)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 663)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 671)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 675)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 679)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 683)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 687)	This is class SHDocVw.IWebBrowser.
	IWebBrowser2 (see page 698)	This is class SHDocVw.IWebBrowser2.
	IWebBrowser2Disp (see page 707)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 720)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 729)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 739)	This is class SHDocVw.IWebBrowserDisp.

Types

Name	Description
BrowserNavConstants (see page 748)	This is type SHDocVw.BrowserNavConstants.
CommandStateChangeConstants (see page 748)	This is type SHDocVw.CommandStateChangeConstants.
CScriptErrorList (see page 748)	This is type SHDocVw.CScriptErrorList.

InternetExplorer (see page 749)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 749)	OLECMDF (see page 750) influences the way in which an OLECMDID (see page 750) command is executed.
OLECMDF (see page 750)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 750)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 751)	This is type SHDocVw.POleVariant1.
RefreshConstants (see page 751)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 751)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 752)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 752)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 752)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 753)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 753)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 753)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 754)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 754)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 754)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 755)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 755)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 755)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 756)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 756)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 756)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 757)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 757)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 757)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 758)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 758)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 758)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 759)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 759)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 759)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 760)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 760)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 760)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 761)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 761)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 761)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 762)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 762)	This is type SHDocVw.TInternetExplorerWindowClosing.
TInternetExplorerWindowSetHeight (see page 762)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 764)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 764)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 764)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 765)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 765)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 765)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.
TWebBrowser_V1FrameNavigateComplete (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.

TWebBrowser_V1NavigateComplete (see page 767)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 767)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 768)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 768)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 768)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 769)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 769)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 769)	This is type SHDocVw.TWebBrowserBeforeNavigate2.
TWebBrowserClientToHostWindow (see page 770)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 770)	This is type SHDocVw.TWebBrowserCommandStateChange.
TWebBrowserDocumentComplete (see page 770)	This is type SHDocVw.TWebBrowserDocumentComplete.
TWebBrowserFileDownload (see page 771)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 771)	This is type SHDocVw.TWebBrowserNavigateComplete2.
TWebBrowserNavigateError (see page 771)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 772)	This is type SHDocVw.TWebBrowserNewWindow2.
TWebBrowserOnFullScreen (see page 772)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 772)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 773)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 773)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 773)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 774)	This is type SHDocVw.TWebBrowserOnVisible.
TWebBrowserPrintTemplateInstantiation (see page 774)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 774)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 775)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 775)	This is type SHDocVw.TWebBrowserProgressChange.
TWebBrowserPropertyChange (see page 775)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 776)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 776)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 776)	This is type SHDocVw.TWebBrowserTitleChange.
TWebBrowserUpdatePageStatus (see page 777)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 777)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 777)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 778)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 778)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 778)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 779)	This is type SHDocVw.TWebBrowserWindowSetWidth.
WebBrowser (see page 779)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 779)	This is type SHDocVw.WebBrowser_V1.

7.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 371)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 372)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 373)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 375)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 376)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 377)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 378)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 379)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 424)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 441)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 452)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 462)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 526)	This is class SHDocVw.TWebBrowser_V1.

7.1.1 CoCScriptErrorList Class

Class Hierarchy

Pascal

```
CoCScriptErrorList = class;
```

C++

```
class CoCScriptErrorList;
```

File

SHDocVw

Description

This is class SHDocVw.CoCScriptErrorList.

7.1.1.1 CoCScriptErrorList Members

The following tables list the members exposed by CoCScriptErrorList.

Public Methods

	Name	Description
	Create (see page 372)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 372)	This is CreateRemote, a member of class CoCScriptErrorList.

7.1.1.2 CoCScriptErrorList Methods

The methods of the CoCScriptErrorList class are listed here.

Public Methods

	Name	Description
	Create (see page 372)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 372)	This is CreateRemote, a member of class CoCScriptErrorList.

7.1.1.2.1 CoCScriptErrorList.Create Method

Pascal

```
class function Create: IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList Create();
```

Description

This is Create, a member of class CoCScriptErrorList.

7.1.1.2.2 CoCScriptErrorList.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoCScriptErrorList.

7.1.2 CoInternetExplorer Class

Class Hierarchy

SHDocVw.CoInternetExplorer

Pascal

```
CoInternetExplorer = class;
```

C++

```
class CoInternetExplorer;
```

File

SHDocVw

Description

This is class SHDocVw.CoInternetExplorer.

7.1.2.1 CoInternetExplorer Members

The following tables list the members exposed by CoInternetExplorer.

Public Methods

	Name	Description
	Create (see page 373)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 373)	This is CreateRemote, a member of class CoInternetExplorer.

7.1.2.2 CoInternetExplorer Methods

The methods of the CoInternetExplorer class are listed here.

Public Methods

	Name	Description
	Create (see page 373)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 373)	This is CreateRemote, a member of class CoInternetExplorer.

7.1.2.2.1 CoInternetExplorer.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoInternetExplorer.

7.1.2.2.2 CoInternetExplorer.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoInternetExplorer.

7.1.3 CoSearchAssistantOC Class

Class Hierarchy

Pascal

```
CoSearchAssistantOC = class;
```

C++

```
class CoSearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.CoSearchAssistantOC.

7.1.3.1 CoSearchAssistantOC Members

The following tables list the members exposed by CoSearchAssistantOC.

Public Methods

	Name	Description
	Create (see page 374)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 374)	This is CreateRemote, a member of class CoSearchAssistantOC.

7.1.3.2 CoSearchAssistantOC Methods

The methods of the CoSearchAssistantOC class are listed here.

Public Methods

	Name	Description
	Create (see page 374)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 374)	This is CreateRemote, a member of class CoSearchAssistantOC.

7.1.3.2.1 CoSearchAssistantOC.Create Method

Pascal

```
class function Create: ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 Create();
```

Description

This is Create, a member of class CoSearchAssistantOC.

7.1.3.2.2 CoSearchAssistantOC.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoSearchAssistantOC.

7.1.4 CoShellBrowserWindow Class

Class Hierarchy

Pascal

```
CoShellBrowserWindow = class;
```

C++

```
class CoShellBrowserWindow;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellBrowserWindow.

7.1.4.1 CoShellBrowserWindow Members

The following tables list the members exposed by CoShellBrowserWindow.

Public Methods

	Name	Description
	Create (see page 375)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 376)	This is CreateRemote, a member of class CoShellBrowserWindow.

7.1.4.2 CoShellBrowserWindow Methods

The methods of the CoShellBrowserWindow class are listed here.

Public Methods

	Name	Description
	Create (see page 375)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 376)	This is CreateRemote, a member of class CoShellBrowserWindow.

7.1.4.2.1 CoShellBrowserWindow.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoShellBrowserWindow.

7.1.4.2.2 CoShellBrowserWindow.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellBrowserWindow.

7.1.5 CoShellNameSpace Class

Class Hierarchy

Pascal

```
CoShellNameSpace = class;
```

C++

```
class CoShellNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellNameSpace.

7.1.5.1 CoShellNameSpace Members

The following tables list the members exposed by CoShellNameSpace.

Public Methods

	Name	Description
	Create (see page 377)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 377)	This is CreateRemote, a member of class CoShellNameSpace.

7.1.5.2 CoShellNameSpace Methods

The methods of the CoShellNameSpace class are listed here.

Public Methods

	Name	Description
	Create (see page 377)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 377)	This is CreateRemote, a member of class CoShellNameSpace.

7.1.5.2.1 CoShellNameSpace.Create Method

Pascal

```
class function Create: IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace Create();
```

Description

This is Create, a member of class CoShellNameSpace.

7.1.5.2.2 CoShellNameSpace.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellNameSpace.

7.1.6 CoShellUIHelper Class

Class Hierarchy

SHDocVw.CoShellUIHelper

Pascal

```
CoShellUIHelper = class;
```

C++

```
class CoShellUIHelper;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellUIHelper.

7.1.6.1 CoShellUIHelper Members

The following tables list the members exposed by CoShellUIHelper.

Public Methods

	Name	Description
	Create (see page 378)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 378)	This is CreateRemote, a member of class CoShellUIHelper.

7.1.6.2 CoShellUIHelper Methods

The methods of the CoShellUIHelper class are listed here.

Public Methods

	Name	Description
	Create (see page 378)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 378)	This is CreateRemote, a member of class CoShellUIHelper.

7.1.6.2.1 CoShellUIHelper.Create Method

Pascal

```
class function Create: IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper Create();
```

Description

This is Create, a member of class CoShellUIHelper.

7.1.6.2.2 CoShellUIHelper.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellUIHelper.

7.1.7 CoShellWindows Class

Class Hierarchy

SHDocVw.CoShellWindows

Pascal

```
CoShellWindows = class;
```

C++

```
class CoShellWindows;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellWindows.

7.1.7.1 CoShellWindows Members

The following tables list the members exposed by CoShellWindows.

Public Methods

	Name	Description
	Create (see page 379)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 379)	This is CreateRemote, a member of class CoShellWindows.

7.1.7.2 CoShellWindows Methods

The methods of the CoShellWindows class are listed here.

Public Methods

	Name	Description
	Create (see page 379)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 379)	This is CreateRemote, a member of class CoShellWindows.

7.1.7.2.1 CoShellWindows.Create Method

Pascal

```
class function Create: IShellWindows;
```

C++

```
__fastcall IShellWindows Create();
```

Description

This is Create, a member of class CoShellWindows.

7.1.7.2.2 CoShellWindows.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellWindows;
```

C++

```
__fastcall IShellWindows CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellWindows.

7.1.8 TInternetExplorer Class

Class Hierarchy

Pascal

```
TInternetExplorer = class(TOleServer);
```

C++

```
class TInternetExplorer : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TInternetExplorer.

7.1.8.1 TInternetExplorer Members

The following tables list the members exposed by TInternetExplorer.

Protected Methods

	Name	Description
	Get_AddressBar (see page 388)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 388)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 388)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 389)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 389)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 389)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 389)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 389)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 390)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 390)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 390)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 390)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 391)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 391)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 391)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 391)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 391)	This is Get_Path, a member of class TInternetExplorer.
	Get_ReadyState (see page 392)	This is Get_ReadyState, a member of class TInternetExplorer.
	Get_RegisterAsBrowser (see page 392)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 392)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 392)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 393)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 393)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 393)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 393)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 393)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 394)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 394)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 394)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 394)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 395)	This is Get_Width, a member of class TInternetExplorer.
	InitServerData (see page 396)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 396)	Dispatches an event from the COM server to the appropriate event handler.
	Set_AddressBar (see page 401)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 402)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 402)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 402)	This is Set_Left, a member of class TInternetExplorer.

	Set_MenuBar (see page 402)	This is Set_MenuBar, a member of class TInternetExplorer.
	Set_Offline (see page 402)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 403)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 403)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 403)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 403)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 404)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 404)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 404)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 404)	This is Set_ToolBar, a member of class TInternetExplorer.
	Set_Top (see page 404)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 405)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 405)	This is Set_Width, a member of class TInternetExplorer.

Public Methods

	Name	Description
	ClientToWindow (see page 386)	This is ClientToWindow, a member of class TInternetExplorer.
V	Connect (see page 386)	Establishes a connection to the COM server.
	ConnectTo (see page 386)	This is ConnectTo, a member of class TInternetExplorer.
V	Create (see page 383)	Allocates memory and constructs a safely initialized instance of a component.
V	Destroy (see page 383)	Disposes of the component and its owned components.
V	Disconnect (see page 387)	Terminates the connection to the COM server.
	ExecWB (see page 387)	This is the overview for the ExecWB method overload.
	GetProperty (see page 395)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 395)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 395)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 395)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 396)	This is GoSearch, a member of class TInternetExplorer.
	Navigate (see page 397)	This is the overview for the Navigate method overload.
	Navigate2 (see page 398)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 400)	This is PutProperty, a member of class TInternetExplorer.
	QueryStatusWB (see page 400)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 400)	This is Quit, a member of class TInternetExplorer.
	Refresh (see page 400)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 401)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 405)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 406)	This is Stop, a member of class TInternetExplorer.

Public Properties

	Name	Description
	AddressBar (see page 407)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 407)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 407)	This is Busy, a member of class TInternetExplorer.
R	Container (see page 408)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 408)	This is DefaultInterface, a member of class TInternetExplorer.
R	Document (see page 408)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 408)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 408)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 409)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 409)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 409)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 409)	This is LocationName, a member of class TInternetExplorer.
R	LocationURL (see page 410)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 410)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 410)	Specifies the name of the component as referenced in code.

	Offline (see page 412)	This is Offline, a member of class TInternetExplorer.
R	Parent (see page 412)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 413)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 413)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 413)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 413)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 413)	This is Resizable, a member of class TInternetExplorer.
	Silent (see page 414)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 414)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 414)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 414)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 415)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 415)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 415)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 415)	This is type_, a member of class TInternetExplorer.
	Visible (see page 415)	This is Visible, a member of class TInternetExplorer.
	Width (see page 416)	This is Width, a member of class TInternetExplorer.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 417)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 417)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 417)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 417)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 417)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 418)	This is OnDownloadComplete, a member of class TInternetExplorer.
	OnFileDownload (see page 418)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 418)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 418)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 419)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 419)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 419)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 419)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 419)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 420)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 420)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 420)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 420)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 421)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 421)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 421)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 421)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 421)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 422)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 422)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 422)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 422)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 423)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 423)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 423)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 423)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 423)	This is OnWindowSetWidth, a member of class TInternetExplorer.

7.1.8.2 TInternetExplorer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3366)

Destroy

System::TObject::Free (see page 3382)

Owner

7.1.8.3 TInternetExplorer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInternetExplorer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3366)

DestroyComponents

Destroying

System::TObject::Free (see page 3382)

Notification

Owner

Release

Components

7.1.8.4 TInternetExplorer Methods

The methods of the TInternetExplorer class are listed here.

Protected Methods

	Name	Description
	Get_AddressBar (see page 388)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 388)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 388)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 389)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 389)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 389)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 389)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 389)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 390)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 390)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 390)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 390)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 391)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 391)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 391)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 391)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 391)	This is Get_Path, a member of class TInternetExplorer.
	Get_ReadyState (see page 392)	This is Get_ReadyState, a member of class TInternetExplorer.

	Get_RegisterAsBrowser (see page 392)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 392)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 392)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 393)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 393)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 393)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 393)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 393)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 394)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 394)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 394)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 394)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 395)	This is Get_Width, a member of class TInternetExplorer.
	InitServerData (see page 396)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 396)	Dispatches an event from the COM server to the appropriate event handler.
	Set_AddressBar (see page 401)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 402)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 402)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 402)	This is Set_Left, a member of class TInternetExplorer.
	Set_MenuBar (see page 402)	This is Set_MenuBar, a member of class TInternetExplorer.
	Set_Offline (see page 402)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 403)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 403)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 403)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 403)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 404)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 404)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 404)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 404)	This is Set_ToolBar, a member of class TInternetExplorer.
	Set_Top (see page 404)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 405)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 405)	This is Set_Width, a member of class TInternetExplorer.

Public Methods

	Name	Description
	ClientToWindow (see page 386)	This is ClientToWindow, a member of class TInternetExplorer.
	Connect (see page 386)	Establishes a connection to the COM server.
	ConnectTo (see page 386)	This is ConnectTo, a member of class TInternetExplorer.
	Disconnect (see page 387)	Terminates the connection to the COM server.
	ExecWB (see page 387)	This is the overview for the ExecWB method overload.
	GetProperty (see page 395)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 395)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 395)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 395)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 396)	This is GoSearch, a member of class TInternetExplorer.
	Navigate (see page 397)	This is the overview for the Navigate method overload.
	Navigate2 (see page 398)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 400)	This is PutProperty, a member of class TInternetExplorer.
	QueryStatusWB (see page 400)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 400)	This is Quit, a member of class TInternetExplorer.
	Refresh (see page 400)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 401)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 405)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 406)	This is Stop, a member of class TInternetExplorer.

7.1.8.4.1 TInternetExplorer.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class TInternetExplorer.

7.1.8.4.2 TInternetExplorer.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

7.1.8.4.3 TInternetExplorer.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IWebBrowser2);
```

C++

```
__fastcall ConnectTo(IWebBrowser2 svrIntf);
```

Description

This is ConnectTo, a member of class TInternetExplorer.

7.1.8.4.4 TInternetExplorer.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

7.1.8.4.5 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 387)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 387)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 388)	This is ExecWB, a member of class TInternetExplorer.

7.1.8.4.5.1 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
```

Description

This is ExecWB, a member of class TInternetExplorer.

7.1.8.4.5.2 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
```

Description

This is ExecWB, a member of class TInternetExplorer.

7.1.8.4.5.3 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class TInternetExplorer.

7.1.8.4.6 TInternetExplorer.Get_AddressBar Method

Pascal

```
function Get_AddressBar: WordBool;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

This is Get_AddressBar, a member of class TInternetExplorer.

7.1.8.4.7 TInternetExplorer.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TInternetExplorer.

7.1.8.4.8 TInternetExplorer.Get_Busy Method

Pascal

```
function Get_Busy: WordBool;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

This is Get_Busy, a member of class TInternetExplorer.

7.1.8.4.9 TInternetExplorer.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TInternetExplorer.

7.1.8.4.10 TInternetExplorer.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TInternetExplorer.

7.1.8.4.11 TInternetExplorer.Get_FullName Method

Pascal

```
function Get_FullName: WideString;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class TInternetExplorer.

7.1.8.4.12 TInternetExplorer.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class TInternetExplorer.

7.1.8.4.13 TInternetExplorer.Get_Height Method

Pascal

```
function Get_Height: Integer;
```

C++

```
__fastcall int Get_Height();
```

Description

This is Get_Height, a member of class TInternetExplorer.

7.1.8.4.14 TInternetExplorer.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class TInternetExplorer.

7.1.8.4.15 TInternetExplorer.Get_Left Method

Pascal

```
function Get_Left: Integer;
```

C++

```
__fastcall int Get_Left();
```

Description

This is Get_Left, a member of class TInternetExplorer.

7.1.8.4.16 TInternetExplorer.Get_LocationName Method

Pascal

```
function Get_LocationName: WideString;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

This is Get_LocationName, a member of class TInternetExplorer.

7.1.8.4.17 TInternetExplorer.Get_LocationURL Method

Pascal

```
function Get_LocationURL: WideString;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

This is Get_LocationURL, a member of class TInternetExplorer.

7.1.8.4.18 TInternetExplorer.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class TInternetExplorer.

7.1.8.4.19 TInternetExplorer.Get_Name Method

Pascal

```
function Get_Name: WideString;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class TInternetExplorer.

7.1.8.4.20 TInternetExplorer.Get_Offline Method

Pascal

```
function Get_Offline: WordBool;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

This is Get_Offline, a member of class TInternetExplorer.

7.1.8.4.21 TInternetExplorer.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TInternetExplorer.

7.1.8.4.22 TInternetExplorer.Get_Path Method

Pascal

```
function Get_Path: WideString;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class TInternetExplorer.

7.1.8.4.23 TInternetExplorer.Get_ReadyState Method

Pascal

```
function Get_ReadyState: tagREADYSTATE;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

This is Get_ReadyState, a member of class TInternetExplorer.

7.1.8.4.24 TInternetExplorer.Get_RegisterAsBrowser Method

Pascal

```
function Get_RegisterAsBrowser: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

This is Get_RegisterAsBrowser, a member of class TInternetExplorer.

7.1.8.4.25 TInternetExplorer.Get_RegisterAsDropTarget Method

Pascal

```
function Get_RegisterAsDropTarget: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.

7.1.8.4.26 TInternetExplorer.Get_Resizable Method

Pascal

```
function Get_Resizable: WordBool;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

This is Get_Resizable, a member of class TInternetExplorer.

7.1.8.4.27 TInternetExplorer.Get_Silent Method

Pascal

```
function Get_Silent: WordBool;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

This is Get_Silent, a member of class TInternetExplorer.

7.1.8.4.28 TInternetExplorer.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class TInternetExplorer.

7.1.8.4.29 TInternetExplorer.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class TInternetExplorer.

7.1.8.4.30 TInternetExplorer.Get_TheaterMode Method

Pascal

```
function Get_TheaterMode: WordBool;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

This is Get_TheaterMode, a member of class TInternetExplorer.

7.1.8.4.31 TInternetExplorer.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class TInternetExplorer.

7.1.8.4.32 TInternetExplorer.Get_Top Method

Pascal

```
function Get_Top: Integer;
```

C++

```
__fastcall int Get_Top();
```

Description

This is Get_Top, a member of class TInternetExplorer.

7.1.8.4.33 TInternetExplorer.Get_TopLevelContainer Method

Pascal

```
function Get_TopLevelContainer: WordBool;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

This is Get_TopLevelContainer, a member of class TInternetExplorer.

7.1.8.4.34 TInternetExplorer.Get_type_ Method

Pascal

```
function Get_type_: WideString;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

This is Get_type_, a member of class TInternetExplorer.

7.1.8.4.35 TInternetExplorer.Get_Visible Method

Pascal

```
function Get_Visible: WordBool;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class TInternetExplorer.

7.1.8.4.36 TInternetExplorer.Get_Width Method

Pascal

```
function Get_Width: Integer;
```

C++

```
__fastcall int Get_Width();
```

Description

This is Get_Width, a member of class TInternetExplorer.

7.1.8.4.37 TInternetExplorer.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class TInternetExplorer.

7.1.8.4.38 TInternetExplorer.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TInternetExplorer.

7.1.8.4.39 TInternetExplorer.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TInternetExplorer.

7.1.8.4.40 TInternetExplorer.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TInternetExplorer.

7.1.8.4.41 TInternetExplorer.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TInternetExplorer.

7.1.8.4.42 TInternetExplorer.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

7.1.8.4.43 TInternetExplorer.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the

appropriate event handlers.

See Also

EventDispatch

7.1.8.4.44 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate (WideString) (see page 397)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant) (see page 397)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant) (see page 397)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 398)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 398)	This is Navigate, a member of class TInternetExplorer.

7.1.8.4.44.1 TInternetExplorer.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TInternetExplorer.

7.1.8.4.44.2 TInternetExplorer.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TInternetExplorer.

7.1.8.4.44.3 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TInternetExplorer.

7.1.8.4.44.4 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TInternetExplorer.

7.1.8.4.44.5 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TInternetExplorer.

7.1.8.4.45 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate2 (OleVariant) (see page 399)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant) (see page 399)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 399)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 399)	This is Navigate2, a member of class TInternetExplorer.

	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 400)	This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.45.1 TInternetExplorer.Navigate2 Method (OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
```

Description

This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.45.2 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
```

Description

This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.45.3 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
```

Description

This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.45.4 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
```

Description

This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.45.5 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

This is Navigate2, a member of class TInternetExplorer.

7.1.8.4.46 TInternetExplorer.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class TInternetExplorer.

7.1.8.4.47 TInternetExplorer.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class TInternetExplorer.

7.1.8.4.48 TInternetExplorer.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class TInternetExplorer.

7.1.8.4.49 TInternetExplorer.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class TInternetExplorer.

7.1.8.4.50 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Refresh2 () (see page 401)	This is Refresh2, a member of class TInternetExplorer.
	TInternetExplorer.Refresh2 (OleVariant) (see page 401)	This is Refresh2, a member of class TInternetExplorer.

7.1.8.4.50.1 TInternetExplorer.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TInternetExplorer.

7.1.8.4.50.2 TInternetExplorer.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TInternetExplorer.

7.1.8.4.51 TInternetExplorer.Set_AddressBar Method

Pascal

```
procedure Set_AddressBar(Value: WordBool);
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

This is Set_AddressBar, a member of class TInternetExplorer.

7.1.8.4.52 TInternetExplorer.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool);
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class TInternetExplorer.

7.1.8.4.53 TInternetExplorer.Set_Height Method

Pascal

```
procedure Set_Height(pl: Integer);
```

C++

```
__fastcall Set_Height(int pl);
```

Description

This is Set_Height, a member of class TInternetExplorer.

7.1.8.4.54 TInternetExplorer.Set_Left Method

Pascal

```
procedure Set_Left(pl: Integer);
```

C++

```
__fastcall Set_Left(int pl);
```

Description

This is Set_Left, a member of class TInternetExplorer.

7.1.8.4.55 TInternetExplorer.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool);
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class TInternetExplorer.

7.1.8.4.56 TInternetExplorer.Set_Offline Method

Pascal

```
procedure Set_Offline(pbOffline: WordBool);
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

This is Set_Offline, a member of class TInternetExplorer.

7.1.8.4.57 TInternetExplorer.Set_RegisterAsBrowser Method

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

This is Set_RegisterAsBrowser, a member of class TInternetExplorer.

7.1.8.4.58 TInternetExplorer.Set_RegisterAsDropTarget Method

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.

7.1.8.4.59 TInternetExplorer.Set_Resizable Method

Pascal

```
procedure Set_Resizable(Value: WordBool);
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

This is Set_Resizable, a member of class TInternetExplorer.

7.1.8.4.60 TInternetExplorer.Set_Silent Method

Pascal

```
procedure Set_Silent(pbSilent: WordBool);
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

This is Set_Silent, a member of class TInternetExplorer.

7.1.8.4.61 TInternetExplorer.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool);
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class TInternetExplorer.

7.1.8.4.62 TInternetExplorer.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString);
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class TInternetExplorer.

7.1.8.4.63 TInternetExplorer.Set_TheaterMode Method

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool);
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

This is Set_TheaterMode, a member of class TInternetExplorer.

7.1.8.4.64 TInternetExplorer.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT);
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class TInternetExplorer.

7.1.8.4.65 TInternetExplorer.Set_Top Method

Pascal

```
procedure Set_Top(pl: Integer);
```

C++

```
__fastcall Set_Top(int pl);
```

Description

This is Set_Top, a member of class TInternetExplorer.

7.1.8.4.66 TInternetExplorer.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool);
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class TInternetExplorer.

7.1.8.4.67 TInternetExplorer.Set_Width Method

Pascal

```
procedure Set_Width(pl: Integer);
```

C++

```
__fastcall Set_Width(int pl);
```

Description

This is Set_Width, a member of class TInternetExplorer.

7.1.8.4.68 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TInternetExplorer.ShowBrowserBar (OleVariant) (see page 405)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant) (see page 406)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 406)	This is ShowBrowserBar, a member of class TInternetExplorer.

7.1.8.4.68.1 TInternetExplorer.ShowBrowserBar Method (OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

7.1.8.4.68.2 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

7.1.8.4.68.3 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

7.1.8.4.69 TInternetExplorer.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TInternetExplorer.

7.1.8.5 TInternetExplorer Properties

The properties of the TInternetExplorer class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 407)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 407)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 407)	This is Busy, a member of class TInternetExplorer.
R	Container (see page 408)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 408)	This is DefaultInterface, a member of class TInternetExplorer.
R	Document (see page 408)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 408)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 408)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 409)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 409)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 409)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 409)	This is LocationName, a member of class TInternetExplorer.

R	LocationURL (see page 410)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 410)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 410)	Specifies the name of the component as referenced in code.
	Offline (see page 412)	This is Offline, a member of class TInternetExplorer.
R	Parent (see page 412)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 413)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 413)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 413)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 413)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 413)	This is Resizable, a member of class TInternetExplorer.
	Silent (see page 414)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 414)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 414)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 414)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 415)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 415)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 415)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 415)	This is type_, a member of class TInternetExplorer.
	Visible (see page 415)	This is Visible, a member of class TInternetExplorer.
	Width (see page 416)	This is Width, a member of class TInternetExplorer.

7.1.8.5.1 TInternetExplorer.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class TInternetExplorer.

7.1.8.5.2 TInternetExplorer.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TInternetExplorer.

7.1.8.5.3 TInternetExplorer.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TInternetExplorer.

7.1.8.5.4 TInternetExplorer.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TInternetExplorer.

7.1.8.5.5 TInternetExplorer.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

This is DefaultInterface, a member of class TInternetExplorer.

7.1.8.5.6 TInternetExplorer.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TInternetExplorer.

7.1.8.5.7 TInternetExplorer.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class TInternetExplorer.

7.1.8.5.8 TInternetExplorer.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class TInternetExplorer.

7.1.8.5.9 TInternetExplorer.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TInternetExplorer.

7.1.8.5.10 TInternetExplorer.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class TInternetExplorer.

7.1.8.5.11 TInternetExplorer.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class TInternetExplorer.

7.1.8.5.12 TInternetExplorer.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TInternetExplorer.

7.1.8.5.13 TInternetExplorer.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TInternetExplorer.

7.1.8.5.14 TInternetExplorer.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class TInternetExplorer.

7.1.8.5.15 TInternetExplorer.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
```

```
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
```

```
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
  end;
end;
```

See Also

ChangeName

SetName

ValidateRename

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

7.1.8.5.16 TInternetExplorer.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class TInternetExplorer.

7.1.8.5.17 TInternetExplorer.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class TInternetExplorer.

7.1.8.5.18 TInternetExplorer.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class TInternetExplorer.

7.1.8.5.19 TInternetExplorer.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class TInternetExplorer.

7.1.8.5.20 TInternetExplorer.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class TInternetExplorer.

7.1.8.5.21 TInternetExplorer.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class TInternetExplorer.

7.1.8.5.22 TInternetExplorer.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class TInternetExplorer.

7.1.8.5.23 TInternetExplorer.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class TInternetExplorer.

7.1.8.5.24 TInternetExplorer.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class TInternetExplorer.

7.1.8.5.25 TInternetExplorer.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class TInternetExplorer.

7.1.8.5.26 TInternetExplorer.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class TInternetExplorer.

7.1.8.5.27 TInternetExplorer.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class TInternetExplorer.

7.1.8.5.28 TInternetExplorer.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class TInternetExplorer.

7.1.8.5.29 TInternetExplorer.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TInternetExplorer.

7.1.8.5.30 TInternetExplorer.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TInternetExplorer.

7.1.8.5.31 TInternetExplorer.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class TInternetExplorer.

7.1.8.5.32 TInternetExplorer.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TInternetExplorer.

7.1.8.6 TInternetExplorer Events

The events of the TInternetExplorer class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 417)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 417)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 417)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 417)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 417)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 418)	This is OnDownloadComplete, a member of class TInternetExplorer.
	OnFileDownload (see page 418)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 418)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 418)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 419)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 419)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 419)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 419)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 419)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 420)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 420)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 420)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 420)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 421)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 421)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 421)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 421)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 421)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 422)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 422)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 422)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 422)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 423)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 423)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 423)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 423)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 423)	This is OnWindowSetWidth, a member of class TInternetExplorer.

7.1.8.6.1 TInternetExplorer.OnBeforeNavigate2 Event

Pascal

```
property OnBeforeNavigate2: TInternetExplorerBeforeNavigate2;
```

C++

```
__property TInternetExplorerBeforeNavigate2 OnBeforeNavigate2;
```

Description

This is OnBeforeNavigate2, a member of class TInternetExplorer.

7.1.8.6.2 TInternetExplorer.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TInternetExplorerClientToHostWindow;
```

C++

```
__property TInternetExplorerClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TInternetExplorer.

7.1.8.6.3 TInternetExplorer.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TInternetExplorerCommandStateChange;
```

C++

```
__property TInternetExplorerCommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TInternetExplorer.

7.1.8.6.4 TInternetExplorer.OnDocumentComplete Event

Pascal

```
property OnDocumentComplete: TInternetExplorerDocumentComplete;
```

C++

```
__property TInternetExplorerDocumentComplete OnDocumentComplete;
```

Description

This is OnDocumentComplete, a member of class TInternetExplorer.

7.1.8.6.5 TInternetExplorer.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TInternetExplorer.

7.1.8.6.6 TInternetExplorer.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TInternetExplorer.

7.1.8.6.7 TInternetExplorer.OnFileDownload Event

Pascal

```
property OnFileDownload: TInternetExplorerFileDownload;
```

C++

```
__property TInternetExplorerFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TInternetExplorer.

7.1.8.6.8 TInternetExplorer.OnFullScreen Event

Pascal

```
property OnFullScreen: TInternetExplorerOnFullScreen;
```

C++

```
__property TInternetExplorerOnFullScreen OnFullScreen;
```

Description

This is OnFullScreen, a member of class TInternetExplorer.

7.1.8.6.9 TInternetExplorer.OnMenuBar Event

Pascal

```
property OnMenuBar: TInternetExplorerOnMenuBar;
```

C++

```
__property TInternetExplorerOnMenuBar OnMenuBar;
```

Description

This is OnMenuBar, a member of class TInternetExplorer.

7.1.8.6.10 TInternetExplorer.OnNavigateComplete2 Event

Pascal

```
property OnNavigateComplete2: TInternetExplorerNavigateComplete2;
```

C++

```
__property TInternetExplorerNavigateComplete2 OnNavigateComplete2;
```

Description

This is OnNavigateComplete2, a member of class TInternetExplorer.

7.1.8.6.11 TInternetExplorer.OnNavigateError Event

Pascal

```
property OnNavigateError: TInternetExplorerNavigateError;
```

C++

```
__property TInternetExplorerNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TInternetExplorer.

7.1.8.6.12 TInternetExplorer.OnNewWindow2 Event

Pascal

```
property OnNewWindow2: TInternetExplorerNewWindow2;
```

C++

```
__property TInternetExplorerNewWindow2 OnNewWindow2;
```

Description

This is OnNewWindow2, a member of class TInternetExplorer.

7.1.8.6.13 TInternetExplorer.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TInternetExplorerPrintTemplateInstantiation;
```

C++

```
__property TInternetExplorerPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.

7.1.8.6.14 TInternetExplorer.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TInternetExplorerPrintTemplateTeardown;
```

C++

```
__property TInternetExplorerPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TInternetExplorer.

7.1.8.6.15 TInternetExplorer.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TInternetExplorerPrivacyImpactedStateChange;
```

C++

```
__property TInternetExplorerPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.

7.1.8.6.16 TInternetExplorer.OnProgressChange Event

Pascal

```
property OnProgressChange: TInternetExplorerProgressChange;
```

C++

```
__property TInternetExplorerProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TInternetExplorer.

7.1.8.6.17 TInternetExplorer.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TInternetExplorerPropertyChange;
```

C++

```
__property TInternetExplorerPropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TInternetExplorer.

7.1.8.6.18 TInternetExplorer.OnQuit Event

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

This is OnQuit, a member of class TInternetExplorer.

7.1.8.6.19 TInternetExplorer.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TInternetExplorerSetSecureLockIcon;
```

C++

```
__property TInternetExplorerSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TInternetExplorer.

7.1.8.6.20 TInternetExplorer.OnStatusBar Event

Pascal

```
property OnStatusBar: TInternetExplorerOnStatusBar;
```

C++

```
__property TInternetExplorerOnStatusBar OnStatusBar;
```

Description

This is OnStatusBar, a member of class TInternetExplorer.

7.1.8.6.21 TInternetExplorer.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TInternetExplorerStatusTextChange;
```

C++

```
__property TInternetExplorerStatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TInternetExplorer.

7.1.8.6.22 TInternetExplorer.OnTheaterMode Event

Pascal

```
property OnTheaterMode: TInternetExplorerOnTheaterMode;
```

C++

```
__property TInternetExplorerOnTheaterMode OnTheaterMode;
```

Description

This is OnTheaterMode, a member of class TInternetExplorer.

7.1.8.6.23 TInternetExplorer.OnTitleChange Event

Pascal

```
property OnTitleChange: TInternetExplorerTitleChange;
```

C++

```
__property TInternetExplorerTitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TInternetExplorer.

7.1.8.6.24 TInternetExplorer.OnToolBar Event

Pascal

```
property OnToolBar: TInternetExplorerOnToolBar;
```

C++

```
__property TInternetExplorerOnToolBar OnToolBar;
```

Description

This is OnToolBar, a member of class TInternetExplorer.

7.1.8.6.25 TInternetExplorer.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TInternetExplorerUpdatePageStatus;
```

C++

```
__property TInternetExplorerUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TInternetExplorer.

7.1.8.6.26 TInternetExplorer.OnVisible Event

Pascal

```
property OnVisible: TInternetExplorerOnVisible;
```

C++

```
__property TInternetExplorerOnVisible OnVisible;
```

Description

This is OnVisible, a member of class TInternetExplorer.

7.1.8.6.27 TInternetExplorer.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TInternetExplorerWindowClosing;
```

C++

```
__property TInternetExplorerWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TInternetExplorer.

7.1.8.6.28 TInternetExplorer.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TInternetExplorerWindowSetHeight;
```

C++

```
__property TInternetExplorerWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TInternetExplorer.

7.1.8.6.29 TInternetExplorer.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TInternetExplorerWindowSetLeft;
```

C++

```
__property TInternetExplorerWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TInternetExplorer.

7.1.8.6.30 TInternetExplorer.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TInternetExplorerWindowSetResizable;
```

C++

```
__property TInternetExplorerWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TInternetExplorer.

7.1.8.6.31 TInternetExplorer.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TInternetExplorerWindowSetTop;
```

C++

```
__property TInternetExplorerWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TInternetExplorer.

7.1.8.6.32 TInternetExplorer.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TInternetExplorerWindowSetWidth;
```

C++

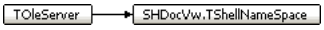
```
__property TInternetExplorerWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TInternetExplorer.

7.1.9 TShellNameSpace Class

Class Hierarchy



Pascal

```
TShellNameSpace = class(TOleServer);
```

C++

```
class TShellNameSpace : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellNameSpace.

7.1.9.1 TShellNameSpace Members

The following tables list the members exposed by TShellNameSpace.

Protected Methods

	Name	Description
	Get_Columns (see page 430)	This is Get_Columns, a member of class TShellNameSpace.
	Get_CountViewTypes (see page 430)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 430)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 431)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 431)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 431)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 431)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 432)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 432)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 432)	This is Get_TVFlags, a member of class TShellNameSpace.
	InitServerData (see page 432)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 433)	Dispatches an event from the COM server to the appropriate event handler.
	Set_Columns (see page 435)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 435)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 435)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 435)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 436)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 436)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 436)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 436)	This is Set_TVFlags, a member of class TShellNameSpace.

Public Methods

	Name	Description
V	Connect (see page 428)	Establishes a connection to the COM server.
	ConnectTo (see page 428)	This is ConnectTo, a member of class TShellNameSpace.
V	Create (see page 425)	Allocates memory and constructs a safely initialized instance of a component.
	CreateSubscriptionForSelection (see page 429)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.
	DeleteSubscriptionForSelection (see page 429)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
V	Destroy (see page 426)	Disposes of the component and its owned components.
V	Disconnect (see page 429)	Terminates the connection to the COM server.
	Expand (see page 430)	This is Expand, a member of class TShellNameSpace.
	Export (see page 430)	This is Export, a member of class TShellNameSpace.
	Import (see page 432)	This is Import, a member of class TShellNameSpace.
	InvokeContextMenuCommand (see page 433)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	MoveSelectionDown (see page 433)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 434)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 434)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 434)	This is NewFolder, a member of class TShellNameSpace.
	ResetSort (see page 434)	This is ResetSort, a member of class TShellNameSpace.
	SelectedItems (see page 435)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 437)	This is SetRoot, a member of class TShellNameSpace.
	SetViewType (see page 437)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 437)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 437)	This is UnselectAll, a member of class TShellNameSpace.

Public Properties

	Name	Description
	Columns (see page 438)	This is Columns, a member of class TShellNameSpace.
R	CountViewTypes (see page 438)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 438)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 438)	This is Depth, a member of class TShellNameSpace.
	EnumOptions (see page 439)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 439)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 439)	This is Mode, a member of class TShellNameSpace.
	Root (see page 439)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 440)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 440)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 440)	This is TVFlags, a member of class TShellNameSpace.

Published Events

	Name	Description
	OnDoubleClick (see page 440)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 441)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 441)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 441)	This is OnSelectionChange, a member of class TShellNameSpace.

7.1.9.2 TShellNameSpace.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3366)

Destroy

System::TObject::Free (see page 3382)

Owner

7.1.9.3 TShellNameSpace.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellNameSpace();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3366)

DestroyComponents

Destroying

System::TObject::Free (see page 3382)

Notification

Owner

Release

Components

7.1.9.4 TShellNameSpace Methods

The methods of the TShellNameSpace class are listed here.

Protected Methods

	Name	Description
	Get_Columns (see page 430)	This is Get_Columns, a member of class TShellNameSpace.
	Get_CountViewTypes (see page 430)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 430)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 431)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 431)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 431)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 431)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 432)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 432)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 432)	This is Get_TVFlags, a member of class TShellNameSpace.
	InitServerData (see page 432)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 433)	Dispatches an event from the COM server to the appropriate event handler.
	Set_Columns (see page 435)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 435)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 435)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 435)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 436)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 436)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 436)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 436)	This is Set_TVFlags, a member of class TShellNameSpace.

Public Methods

	Name	Description
	Connect (see page 428)	Establishes a connection to the COM server.
	ConnectTo (see page 428)	This is ConnectTo, a member of class TShellNameSpace.
	CreateSubscriptionForSelection (see page 429)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.

	DeleteSubscriptionForSelection (see page 429)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
	Disconnect (see page 429)	Terminates the connection to the COM server.
	Expand (see page 430)	This is Expand, a member of class TShellNameSpace.
	Export (see page 430)	This is Export, a member of class TShellNameSpace.
	Import (see page 432)	This is Import, a member of class TShellNameSpace.
	InvokeContextMenuCommand (see page 433)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	MoveSelectionDown (see page 433)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 434)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 434)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 434)	This is NewFolder, a member of class TShellNameSpace.
	ResetSort (see page 434)	This is ResetSort, a member of class TShellNameSpace.
	SelectedItems (see page 435)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 437)	This is SetRoot, a member of class TShellNameSpace.
	SetViewType (see page 437)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 437)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 437)	This is UnselectAll, a member of class TShellNameSpace.

7.1.9.4.1 TShellNameSpace.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

7.1.9.4.2 TShellNameSpace.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellNameSpace);
```

C++

```
__fastcall ConnectTo(IShellNameSpace svrIntf);
```

Description

This is ConnectTo, a member of class TShellNameSpace.

7.1.9.4.3 TShellNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class TShellNameSpace.

7.1.9.4.4 TShellNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.

7.1.9.4.5 TShellNameSpace.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

7.1.9.4.6 TShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class TShellNameSpace.

7.1.9.4.7 TShellNameSpace.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class TShellNameSpace.

7.1.9.4.8 TShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class TShellNameSpace.

7.1.9.4.9 TShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class TShellNameSpace.

7.1.9.4.10 TShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class TShellNameSpace.

7.1.9.4.11 TShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class TShellNameSpace.

7.1.9.4.12 TShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class TShellNameSpace.

7.1.9.4.13 TShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class TShellNameSpace.

7.1.9.4.14 TShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class TShellNameSpace.

7.1.9.4.15 TShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class TShellNameSpace.

7.1.9.4.16 TShellNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.

7.1.9.4.17 TShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class TShellNameSpace.

7.1.9.4.18 TShellNameSpace.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class TShellNameSpace.

7.1.9.4.19 TShellNameSpace.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

7.1.9.4.20 TShellNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class TShellNameSpace.

7.1.9.4.21 TShellNameSpace.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch

7.1.9.4.22 TShellNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class TShellNameSpace.

7.1.9.4.23 TShellNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class TShellNameSpace.

7.1.9.4.24 TShellNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class TShellNameSpace.

7.1.9.4.25 TShellNameSpace.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class TShellNameSpace.

7.1.9.4.26 TShellNameSpace.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class TShellNameSpace.

7.1.9.4.27 TShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class TShellNameSpace.

7.1.9.4.28 TShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString);
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class TShellNameSpace.

7.1.9.4.29 TShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT);
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class TShellNameSpace.

7.1.9.4.30 TShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer);
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class TShellNameSpace.

7.1.9.4.31 TShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord);
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class TShellNameSpace.

7.1.9.4.32 TShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT);
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class TShellNameSpace.

7.1.9.4.33 TShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant);
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class TShellNameSpace.

7.1.9.4.34 TShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch);
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class TShellNameSpace.

7.1.9.4.35 TShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord);
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class TShellNameSpace.

7.1.9.4.36 TShellNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class TShellNameSpace.

7.1.9.4.37 TShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class TShellNameSpace.

7.1.9.4.38 TShellNameSpace.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class TShellNameSpace.

7.1.9.4.39 TShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class TShellNameSpace.

7.1.9.5 TShellNameSpace Properties

The properties of the TShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 438)	This is Columns, a member of class TShellNameSpace.
R	CountViewTypes (see page 438)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 438)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 438)	This is Depth, a member of class TShellNameSpace.
	EnumOptions (see page 439)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 439)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 439)	This is Mode, a member of class TShellNameSpace.
	Root (see page 439)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 440)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 440)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 440)	This is TVFlags, a member of class TShellNameSpace.

7.1.9.5.1 TShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class TShellNameSpace.

7.1.9.5.2 TShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class TShellNameSpace.

7.1.9.5.3 TShellNameSpace.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellNameSpace;
```

C++

```
__property IShellNameSpace DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellNameSpace.

7.1.9.5.4 TShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class TShellNameSpace.

7.1.9.5.5 TShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class TShellNameSpace.

7.1.9.5.6 TShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class TShellNameSpace.

7.1.9.5.7 TShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class TShellNameSpace.

7.1.9.5.8 TShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class TShellNameSpace.

7.1.9.5.9 TShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class TShellNameSpace.

7.1.9.5.10 TShellNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class TShellNameSpace.

7.1.9.5.11 TShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class TShellNameSpace.

7.1.9.6 TShellNameSpace Events

The events of the TShellNameSpace class are listed here.

Published Events

	Name	Description
	OnDoubleClick (see page 440)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 441)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 441)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 441)	This is OnSelectionChange, a member of class TShellNameSpace.

7.1.9.6.1 TShellNameSpace.OnDoubleClick Event

Pascal

```
property OnDoubleClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDoubleClick;
```

Description

This is OnDoubleClick, a member of class TShellNameSpace.

7.1.9.6.2 TShellNameSpace.OnFavoritesSelectionChange Event

Pascal

```
property OnFavoritesSelectionChange: TShellNameSpaceFavoritesSelectionChange;
```

C++

```
__property TShellNameSpaceFavoritesSelectionChange OnFavoritesSelectionChange;
```

Description

This is OnFavoritesSelectionChange, a member of class TShellNameSpace.

7.1.9.6.3 TShellNameSpace.OnInitialized Event

Pascal

```
property OnInitialized: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnInitialized;
```

Description

This is OnInitialized, a member of class TShellNameSpace.

7.1.9.6.4 TShellNameSpace.OnSelectionChange Event

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

This is OnSelectionChange, a member of class TShellNameSpace.

7.1.10 TShellUIHelper Class

Class Hierarchy

Pascal

```
TShellUIHelper = class(TOleServer);
```

C++

```
class TShellUIHelper : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellUIHelper.

7.1.10.1 TShellUIHelper Members

The following tables list the members exposed by TShellUIHelper.

Protected Methods

	Name	Description
	InitServerData (see page 450)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	AddChannel (see page 444)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 445)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 446)	This is the overview for the AddFavorite method overload.
	AutoCompleteAttach (see page 447)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 448)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 448)	This is the overview for the AutoScan method overload.
	Connect (see page 449)	Establishes a connection to the COM server.
	ConnectTo (see page 449)	This is ConnectTo, a member of class TShellUIHelper.
	Create (see page 442)	Allocates memory and constructs a safely initialized instance of a component.
	Destroy (see page 443)	Disposes of the component and its owned components.
	Disconnect (see page 450)	Terminates the connection to the COM server.
	ImportExportFavorites (see page 450)	This is ImportExportFavorites, a member of class TShellUIHelper.
	IsSubscribed (see page 451)	This is IsSubscribed, a member of class TShellUIHelper.
	NavigateAndFind (see page 451)	This is NavigateAndFind, a member of class TShellUIHelper.
	RefreshOfflineDesktop (see page 451)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	ResetFirstBootMode (see page 451)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 452)	This is ResetSafeMode, a member of class TShellUIHelper.
	ShowBrowserUI (see page 452)	This is ShowBrowserUI, a member of class TShellUIHelper.

Public Properties

	Name	Description
	DefaultInterface (see page 452)	This is DefaultInterface, a member of class TShellUIHelper.

7.1.10.2 TShellUIHelper.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3366)

Destroy

System::TObject::Free (see page 3382)

Owner

7.1.10.3 TShellUIHelper.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellUIHelper();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3366)

DestroyComponents

Destroying

System::TObject::Free (see page 3382)

Notification

Owner

Release

Components

7.1.10.4 TShellUIHelper Methods

The methods of the TShellUIHelper class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 450)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	AddChannel (see page 444)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 445)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 446)	This is the overview for the AddFavorite method overload.
	AutoCompleteAttach (see page 447)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 448)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 448)	This is the overview for the AutoScan method overload.
	Connect (see page 449)	Establishes a connection to the COM server.
	ConnectTo (see page 449)	This is ConnectTo, a member of class TShellUIHelper.
	Disconnect (see page 450)	Terminates the connection to the COM server.
	ImportExportFavorites (see page 450)	This is ImportExportFavorites, a member of class TShellUIHelper.
	IsSubscribed (see page 451)	This is IsSubscribed, a member of class TShellUIHelper.
	NavigateAndFind (see page 451)	This is NavigateAndFind, a member of class TShellUIHelper.
	RefreshOfflineDesktop (see page 451)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	ResetFirstBootMode (see page 451)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 452)	This is ResetSafeMode, a member of class TShellUIHelper.
	ShowBrowserUI (see page 452)	This is ShowBrowserUI, a member of class TShellUIHelper.

7.1.10.4.1 TShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class TShellUIHelper.

7.1.10.4.2 AddDesktopComponent Method

This is the overview for the AddDesktopComponent method overload.

Overload List

	Name	Description
	TShellUIHelper.AddDesktopComponent (WideString, WideString) (see page 445)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant) (see page 445)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant) (see page 445)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant) (see page 446)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 446)	This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.2.1 TShellUIHelper.AddDesktopComponent Method (WideString, WideString)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.2.2 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.2.3 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.2.4 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.2.5 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

7.1.10.4.3 AddFavorite Method

This is the overview for the AddFavorite method overload.

Overload List

	Name	Description
	TShellUIHelper.AddFavorite (WideString) (see page 446)	This is AddFavorite, a member of class TShellUIHelper.
	TShellUIHelper.AddFavorite (WideString, OleVariant) (see page 447)	This is AddFavorite, a member of class TShellUIHelper.

7.1.10.4.3.1 TShellUIHelper.AddFavorite Method (WideString)

Pascal

```
procedure AddFavorite(const URL: WideString); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

7.1.10.4.3.2 TShellUIHelper.AddFavorite Method (WideString, OleVariant)

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

7.1.10.4.4 AutoCompleteAttach Method

This is the overview for the AutoCompleteAttach method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteAttach () (see page 447)	This is AutoCompleteAttach, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteAttach (OleVariant) (see page 447)	This is AutoCompleteAttach, a member of class TShellUIHelper.

7.1.10.4.4.1 TShellUIHelper.AutoCompleteAttach Method ()

Pascal

```
procedure AutoCompleteAttach; overload;
```

C++

```
__fastcall AutoCompleteAttach();
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

7.1.10.4.4.2 TShellUIHelper.AutoCompleteAttach Method (OleVariant)

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

7.1.10.4.5 AutoCompleteSaveForm Method

This is the overview for the AutoCompleteSaveForm method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteSaveForm () (see page 448)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteSaveForm (OleVariant) (see page 448)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.

7.1.10.4.5.1 TShellUIHelper.AutoCompleteSaveForm Method ()

Pascal

```
procedure AutoCompleteSaveForm; overload;
```

C++

```
__fastcall AutoCompleteSaveForm();
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

7.1.10.4.5.2 TShellUIHelper.AutoCompleteSaveForm Method (OleVariant)

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

7.1.10.4.6 AutoScan Method

This is the overview for the AutoScan method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoScan (WideString, WideString) (see page 448)	This is AutoScan, a member of class TShellUIHelper.
	TShellUIHelper.AutoScan (WideString, WideString, OleVariant) (see page 449)	This is AutoScan, a member of class TShellUIHelper.

7.1.10.4.6.1 TShellUIHelper.AutoScan Method (WideString, WideString)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl);
```

Description

This is AutoScan, a member of class TShellUIHelper.

7.1.10.4.6.2 TShellUIHelper.AutoScan Method (WideString, WideString, OleVariant)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class TShellUIHelper.

7.1.10.4.7 TShellUIHelper.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

7.1.10.4.8 TShellUIHelper.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellUIHelper);
```

C++

```
__fastcall ConnectTo(IShellUIHelper svrIntf);
```

Description

This is ConnectTo, a member of class TShellUIHelper.

7.1.10.4.9 TShellUIHelper.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

7.1.10.4.10 TShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class TShellUIHelper.

7.1.10.4.11 TShellUIHelper.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

7.1.10.4.12 TShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class TShellUIHelper.

7.1.10.4.13 TShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class TShellUIHelper.

7.1.10.4.14 TShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class TShellUIHelper.

7.1.10.4.15 TShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class TShellUIHelper.

7.1.10.4.16 TShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class TShellUIHelper.

7.1.10.4.17 TShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class TShellUIHelper.

7.1.10.5 TShellUIHelper Properties

The properties of the TShellUIHelper class are listed here.

Public Properties

	Name	Description
	DefaultInterface (see page 452)	This is DefaultInterface, a member of class TShellUIHelper.

7.1.10.5.1 TShellUIHelper.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellUIHelper;
```

C++

```
__property IShellUIHelper DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellUIHelper.

7.1.11 TShellWindows Class

Class Hierarchy

Pascal

```
TShellWindows = class(TOleServer);
```

C++

```
class TShellWindows : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellWindows.

7.1.11.1 TShellWindows Members

The following tables list the members exposed by TShellWindows.

Protected Methods

	Name	Description
	Get_Count (see page 457)	This is Get_Count, a member of class TShellWindows.
V	InitServerData (see page 457)	Initializes the ServerData property to reflect the type library information of the COM server.
V	InvokeEvent (see page 458)	Dispatches an event from the COM server to the appropriate event handler.

Public Methods

	Name	Description
	_NewEnum (see page 456)	This is _NewEnum, a member of class TShellWindows.
V	Connect (see page 456)	Establishes a connection to the COM server.
	ConnectTo (see page 456)	This is ConnectTo, a member of class TShellWindows.
V	Create (see page 453)	Allocates memory and constructs a safely initialized instance of a component.
V	Destroy (see page 454)	Disposes of the component and its owned components.
V	Disconnect (see page 457)	Terminates the connection to the COM server.
	FindWindowSW (see page 457)	This is FindWindowSW, a member of class TShellWindows.
	Item (see page 458)	This is the overview for the Item method overload.
	OnActivated (see page 459)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 459)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 459)	This is OnNavigate, a member of class TShellWindows.
	ProcessAttachDetach (see page 460)	This is ProcessAttachDetach, a member of class TShellWindows.
	Register (see page 460)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 460)	This is RegisterPending, a member of class TShellWindows.
	Revoke (see page 460)	This is Revoke, a member of class TShellWindows.

Public Properties

	Name	Description
R	Count (see page 461)	This is Count, a member of class TShellWindows.
R	DefaultInterface (see page 461)	This is DefaultInterface, a member of class TShellWindows.

Published Events

	Name	Description
	OnWindowRegistered (see page 461)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 462)	This is OnWindowRevoked, a member of class TShellWindows.

7.1.11.2 TShellWindows.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

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

C++

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

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3366)

Destroy

System::TObject::Free (see page 3382)

Owner

7.1.11.3 TShellWindows.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellWindows();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3366)

DestroyComponents

Destroying

System::TObject::Free (see page 3382)

Notification

Owner

Release

Components

7.1.11.4 TShellWindows Methods

The methods of the TShellWindows class are listed here.

Protected Methods

	Name	Description
	Get_Count (see page 457)	This is Get_Count, a member of class TShellWindows.
	InitServerData (see page 457)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 458)	Dispatches an event from the COM server to the appropriate event handler.

Public Methods

	Name	Description
	_NewEnum (see page 456)	This is _NewEnum, a member of class TShellWindows.
	Connect (see page 456)	Establishes a connection to the COM server.
	ConnectTo (see page 456)	This is ConnectTo, a member of class TShellWindows.
	Disconnect (see page 457)	Terminates the connection to the COM server.
	FindWindowSW (see page 457)	This is FindWindowSW, a member of class TShellWindows.
	Item (see page 458)	This is the overview for the Item method overload.
	OnActivated (see page 459)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 459)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 459)	This is OnNavigate, a member of class TShellWindows.
	ProcessAttachDetach (see page 460)	This is ProcessAttachDetach, a member of class TShellWindows.
	Register (see page 460)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 460)	This is RegisterPending, a member of class TShellWindows.
	Revoke (see page 460)	This is Revoke, a member of class TShellWindows.

7.1.11.4.1 TShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class TShellWindows.

7.1.11.4.2 TShellWindows.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

7.1.11.4.3 TShellWindows.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellWindows);
```

C++

```
__fastcall ConnectTo(IShellWindows svrIntf);
```

Description

This is ConnectTo, a member of class TShellWindows.

7.1.11.4.4 TShellWindows.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

7.1.11.4.5 TShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class TShellWindows.

7.1.11.4.6 TShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class TShellWindows.

7.1.11.4.7 TShellWindows.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

7.1.11.4.8 TShellWindows.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch

7.1.11.4.9 Item Method

This is the overview for the Item method overload.

Overload List

	Name	Description
	TShellWindows.Item () (see page 458)	This is Item, a member of class TShellWindows.
	TShellWindows.Item (OleVariant) (see page 459)	This is Item, a member of class TShellWindows.

7.1.11.4.9.1 TShellWindows.Item Method ()

Pascal

```
function Item: IDispatch; overload;
```

C++

```
__fastcall IDispatch Item();
```

Description

This is Item, a member of class TShellWindows.

7.1.11.4.9.2 TShellWindows.Item Method (OleVariant)

Pascal

```
function Item(index: OleVariant): IDispatch; overload;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class TShellWindows.

7.1.11.4.10 TShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class TShellWindows.

7.1.11.4.11 TShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class TShellWindows.

7.1.11.4.12 TShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class TShellWindows.

7.1.11.4.13 TShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class TShellWindows.

7.1.11.4.14 TShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; HWND: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int HWND, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class TShellWindows.

7.1.11.4.15 TShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class TShellWindows.

7.1.11.4.16 TShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class TShellWindows.

7.1.11.5 TShellWindows Properties

The properties of the TShellWindows class are listed here.

Public Properties

	Name	Description
	Count (see page 461)	This is Count, a member of class TShellWindows.
	DefaultInterface (see page 461)	This is DefaultInterface, a member of class TShellWindows.

7.1.11.5.1 TShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TShellWindows.

7.1.11.5.2 TShellWindows.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellWindows;
```

C++

```
__property IShellWindows DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellWindows.

7.1.11.6 TShellWindows Events

The events of the TShellWindows class are listed here.

Published Events

	Name	Description
	OnWindowRegistered (see page 461)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 462)	This is OnWindowRevoked, a member of class TShellWindows.

7.1.11.6.1 TShellWindows.OnWindowRegistered Event

Pascal

```
property OnWindowRegistered: TShellWindowsWindowRegistered;
```

C++

```
__property TShellWindowsWindowRegistered OnWindowRegistered;
```

Description

This is OnWindowRegistered, a member of class TShellWindows.

7.1.11.6.2 TShellWindows.OnWindowRevoked Event

Pascal

```
property OnWindowRevoked: TShellWindowsWindowRevoked;
```

C++

```
__property TShellWindowsWindowRevoked OnWindowRevoked;
```

Description

This is OnWindowRevoked, a member of class TShellWindows.

7.1.12 TWebBrowser Class

TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).

Class Hierarchy

Pascal

```
TWebBrowser = class(TOleControl);
```

C++

```
class TWebBrowser : public TOleControl;
```

File

SHDocVw

Description

TWebBrowser wraps the IWebBrowser2 (see page 698) interface from Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL) to allow you to create a customized Web browsing application or to add Internet, file and network browsing, document viewing, and data downloading capabilities to your applications.

Note: Because TWebBrowser wraps the SHDOCVW.DLL, you must have SHDOCVW.DLL installed to use this component. This DLL ships with Internet Explorer 4 and later.

7.1.12.1 TWebBrowser Members

The following tables list the members exposed by TWebBrowser.

Protected Methods

	Name	Description
	CreateControl (see page 466)	This is CreateControl, a member of class TWebBrowser.
	Get_Application (see page 467)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 467)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 467)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 467)	This is Get_Parent, a member of class TWebBrowser.
	InitControlData (see page 470)	This is InitControlData, a member of class TWebBrowser.

Public Methods

	Name	Description
	ClientToWindow (see page 465)	Not used in TWebBrowser (see page 462).
	ExecWB (see page 466)	This is the overview for the ExecWB method overload.
	GetProperty (see page 468)	Not used in TWebBrowser (see page 462).
	GoBack (see page 468)	Navigates to the previous item in the history list.
	GoForward (see page 468)	Navigates to the next item in the history list.
	GoHome (see page 469)	Navigates to the current start page.
	GoSearch (see page 469)	Navigates to the current search page.
	Navigate (see page 470)	This is the overview for the Navigate method overload.
	Navigate2 (see page 472)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 473)	Not used in TWebBrowser (see page 462).
	QueryStatusWB (see page 474)	Returns information about what commands the Web browser supports.
	Quit (see page 474)	Not used in TWebBrowser (see page 462).
	Refresh (see page 474)	Reloads the current document.
	Refresh2 (see page 475)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 476)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 476)	Stops the current operation.

Public Properties

	Name	Description
R	Application (see page 482)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 779) control, or, if that is unavailable, the interface for the WebBrowser (see page 779) control.
R	Busy (see page 482)	Indicates whether the WebBrowser (see page 779) control is in the middle of a navigation or downloading operation.
R	Container (see page 483)	Provides access to the automation interface of the Web browser's container.
R	ControlInterface (see page 483)	Provides access to the interface for this Web browser component.
R	DefaultInterface (see page 484)	Provides access to the default interface for the component.
R	Document (see page 484)	Provides access to the Automation interface of the active document.
R	FullName (see page 486)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
R	HWnd (see page 487)	This is HWnd, a member of class TWebBrowser.
R	LocationName (see page 487)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 488)	Identifies the URL or path of the document or folder currently displayed.
R	Name (see page 488)	Indicates the name for the executable that hosts the Web Browser component.
R	Parent (see page 489)	Provides access to the Automation interface of the Web browser's container.
R	Path (see page 490)	Indicates the name for the executable that hosts the Web Browser component.
R	ReadyState (see page 496)	Indicates the state of the Web browser component.
R	TopLevelContainer (see page 501)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 501)	This is type_, a member of class TWebBrowser.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 503)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 503)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 503)	Occurs when the ability to execute certain TWebBrowser (see page 462) methods changes.
	OnDocumentComplete (see page 504)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 504)	Occurs when the Web browser starts downloading a document.
	OnDownloadComplete (see page 505)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 505)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 508)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 512)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 513)	Occurs when a control receives the input focus.

	OnExit (see page 515)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 518)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 518)	Not used in TWebBrowser (see page 462).
	OnMenuBar (see page 518)	Not used in TWebBrowser (see page 462).
	OnNavigateComplete2 (see page 519)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 519)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 519)	Occurs when a new, hidden, non-navigated Web browser window is needed.
	OnPrintTemplateInstantiation (see page 520)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 520)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 520)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 520)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 521)	Not used in TWebBrowser (see page 462).
	OnQuit (see page 521)	Not used in TWebBrowser (see page 462).
	OnSetSecureLockIcon (see page 521)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 522)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 522)	Not used in TWebBrowser (see page 462).
	OnStatusTextChange (see page 523)	Not used in TWebBrowser (see page 462).
	OnTheaterMode (see page 523)	Not used in TWebBrowser (see page 462).
	OnTitleChange (see page 523)	Occurs when the title of a document in the WebBrowser (see page 779) control becomes available or changes.
	OnToolBar (see page 524)	Not used in TWebBrowser (see page 462).
	OnUpdatePageStatus (see page 524)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 524)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 524)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 525)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 525)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 525)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 525)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 525)	This is OnWindowSetWidth, a member of class TWebBrowser.

Published Properties

	Name	Description
	AddressBar (see page 478)	Not used in TWebBrowser (see page 462).
	Align (see page 478)	Determines how the control aligns within its container (parent control).
	Anchors (see page 481)	Specifies how the control is anchored to its parent.
	DragCursor (see page 484)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 485)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 487)	Not used in TWebBrowser (see page 462).
	MenuBar (see page 488)	Not used in TWebBrowser (see page 462).
	Offline (see page 489)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 490)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 491)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 496)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 496)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 497)	Not used in TWebBrowser (see page 462).
	ShowHint (see page 497)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 499)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 499)	Not used in TWebBrowser (see page 462).
	StatusText (see page 499)	Not used in TWebBrowser (see page 462).
	TabOrder (see page 499)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 500)	Determines if the user can tab to a control.
	TheaterMode (see page 500)	Not used in TWebBrowser (see page 462).

	ToolBar (see page 501)	Not used in TWebBrowser (see page 462).
	Visible (see page 502)	Specifies whether the Web browser component appears onscreen.

7.1.12.2 TWebBrowser Methods

The methods of the TWebBrowser class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 466)	This is CreateControl, a member of class TWebBrowser.
	Get_Application (see page 467)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 467)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 467)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 467)	This is Get_Parent, a member of class TWebBrowser.
	InitControlData (see page 470)	This is InitControlData, a member of class TWebBrowser.

Public Methods

	Name	Description
	ClientToWindow (see page 465)	Not used in TWebBrowser (see page 462).
	ExecWB (see page 466)	This is the overview for the ExecWB method overload.
	GetProperty (see page 468)	Not used in TWebBrowser (see page 462).
	GoBack (see page 468)	Navigates to the previous item in the history list.
	GoForward (see page 468)	Navigates to the next item in the history list.
	GoHome (see page 469)	Navigates to the current start page.
	GoSearch (see page 469)	Navigates to the current search page.
	Navigate (see page 470)	This is the overview for the Navigate method overload.
	Navigate2 (see page 472)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 473)	Not used in TWebBrowser (see page 462).
	QueryStatusWB (see page 474)	Returns information about what commands the Web browser supports.
	Quit (see page 474)	Not used in TWebBrowser (see page 462).
	Refresh (see page 474)	Reloads the current document.
	Refresh2 (see page 475)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 476)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 476)	Stops the current operation.

7.1.12.2.1 TWebBrowser.ClientToWindow Method

Not used in TWebBrowser (see page 462).

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Internet Explorer uses ClientToWindow to convert between client coordinates and window coordinates.

7.1.12.2.2 TWebBrowser.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser.

7.1.12.2.3 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 466)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 466)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 466)	Executes a command.

7.1.12.2.3.1 TWebBrowser.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Executes a command.

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

ExecWB lets you use the IOleCommandTarget interface to pass a command to the Web browser control. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command to execute. The Web Browser does not support all of the commands for which Microsoft providesconstants. Use the QueryStatusWB (see page 474) method to determine what commands can be used.

cmdexecopt modifies how the command is executed.

pvaIn specifies a Variant (see page 3430) array that contains any input arguments. This value can be omitted if there are not input arguments.

pvaOutspecifies a Variant (see page 3430) that receives the results of the command, if any. This value can be omitted if there are no results.

See Also

QueryStatusWB (see page 474)

DoObjectVerb

GetObjectVerbs

7.1.12.2.4 TWebBrowser.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser.

7.1.12.2.5 TWebBrowser.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser.

7.1.12.2.6 TWebBrowser.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser.

7.1.12.2.7 TWebBrowser.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser.

7.1.12.2.8 TWebBrowser.GetProperty Method

Not used in TWebBrowser (see page 462).

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Internet Explorer uses GetProperty to return a property value.

7.1.12.2.9 TWebBrowser.GoBack Method

Navigates to the previous item in the history list.

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

Call GoBack to return to the last document in the history list. GoBack has no effect unless the history list contains additional documents.

Note: The Navigate (see page 470) or Navigate2 (see page 472) command can change the history list, depending on the specified flags.

See Also

OnCommandStateChange (see page 503)

GoForward (see page 468)

GoHome (see page 469)

GoSearch (see page 469)

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.2.10 TWebBrowser.GoForward Method

Navigates to the next item in the history list.

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

Call GoForward to return to the next document in the history list. GoForward reverses the effect of the last GoBack (see page

468) command.

See Also

OnCommandStateChange (see page 503)

GoBack (see page 468)

GoHome (see page 469)

GoSearch (see page 469)

7.1.12.2.11 TWebBrowser.GoHome Method

Navigates to the current start page.

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

Call GoHome to navigate to the registered home or start page. The URL for this page is specified in the Registry at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name Startpage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 468)

GoForward (see page 468)

GoSearch (see page 469)

TRegistry

TRegistryIniFile

7.1.12.2.12 TWebBrowser.GoSearch Method

Navigates to the current search page.

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

Call GoSearch to navigate to the registered search page. The URL for this page is specified in the Registry at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name SearchPage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 468)

GoForward (see page 468)

GoHome (see page 469)

TRegistry

TRegistryIniFile

7.1.12.2.13 TWebBrowser.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser.

7.1.12.2.14 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate (WideString) (see page 470)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant) (see page 470)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant) (see page 470)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 470)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 470)	Navigates to a resource identified by a path name or URL.

7.1.12.2.14.1 TWebBrowser.Navigate Method (WideString)

Navigates to a resource identified by a path name or URL.

Pascal

```
procedure Navigate(const URL: WideString); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

Use Navigate to locate and download a specific resource. Navigate can send an HTTP message to a specified URL and display the results or simply display the results of a specified file.

URL specifies the UNC path name of a file or the Uniform Resource Locator of an Internet resource that the Web browser should display.

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 796)	1	Open the resource or file in a new window.
navNoHistory (see page 795)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 796)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 796)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 795)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate to generate an HTTP POST message. If PostData is NULL, Navigate generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

Note: To navigate to a location that can't be expressed as an URL (for example, to an item identifier list), use the Navigate2 (see page 472) method instead.

See Also

Navigate2 (see page 472)

GoBack (see page 468)

GoForward (see page 468)

GoHome (see page 469)

GoSearch (see page 469)

OnBeforeNavigate2 (see page 503)

OnNavigateComplete2 (see page 519)

TWebRequest

7.1.12.2.15 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate2 (OleVariant) (see page 472)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant) (see page 472)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 472)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 472)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 472)	Navigates to a specified resource.

7.1.12.2.15.1 TWebBrowser.Navigate2 Method (OleVariant)

Navigates to a specified resource.

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

Use Navigate2 to locate and download a specific resource. Navigate (see page 470) can send an HTTP message to a specified URL and display the results, display the results of a specified file, or navigate to a resource that can't be expressed as an URL such as an item identifier list.

URL specifies the UNC path name of a file, the Uniform Resource Locator of an Internet resource, or a pointer to an item identifier list (PIDL).

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 796)	1	Open the resource or file in a new window.
navNoHistory (see page 795)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 796)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 796)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 795)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate (see page 470) to generate an HTTP POST message. If PostData is NULL, Navigate (see page 470) generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

See Also

Navigate (see page 470)

GoBack (see page 468)

GoForward (see page 468)

GoHome (see page 469)

GoSearch (see page 469)

OnBeforeNavigate2 (see page 503)

OnNavigateComplete2 (see page 519)

TWebRequest

7.1.12.2.16 TWebBrowser.PutProperty Method

Not used in TWebBrowser (see page 462).

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Internet Explorer uses PutProperty to set a property value.

7.1.12.2.17 TWebBrowser.QueryStatusWB Method

Returns information about what commands the Web browser supports.

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

Use QueryStatusWB to determine what commands can be executed using the ExecWB (see page 466) method. QueryStatusWB uses the QueryStatus method of the IOleCommandTarget interface to obtain information about the status of commands. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command about which you want information.

QueryStatusWB returns a value made by adding together flags that indicate the status of the command.

See Also

ExecWB (see page 466)

7.1.12.2.18 TWebBrowser.Quit Method

Not used in TWebBrowser (see page 462).

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

Do not use this method. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Internet Explorer responds to Quit by shutting down.

7.1.12.2.19 TWebBrowser.Refresh Method

Reloads the current document.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to reload the current document. This is useful when the Web browser is displaying the resource at an URL that may have changed.

Refresh always sends a pragma:nocache header, telling the server not to return a cached copy. This can cause problems with some servers. To prevent the pragma:nocache header, or to refresh only expired pages, use the Refresh2 (see page 475) method instead.

See Also

Refresh2 (see page 475)

Document (see page 484)

7.1.12.2.20 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser.Refresh2 () (see page 475)	Reloads the current document if necessary.
	TWebBrowser.Refresh2 (OleVariant) (see page 475)	Reloads the current document if necessary.

7.1.12.2.20.1 TWebBrowser.Refresh2 Method ()

Reloads the current document if necessary.

Pascal

```
procedure Refresh2; overload;
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2();
__fastcall Refresh2(OleVariant Level);
```

Description

Call Refresh2 to reload the current document. Unlike the Refresh (see page 474) method, Refresh2 lets you specify what level of refresh to perform.

Level indicates what type of information is refreshed. The following table lists the possible values:

Constant	Value	Meaning
REFRESH_NORMAL (see page 820)	0	Perform a lightweight refresh that does not include the pragma:nocache header. The pragma:nocache header tells the server not to return a cached copy. This can cause problems with some servers.
REFRESH_IFEXPIRED (see page 820)	1	Only refresh if the page has expired. Do not include the pragma:nocache header.
REFRESH_CONTINUE (see page 819)	2	For internal use only.
REFRESH_COMPLETELY (see page 819)	3	Perform a full refresh, including the pragma:nocache header. Using this option is the same as calling the Refresh (see page 474) method.

When Level is omitted, a value of REFRESH_COMPLETELY (see page 819) is assumed.

See Also

Refresh (see page 474)

7.1.12.2.21 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TWebBrowser.ShowBrowserBar (OleVariant) (see page 476)	Not used in TWebBrowser (see page 462)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant) (see page 476)	Not used in TWebBrowser (see page 462)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 476)	Not used in TWebBrowser (see page 462)

7.1.12.2.21.1 TWebBrowser.ShowBrowserBar Method (OleVariant)

Not used in TWebBrowser (see page 462)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Internet Explorer uses ShowBrowserBar to show or hide a specified browser bar.

7.1.12.2.22 TWebBrowser.Stop Method

Stops the current operation.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Stop cancels any pending navigation or download operation and terminates any dynamic page elements, such as background sounds and animations.

See Also

OnCommandStateChange (see page 503)

GoForward (see page 468)

GoHome (see page 469)

GoSearch (see page 469)

GoBack (see page 468)

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.3 TWebBrowser Properties

The properties of the TWebBrowser class are listed here.

Public Properties

	Name	Description
	Application (see page 482)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 779) control, or, if that is unavailable, the interface for the WebBrowser (see page 779) control.
	Busy (see page 482)	Indicates whether the WebBrowser (see page 779) control is in the middle of a navigation or downloading operation.
	Container (see page 483)	Provides access to the automation interface of the Web browser's container.
	ControlInterface (see page 483)	Provides access to the interface for this Web browser component.
	DefaultInterface (see page 484)	Provides access to the default interface for the component.
	Document (see page 484)	Provides access to the Automation interface of the active document.
	FullName (see page 486)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
	HWnd (see page 487)	This is HWnd, a member of class TWebBrowser.
	LocationName (see page 487)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
	LocationURL (see page 488)	Identifies the URL or path of the document or folder currently displayed.
	Name (see page 488)	Indicates the name for the executable that hosts the Web Browser component.
	Parent (see page 489)	Provides access to the Automation interface of the Web browser's container.
	Path (see page 490)	Indicates the name for the executable that hosts the Web Browser component.
	ReadyState (see page 496)	Indicates the state of the Web browser component.
	TopLevelContainer (see page 501)	Indicates whether the Web browser is a top-level Automation object.
	type_ (see page 501)	This is type_, a member of class TWebBrowser.

Published Properties

	Name	Description
	AddressBar (see page 478)	Not used in TWebBrowser (see page 462).
	Align (see page 478)	Determines how the control aligns within its container (parent control).
	Anchors (see page 481)	Specifies how the control is anchored to its parent.
	DragCursor (see page 484)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 485)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 487)	Not used in TWebBrowser (see page 462).
	MenuBar (see page 488)	Not used in TWebBrowser (see page 462).
	Offline (see page 489)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 490)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 491)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 496)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 496)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 497)	Not used in TWebBrowser (see page 462).
	ShowHint (see page 497)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 499)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 499)	Not used in TWebBrowser (see page 462).
	StatusText (see page 499)	Not used in TWebBrowser (see page 462).
	TabOrder (see page 499)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 500)	Determines if the user can tab to a control.

	TheaterMode (see page 500)	Not used in TWebBrowser (see page 462).
	ToolBar (see page 501)	Not used in TWebBrowser (see page 462).
	Visible (see page 502)	Specifies whether the Web browser component appears onscreen.

7.1.12.3.1 TWebBrowser.AddressBar Property

Not used in TWebBrowser (see page 462).

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

Do not use AddressBar. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. AddressBar shows or hides the URL address bar on Internet Explorer. TWebBrowser (see page 462) does not have an address bar.

7.1.12.3.2 TWebBrowser.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
```

```
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

7.1.12.3.3 TWebBrowser.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.12.3.4 TWebBrowser.Application Property

Provides access to the Automation interface for the application that contains the WebBrowser (see page 779) control, or, if that is unavailable, the interface for the WebBrowser (see page 779) control.

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

Use Application to access the interface for the application containing the WebBrowser (see page 779) control. If the application does not have an Automation interface, or if the interface is not accessible, Application provides access to the Automation interface of the Web browser control.

See Also

Container (see page 483)

Parent (see page 489)

ControlInterface (see page 483)

7.1.12.3.5 TWebBrowser.Busy Property

Indicates whether the WebBrowser (see page 779) control is in the middle of a navigation or downloading operation.

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

Read (see page 3727) Busy to determine whether the Web browser component is currently navigating to a new URL or

downloading a document. If Busy is true, you can interrupt the current operation using the Stop (see page 476) method before starting a new operation.

Note: The Web browser receives events at the beginning and end of each operation, as well as at discrete points during a download operation.

See Also

ReadyState (see page 496)

OnDocumentComplete (see page 504)

OnBeforeNavigate2 (see page 503)

OnNavigateComplete2 (see page 519)

OnDownloadComplete (see page 505)

OnDownloadBegin (see page 504)

OnProgressChange (see page 520)

Stop (see page 476)

7.1.12.3.6 TWebBrowser.Container Property

Provides access to the automation interface of the Web browser's container.

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

Use Container to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, Container is nil.

See Also

Application (see page 482)

Parent (see page 489)

ControlInterface (see page 483)

TopLevelContainer (see page 501)

7.1.12.3.7 TWebBrowser.ControlInterface Property

Provides access to the interface for this Web browser component.

Pascal

```
property ControlInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 ControlInterface;
```

Description

Use ControlInterface to access the interface for the WebBrowser (see page 779) component. This interface lets an Automation

controller access the properties, methods, and events of the Web browser.

See Also

Application (see page 482)

Document (see page 484)

Parent (see page 489)

DefaultInterface (see page 484)

7.1.12.3.8 TWebBrowser.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for the component. As implemented in TWebBrowser (see page 462), this is the same as the ControlInterface (see page 483) property.

See Also

ControlInterface (see page 483)

7.1.12.3.9 TWebBrowser.Document Property

Provides access to the Automation interface of the active document.

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

Use Document to access an IDispatch (see page 3498) interface for manipulating the contents of the HTML document through the COM object model. If there is no active document, Document is nil. That is, you must first load a document using the Navigate (see page 470) or Navigate2 (see page 472) method before you can manipulate it using the Document interface.

For more information about this interface, see the document object section of Microsoft's Dynamic HTML reference.

See Also

ControlInterface (see page 483)

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.3.10 TWebBrowser.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

7.1.12.3.11 TWebBrowser.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}
```

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

7.1.12.3.12 TWebBrowser.FullName Property

Indicates the fully qualified path name for the executable that hosts the Web Browser component.

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

Read (see page 3727) FullName to obtain the name of the executable as a WideString (see page 3476). Unlike the Name (see page 488) property, FullName specifies a fully qualified path name.

See Also

Application (see page 482)

ExeName

Name (see page 488)

Path (see page 490)

7.1.12.3.13 TWebBrowser.FullScreen Property

Not used in TWebBrowser (see page 462).

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

Do not use FullScreen. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. FullScreen determines whether Internet Explorer is maximized.

7.1.12.3.14 TWebBrowser.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class TWebBrowser.

7.1.12.3.15 TWebBrowser.LocationName Property

Identifies the short (UI-friendly) name of the URL or file currently displayed.

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

Read (see page 3727) LocationName to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName is the title of that page. If the resource is a folder or file on the local network or on a disk, LocationName is the full UNC name of the folder or file.

See Also

LocationURL (see page 488)

OnTitleChange (see page 523)

7.1.12.3.16 TWebBrowser.LocationURL Property

Identifies the URL or path of the document or folder currently displayed.

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

Read (see page 3727) LocationURL to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName (see page 487) is its URL. If the resource is a folder or file on the local network or on a disk, LocationName (see page 487) is the full UNC name of the folder or file.

Note: LocationURL is a read-only property. To direct the Web browser to display the resource at a specific URL, folder, or file, use the Navigate (see page 470) or Navigate2 (see page 472) method.

See Also

LocationName (see page 487)

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.3.17 TWebBrowser.MenuBar Property

Not used in TWebBrowser (see page 462).

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

Do not use MenuBar. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. MenuBar shows or hides the menu bar in Internet Explorer. TWebBrowser (see page 462) does not have a menu bar.

7.1.12.3.18 TWebBrowser.Name Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Read (see page 3727) Name to obtain the name of the executable that hosts the Web browser. Unlike the FullName (see page 486) property, Name does not include the path.

See Also

Application (see page 482)

FullName (see page 486)

Path (see page 490)

7.1.12.3.19 TWebBrowser.Offline Property

Indicates whether the Web browser gets its document from the local cache.

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

Set Offline to true to work offline with the Web browser. When the Web browser is offline, it can't read HTML pages using an Internet connection. Instead, you must specify a source document as a UNC filename.

Note: You can use Internet Explorer to save HTML pages to a local cache so that they are available for offline work.

See Also

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.3.20 TWebBrowser.Parent Property

Provides access to the Automation interface of the Web browser's container.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use Parent to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, the value of Parent is the same as ControlInterface (see page 483).

Use the TopLevelContainer (see page 501) property to determine which interface is the value of Parent.

See Also

Container (see page 483)

ControlInterface (see page 483)

Application (see page 482)

TopLevelContainer (see page 501)

7.1.12.3.21 TWebBrowser.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

7.1.12.3.22 TWebBrowser.Path Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

Read (see page 3727) Path to obtain the path of the executable that hosts the Web browser.

See Also

FullName (see page 486)

Name (see page 488)

7.1.12.3.23 TWebBrowser.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

7.1.12.3.24 TWebBrowser.ReadyState Property

Indicates the state of the Web browser component.

Pascal

```
property ReadyState: TOleEnum;
```

C++

```
__property TOleEnum ReadyState;
```

Description

Read (see page 3727) ReadyState to obtain state information specific to the Web browser control. Use this information to avoid attempting to use the Web browser control when it is not ready to perform an action.

The following table lists the possible values:

Constant	Value	Meaning
READYSTATE_UNINITIALIZED (see page 819)	0	Default initialization state. The Control has not begun loading any properties.
READYSTATE_LOADING (see page 818)	1	The control is loading its properties.
READYSTATE_LOADED (see page 818)	2	The control has initialized its properties but is not yet interactive.
READYSTATE_INTERACTIVE (see page 818)	3	The control is interactive, but not all of its data is available. (It is downloading a document or frame).
READYSTATE_COMPLETE (see page 817)	4	The control is interactive and all of its data is available. (The current document is fully downloaded).

See Also

ComponentState

ControlState

Busy (see page 482)

7.1.12.3.25 TWebBrowser.RegisterAsBrowser Property

Registers the control as a top-level browser for target name resolution.

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

Set RegisterAsBrowser to true when using the Web browser control to implement a top-level browser.

7.1.12.3.26 TWebBrowser.RegisterAsDropTarget Property

Specifies whether users can load a document using drag-and-drop.

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

Set RegisterAsDropTarget to true to allow users to drop HTML documents onto the Web browser. When RegisterAsDropTarget is true, users can load a document by dragging it onto the Web browser. When RegisterAsDropTarget is false, dragging a document onto the Web browser has no effect.

See Also

Navigate (see page 470)

Navigate2 (see page 472)

7.1.12.3.27 TWebBrowser.Resizable Property

Not used in TWebBrowser (see page 462).

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

Do not use Resizable. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. Resizable determines whether the Internet Explorer window can be resized.

7.1.12.3.28 TWebBrowser.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

7.1.12.3.29 TWebBrowser.Silent Property

Specifies whether the Web browser can display dialog boxes.

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

Set Silent to true to prevent the Web browser control from displaying dialog boxes.

7.1.12.3.30 TWebBrowser.StatusBar Property

Not used in TWebBrowser (see page 462).

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

Do not use StatusBar. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. StatusBar shows or hides the status bar in Internet Explorer. TWebBrowser (see page 462) does not have a status bar.

7.1.12.3.31 TWebBrowser.StatusText Property

Not used in TWebBrowser (see page 462).

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

Do not use StatusText. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. StatusText specifies the text that appears on the status bar in Internet Explorer. TWebBrowser (see page 462) does not have a status bar.

7.1.12.3.32 TWebBrowser.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

7.1.12.3.33 TWebBrowser.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

7.1.12.3.34 TWebBrowser.TheaterMode Property

Not used in TWebBrowser (see page 462).

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

Do not use TheaterMode. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. TheaterMode changes the display of Internet Explorer to use theater or normal mode.

7.1.12.3.35 TWebBrowser.ToolBar Property

Not used in TWebBrowser (see page 462).

Pascal

```
property ToolBar: Integer;
```

C++

```
__property int ToolBar;
```

Description

Do not use ToolBar. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. ToolBar determines which toolbar is displayed in Internet Explorer. TWebBrowser (see page 462) does not have any tool bars.

7.1.12.3.36 TWebBrowser.TopLevelContainer Property

Indicates whether the Web browser is a top-level Automation object.

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

Read (see page 3727) TopLevelContainer to determine whether the Web browser has a parent with an Automation interface. When TopLevelContainer is false, the Container (see page 483) and Parent (see page 489) properties provide access to the parent's interface. When TopLevelContainer is true, the value of Container (see page 483) is nil and the value of Parent (see page 489) is the Web browser's own interface.

See Also

Container (see page 483)

Parent (see page 489)

7.1.12.3.37 TWebBrowser.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TWebBrowser.

7.1.12.3.38 TWebBrowser.Visible Property

Specifies whether the Web browser component appears onscreen.

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

Use Visible to show or hide the Web browser control.

See Also

Enabled

7.1.12.4 TWebBrowser Events

The events of the TWebBrowser class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 503)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 503)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 503)	Occurs when the ability to execute certain TWebBrowser (see page 462) methods changes.
	OnDocumentComplete (see page 504)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 504)	Occurs when the Web browser starts downloading a document.
	OnDownloadComplete (see page 505)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 505)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 508)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 512)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 513)	Occurs when a control receives the input focus.
	OnExit (see page 515)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 518)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 518)	Not used in TWebBrowser (see page 462).
	OnMenuBar (see page 518)	Not used in TWebBrowser (see page 462).
	OnNavigateComplete2 (see page 519)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 519)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 519)	Occurs when a new, hidden, non-navigated Web browser window is needed.
	OnPrintTemplateInstantiation (see page 520)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 520)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 520)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 520)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 521)	Not used in TWebBrowser (see page 462).
	OnQuit (see page 521)	Not used in TWebBrowser (see page 462).
	OnSetSecureLockIcon (see page 521)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 522)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 522)	Not used in TWebBrowser (see page 462).
	OnStatusTextChange (see page 523)	Not used in TWebBrowser (see page 462).
	OnTheaterMode (see page 523)	Not used in TWebBrowser (see page 462).

	OnTitleChange (see page 523)	Occurs when the title of a document in the WebBrowser (see page 779) control becomes available or changes.
	OnToolBar (see page 524)	Not used in TWebBrowser (see page 462).
	OnUpdatePageStatus (see page 524)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 524)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 524)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 525)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 525)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 525)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 525)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 525)	This is OnWindowSetWidth, a member of class TWebBrowser.

7.1.12.4.1 TWebBrowser.OnBeforeNavigate2 Event

Occurs immediately before the Web browser navigates to a new resource.

Pascal

```
property OnBeforeNavigate2: TWebBrowserBeforeNavigate2;
```

C++

```
__property TWebBrowserBeforeNavigate2 OnBeforeNavigate2;
```

Description

Write (see page 3789) an OnBeforeNavigate2 event handler to redirect or cancel a change to a different URL. This event may occur as the result of a call to the Navigate (see page 470) or Navigate2 (see page 472) method, or the user clicking a link.

See Also

Navigate (see page 470)

Navigate2 (see page 472)

OnNavigateComplete2 (see page 519)

TWebRequest

7.1.12.4.2 TWebBrowser.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TWebBrowserClientToHostWindow;
```

C++

```
__property TWebBrowserClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TWebBrowser.

7.1.12.4.3 TWebBrowser.OnCommandStateChange Event

Occurs when the ability to execute certain TWebBrowser (see page 462) methods changes.

Pascal

```
property OnCommandStateChange: TWebBrowserCommandStateChange;
```

C++

```
__property TWebBrowserCommandStateChange OnCommandStateChange;
```

Description

Write (see page 3789) an OnCommandStateChange to update any controls in the application whose state depends on the ability to execute the Web browser's methods.

See Also

Stop (see page 476)

GoBack (see page 468)

GoForward (see page 468)

Busy (see page 482)

QueryStatusWB (see page 474)

7.1.12.4.4 TWebBrowser.OnDocumentComplete Event

Occurs when the document being navigated to reaches ReadyState_Complete.

Pascal

```
property OnDocumentComplete: TWebBrowserDocumentComplete;
```

C++

```
__property TWebBrowserDocumentComplete OnDocumentComplete;
```

Description

Write (see page 3789) an OnDocumentComplete event handler to take specific action when a frame or document is fully loaded into the Web browser .For a document without frames, this event occurs once when the document finishes loading. On a document containing multiple frames, this event occurs once for each frame. When the multiple-frame document finishes loading, the Web browser fires the event one final time.

See Also

ReadyState (see page 496)

OnDownloadBegin (see page 504)

OnDownloadComplete (see page 505)

Busy (see page 482)

Navigate (see page 470)

Navigate2 (see page 472)

OnBeforeNavigate2 (see page 503)

7.1.12.4.5 TWebBrowser.OnDownloadBegin Event

Occurs when the Web browser starts downloading a document.

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

Write (see page 3789) an OnDownloadBegin event handler to take specific action after the Web browser has located a document and immediately before it starts downloading the document. For example, use the OnDownloadBegin event to launch an animation control the represents downloading or a progress bar that is updated by an OnProgressChange (see page 520) event handler. The control can then be stopped in an OnDownloadComplete (see page 505) event handler.

Note: To take specific action when the Web browser looks up the resource, rather than when it begins downloading, use the OnBeforeNavigate2 (see page 503) event. OnDownloadBegin occurs shortly after OnBeforeNavigate2 (see page 503).

See Also

OnDocumentComplete (see page 504)

OnDownloadComplete (see page 505)

Navigate (see page 470)

Navigate2 (see page 472)

OnBeforeNavigate2 (see page 503)

TAnimate

OnNavigateComplete2 (see page 519)

OnProgressChange (see page 520)

7.1.12.4.6 TWebBrowser.OnDownloadComplete Event

Occurs when a navigation operation finishes, is halted, or fails.

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

Write (see page 3789) an OnDownLoadComplete event handler to take specific action after the Web browser stops a downloading operation. For example, use the OnDownloadComplete event to stop an download indication that is started in an OnDownloadBegin (see page 504) event handler.

Note: Unlike the OnNavigateComplete2 (see page 519) event, OnDownloadComplete occurs even if the Web browser does not successfully navigate to an URL.

See Also

OnDocumentComplete (see page 504)

OnDownloadBegin (see page 504)

Navigate (see page 470)

Navigate2 (see page 472)

OnNavigateComplete2 (see page 519)

7.1.12.4.7 TWebBrowser.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.12.4.8 TWebBrowser.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.12.4.9 TWebBrowser.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

7.1.12.4.10 TWebBrowser.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

7.1.12.4.11 TWebBrowser.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
```

```
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

7.1.12.4.12 TWebBrowser.OnFileDownload Event

Pascal

```
property OnFileDownload: TWebBrowserFileDownload;
```

C++

```
__property TWebBrowserFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TWebBrowser.

7.1.12.4.13 TWebBrowser.OnFullScreen Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnFullScreen: TWebBrowserOnFullScreen;
```

C++

```
__property TWebBrowserOnFullScreen OnFullScreen;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnFullScreen occurs with Internet Explorer when the application is maximized or restored from a maximized state.

7.1.12.4.14 TWebBrowser.OnMenuBar Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnMenuBar: TWebBrowserOnMenuBar;
```

C++

```
__property TWebBrowserOnMenuBar OnMenuBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnMenuBar occurs with Internet Explorer when the menu bar is hidden or shown.

7.1.12.4.15 TWebBrowser.OnNavigateComplete2 Event

Occurs immediately after the Web browser successfully navigates to a new location.

Pascal

```
property OnNavigateComplete2: TWebBrowserNavigateComplete2;
```

C++

```
__property TWebBrowserNavigateComplete2 OnNavigateComplete2;
```

Description

Write (see page 3789) an OnNavigateComplete2 event handler to take specific action when the Web browser successfully navigates to a new resource. The event can occur before the document is fully downloaded, but when it occurs at least part of the document must be received and a viewer for the document created.

Note: Unlike the OnDownloadComplete (see page 505) event, OnNavigateComplete2 does not occur if the operation is not successful.

See Also

OnDocumentComplete (see page 504)

OnDownloadBegin (see page 504)

Navigate (see page 470)

Navigate2 (see page 472)

OnDownloadComplete (see page 505)

OnBeforeNavigate2 (see page 503)

7.1.12.4.16 TWebBrowser.OnNavigateError Event

Pascal

```
property OnNavigateError: TWebBrowserNavigateError;
```

C++

```
__property TWebBrowserNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TWebBrowser.

7.1.12.4.17 TWebBrowser.OnNewWindow2 Event

Occurs when a new, hidden, non-navigated Web browser window is needed.

Pascal

```
property OnNewWindow2: TWebBrowserNewWindow2;
```

C++

```
__property TWebBrowserNewWindow2 OnNewWindow2;
```

Description

Write (see page 3789) an OnNewWindow2 to take specific action immediately before the Web browser creates a new window for displaying a resource. This window may be needed because the user shift-clicked on a link, the user right-clicked on a link and chose "open in new window", the frame for the target URL does not yet exist, or the Navigate (see page 470) or Navigate2 (see page 472) requested that the target document appear in a new window.

See Also

OnBeforeNavigate2 (see page 503)

7.1.12.4.18 TWebBrowser.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TWebBrowserPrintTemplateInstantiation;
```

C++

```
__property TWebBrowserPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TWebBrowser.

7.1.12.4.19 TWebBrowser.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TWebBrowserPrintTemplateTeardown;
```

C++

```
__property TWebBrowserPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TWebBrowser.

7.1.12.4.20 TWebBrowser.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TWebBrowserPrivacyImpactedStateChange;
```

C++

```
__property TWebBrowserPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.

7.1.12.4.21 TWebBrowser.OnProgressChange Event

Occurs when the progress of a download operation is updated.

Pascal

```
property OnProgressChange: TWebBrowserProgressChange;
```

C++

```
__property TWebBrowserProgressChange OnProgressChange;
```

Description

Write (see page 3789) an OnProgressChange event handler to provide visual feedback about the download process. For example, an OnProgressChange event handler can update a TProgressBar component or display the number of bytes downloaded so far.

See Also

OnDocumentComplete (see page 504)

OnDownloadBegin (see page 504)

Navigate (see page 470)

Navigate2 (see page 472)

OnDownloadComplete (see page 505)

TProgressBar

7.1.12.4.22 TWebBrowser.OnPropertyChange Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnPropertyChange: TWebBrowserPropertyChange;
```

C++

```
__property TWebBrowserPropertyChange OnPropertyChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnPropertyChange occurs with Internet Explorer when a property is modified using the PutProperty (see page 473) method.

7.1.12.4.23 TWebBrowser.OnQuit Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnQuit occurs when the Internet Explorer is about to shut down.

7.1.12.4.24 TWebBrowser.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TWebBrowserSetSecureLockIcon;
```

C++

```
__property TWebBrowserSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TWebBrowser.

7.1.12.4.25 TWebBrowser.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

7.1.12.4.26 TWebBrowser.OnStatusBar Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnStatusBar: TWebBrowserOnStatusBar;
```

C++

```
__property TWebBrowserOnStatusBar OnStatusBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnStatusBar occurs when the Internet Explorer first shows or hides its status bar.

7.1.12.4.27 TWebBrowser.OnStatusTextChange Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnStatusTextChange: TWebBrowserStatusTextChange;
```

C++

```
__property TWebBrowserStatusTextChange OnStatusTextChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnStatusTextChange occurs when the text displayed in the Internet Explorer's status bar changes.

7.1.12.4.28 TWebBrowser.OnTheaterMode Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnTheaterMode: TWebBrowserOnTheaterMode;
```

C++

```
__property TWebBrowserOnTheaterMode OnTheaterMode;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnTheaterMode occurs when the Internet Explorer changes into or out of theater mode.

7.1.12.4.29 TWebBrowser.OnTitleChange Event

Occurs when the title of a document in the WebBrowser (see page 779) control becomes available or changes.

Pascal

```
property OnTitleChange: TWebBrowserTitleChange;
```

C++

```
__property TWebBrowserTitleChange OnTitleChange;
```

Description

Write (see page 3789) an OnTitleChange event handler to respond when the Web browser obtains information about the document title. Because the title can change while an HTML page is downloading, LocationName (see page 487) is originally set to the URL of the document. After the title specified in the HTML page, if any, becomes available, LocationName (see page 487) is changed to reflect the actual title.

See Also

LocationName (see page 487)

7.1.12.4.30 TWebBrowser.OnToolBar Event

Not used in TWebBrowser (see page 462).

Pascal

```
property OnToolBar: TWebBrowserOnToolBar;
```

C++

```
__property TWebBrowserOnToolBar OnToolBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 462) interface because the IWebBrowser2 (see page 698) interface is shared with Internet Explorer. OnToolBar occurs when the Internet Explorer changes which tool bars are visible.

7.1.12.4.31 TWebBrowser.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TWebBrowserUpdatePageStatus;
```

C++

```
__property TWebBrowserUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TWebBrowser.

7.1.12.4.32 TWebBrowser.OnVisible Event

Occurs when the Web browser window is about to be shown or hidden.

Pascal

```
property OnVisible: TWebBrowserOnVisible;
```

C++

```
__property TWebBrowserOnVisible OnVisible;
```

Description

Write (see page 3789) an OnVisible event handler to make adjustments to compensate for the appearance or disappearance of the Web browser window.

See Also

Visible (see page 502)

7.1.12.4.33 TWebBrowser.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TWebBrowserWindowClosing;
```

C++

```
__property TWebBrowserWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TWebBrowser.

7.1.12.4.34 TWebBrowser.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TWebBrowserWindowSetHeight;
```

C++

```
__property TWebBrowserWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TWebBrowser.

7.1.12.4.35 TWebBrowser.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TWebBrowserWindowSetLeft;
```

C++

```
__property TWebBrowserWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TWebBrowser.

7.1.12.4.36 TWebBrowser.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TWebBrowserWindowSetResizable;
```

C++

```
__property TWebBrowserWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TWebBrowser.

7.1.12.4.37 TWebBrowser.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TWebBrowserWindowSetTop;
```

C++

```
__property TWebBrowserWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TWebBrowser.

7.1.12.4.38 TWebBrowser.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TWebBrowserWindowSetWidth;
```

C++

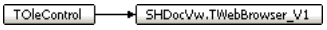
```
__property TWebBrowserWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TWebBrowser.

7.1.13 TWebBrowser_V1 Class

Class Hierarchy



Pascal

```
TWebBrowser_V1 = class(TOleControl);
```

C++

```
class TWebBrowser_V1 : public TOleControl;
```

File

SHDocVw

Description

This is class SHDocVw.TWebBrowser_V1.

7.1.13.1 TWebBrowser_V1 Members

The following tables list the members exposed by TWebBrowser_V1.

Protected Methods

	Name	Description
	CreateControl (see page 528)	This is CreateControl, a member of class TWebBrowser_V1.
	Get_Application (see page 528)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 528)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 529)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 529)	This is Get_Parent, a member of class TWebBrowser_V1.
V	InitControlData (see page 530)	This is InitControlData, a member of class TWebBrowser_V1.

Public Methods

	Name	Description
	GoBack (see page 529)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 529)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 529)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 530)	This is GoSearch, a member of class TWebBrowser_V1.
	Navigate (see page 530)	This is the overview for the Navigate method overload.
	Refresh (see page 532)	Repaints the control on the screen.
	Refresh2 (see page 532)	This is the overview for the Refresh2 method overload.
	Stop (see page 533)	This is Stop, a member of class TWebBrowser_V1.

Public Properties

	Name	Description
R	Application (see page 538)	This is Application, a member of class TWebBrowser_V1.
R	Busy (see page 538)	This is Busy, a member of class TWebBrowser_V1.
R	Container (see page 538)	This is Container, a member of class TWebBrowser_V1.

	ControlInterface (see page 538)	This is ControlInterface, a member of class TWebBrowser_V1.
	DefaultInterface (see page 538)	This is DefaultInterface, a member of class TWebBrowser_V1.
	Document (see page 539)	This is Document, a member of class TWebBrowser_V1.
	LocationName (see page 541)	This is LocationName, a member of class TWebBrowser_V1.
	LocationURL (see page 541)	This is LocationURL, a member of class TWebBrowser_V1.
	Parent (see page 541)	Indicates the parent of the control.
	TopLevelContainer (see page 551)	This is TopLevelContainer, a member of class TWebBrowser_V1.
	type_ (see page 551)	This is type_, a member of class TWebBrowser_V1.

Published Events

	Name	Description
	OnBeforeNavigate (see page 554)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 554)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 554)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 554)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 554)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 557)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 561)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 562)	Occurs when a control receives the input focus.
	OnExit (see page 564)	Occurs when the input focus shifts away from one control to another.
	OnFrameBeforeNavigate (see page 567)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 567)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 567)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 568)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 568)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 568)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 568)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 568)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 569)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 569)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 570)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 570)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 570)	This is OnWindowMove, a member of class TWebBrowser_V1.
	OnWindowResize (see page 570)	This is OnWindowResize, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 534)	Determines how the control aligns within its container (parent control).
	Anchors (see page 537)	Specifies how the control is anchored to its parent.
	DragCursor (see page 539)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 539)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ParentShowHint (see page 543)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 543)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 548)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 550)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 551)	Determines if the user can tab to a control.
	Visible (see page 551)	Determines whether the component appears onscreen.

7.1.13.2 TWebBrowser_V1 Methods

The methods of the TWebBrowser_V1 class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 528)	This is CreateControl, a member of class TWebBrowser_V1.
	Get_Application (see page 528)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 528)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 529)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 529)	This is Get_Parent, a member of class TWebBrowser_V1.
	InitControlData (see page 530)	This is InitControlData, a member of class TWebBrowser_V1.

Public Methods

	Name	Description
	GoBack (see page 529)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 529)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 529)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 530)	This is GoSearch, a member of class TWebBrowser_V1.
	Navigate (see page 530)	This is the overview for the Navigate method overload.
	Refresh (see page 532)	Repaints the control on the screen.
	Refresh2 (see page 532)	This is the overview for the Refresh2 method overload.
	Stop (see page 533)	This is Stop, a member of class TWebBrowser_V1.

7.1.13.2.1 TWebBrowser_V1.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser_V1.

7.1.13.2.2 TWebBrowser_V1.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser_V1.

7.1.13.2.3 TWebBrowser_V1.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser_V1.

7.1.13.2.4 TWebBrowser_V1.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser_V1.

7.1.13.2.5 TWebBrowser_V1.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser_V1.

7.1.13.2.6 TWebBrowser_V1.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TWebBrowser_V1.

7.1.13.2.7 TWebBrowser_V1.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TWebBrowser_V1.

7.1.13.2.8 TWebBrowser_V1.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TWebBrowser_V1.

7.1.13.2.9 TWebBrowser_V1.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TWebBrowser_V1.

7.1.13.2.10 TWebBrowser_V1.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser_V1.

7.1.13.2.11 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Navigate (WideString) (see page 530)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant) (see page 531)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant) (see page 531)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 531)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 531)	This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.11.1 TWebBrowser_V1.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.11.2 TWebBrowser_V1.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.11.3 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.11.4 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.11.5 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

7.1.13.2.12 TWebBrowser_V1.Refresh Method

Repaints the control on the screen.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh method to repaint the control immediately. Refresh calls the Repaint method. Use the Refresh and Repaint methods interchangeably.

See Also

Invalidate

Parent

Repaint

Update

7.1.13.2.13 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Refresh2 () (see page 532)	This is Refresh2, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Refresh2 (OleVariant) (see page 532)	This is Refresh2, a member of class TWebBrowser_V1.

7.1.13.2.13.1 TWebBrowser_V1.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

7.1.13.2.13.2 TWebBrowser_V1.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

7.1.13.2.14 TWebBrowser_V1.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TWebBrowser_V1.

7.1.13.3 TWebBrowser_V1 Properties

The properties of the TWebBrowser_V1 class are listed here.

Public Properties

	Name	Description
	Application (see page 538)	This is Application, a member of class TWebBrowser_V1.
	Busy (see page 538)	This is Busy, a member of class TWebBrowser_V1.
	Container (see page 538)	This is Container, a member of class TWebBrowser_V1.
	ControlInterface (see page 538)	This is ControlInterface, a member of class TWebBrowser_V1.
	DefaultInterface (see page 538)	This is DefaultInterface, a member of class TWebBrowser_V1.
	Document (see page 539)	This is Document, a member of class TWebBrowser_V1.
	LocationName (see page 541)	This is LocationName, a member of class TWebBrowser_V1.
	LocationURL (see page 541)	This is LocationURL, a member of class TWebBrowser_V1.
	Parent (see page 541)	Indicates the parent of the control.
	TopLevelContainer (see page 551)	This is TopLevelContainer, a member of class TWebBrowser_V1.
	type_ (see page 551)	This is type_, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 534)	Determines how the control aligns within its container (parent control).
	Anchors (see page 537)	Specifies how the control is anchored to its parent.
	DragCursor (see page 539)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 539)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ParentShowHint (see page 543)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 543)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 548)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 550)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 551)	Determines if the user can tab to a control.
	Visible (see page 551)	Determines whether the component appears onscreen.

7.1.13.3.1 TWebBrowser_V1.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
```

```
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

7.1.13.3.2 TWebBrowser_V1.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

7.1.13.3.3 TWebBrowser_V1.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TWebBrowser_V1.

7.1.13.3.4 TWebBrowser_V1.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TWebBrowser_V1.

7.1.13.3.5 TWebBrowser_V1.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TWebBrowser_V1.

7.1.13.3.6 TWebBrowser_V1.ControlInterface Property

Pascal

```
property ControlInterface: IWebBrowser;
```

C++

```
__property IWebBrowser ControlInterface;
```

Description

This is ControlInterface, a member of class TWebBrowser_V1.

7.1.13.3.7 TWebBrowser_V1.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser;
```

C++

```
__property IWebBrowser DefaultInterface;
```

Description

This is DefaultInterface, a member of class TWebBrowser_V1.

7.1.13.3.8 TWebBrowser_V1.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TWebBrowser_V1.

7.1.13.3.9 TWebBrowser_V1.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

7.1.13.3.10 TWebBrowser_V1.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
```

```
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

7.1.13.3.11 TWebBrowser_V1.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TWebBrowser_V1.

7.1.13.3.12 TWebBrowser_V1.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TWebBrowser_V1.

7.1.13.3.13 TWebBrowser_V1.Parent Property

Indicates the parent of the control.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use the Parent property to get or set the parent of this control. The parent of a control is the control that contains the control. For example, if an application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

To serve as a parent, a control must be an instance of a descendant of TWinControl.

When creating a new control at runtime, assign a Parent property value for the new control. Usually, this is a form, panel, group box, or some control that is designed to contain another. Changing the parent of a control moves the control onscreen so that it is displayed within the new parent. When the parent control moves, the child moves with the parent.

Some controls (such as ActiveX controls) are contained in native windows rather than in a parent VCL control. For these controls, the value of Parent is nil (Delphi) or NULL (C++) and the ParentWindow property specifies the window.

Note: The Parent property declared in TControl is similar to the Owner property declared in TComponent, in that the Parent of a control frees the control just as the Owner of a component frees that Component. However, the Parent of a control is always a windowed control that visually contains the control, and is responsible for writing the control to a stream when the form is saved. The Owner of a component is the component that was passed as a parameter in the constructor and, if assigned, initiates the process of saving all objects (including the control and its parent) when the form is saved.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Button1->Parent != GroupBox1) // second click?
  {
    RemoveControl(Button1);
    GroupBox1->InsertControl(Button1);
  }
}

/*
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
*/

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Button2->Parent = GroupBox1;
}
```

Delphi Examples:

```
{
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Button1.Parent <> GroupBox1) then // second click?
  begin
    RemoveControl(Button1);
    GroupBox1.InsertControl(Button1);
  end;
end;

{
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
}
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Parent := GroupBox1;
end;
```

See Also

Owner

Controls

HostDockSite

ParentWindow

7.1.13.3.14 TWebBrowser_V1.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

7.1.13.3.15 TWebBrowser_V1.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
```

```
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
    else
      Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

7.1.13.3.16 TWebBrowser_V1.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

7.1.13.3.17 TWebBrowser_V1.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

7.1.13.3.18 TWebBrowser_V1.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

7.1.13.3.19 TWebBrowser_V1.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TWebBrowser_V1.

7.1.13.3.20 TWebBrowser_V1.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TWebBrowser_V1.

7.1.13.3.21 TWebBrowser_V1.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

7.1.13.4 TWebBrowser_V1 Events

The events of the TWebBrowser_V1 class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate (see page 554)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 554)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 554)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 554)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 554)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 557)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 561)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 562)	Occurs when a control receives the input focus.
	OnExit (see page 564)	Occurs when the input focus shifts away from one control to another.
	OnFrameBeforeNavigate (see page 567)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 567)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 567)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 568)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 568)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 568)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 568)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 568)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 569)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 569)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 570)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 570)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 570)	This is OnWindowMove, a member of class TWebBrowser_V1.
	OnWindowResize (see page 570)	This is OnWindowResize, a member of class TWebBrowser_V1.

7.1.13.4.1 TWebBrowser_V1.OnBeforeNavigate Event

Pascal

```
property OnBeforeNavigate: TWebBrowser_V1BeforeNavigate;
```

C++

```
__property TWebBrowser_V1BeforeNavigate OnBeforeNavigate;
```

Description

This is OnBeforeNavigate, a member of class TWebBrowser_V1.

7.1.13.4.2 TWebBrowser_V1.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TWebBrowser_V1CommandStateChange;
```

C++

```
__property TWebBrowser_V1CommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TWebBrowser_V1.

7.1.13.4.3 TWebBrowser_V1.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TWebBrowser_V1.

7.1.13.4.4 TWebBrowser_V1.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TWebBrowser_V1.

7.1.13.4.5 TWebBrowser_V1.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
```

```
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

7.1.13.4.6 TWebBrowser_V1.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

7.1.13.4.7 TWebBrowser_V1.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

7.1.13.4.8 TWebBrowser_V1.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
```

```
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

7.1.13.4.9 TWebBrowser_V1.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
```

```
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
```

```
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

7.1.13.4.10 TWebBrowser_V1.OnFrameBeforeNavigate Event

Pascal

```
property OnFrameBeforeNavigate: TWebBrowser_V1FrameBeforeNavigate;
```

C++

```
__property TWebBrowser_V1FrameBeforeNavigate OnFrameBeforeNavigate;
```

Description

This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.

7.1.13.4.11 TWebBrowser_V1.OnFrameNavigateComplete Event

Pascal

```
property OnFrameNavigateComplete: TWebBrowser_V1FrameNavigateComplete;
```

C++

```
__property TWebBrowser_V1FrameNavigateComplete OnFrameNavigateComplete;
```

Description

This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.

7.1.13.4.12 TWebBrowser_V1.OnFrameNewWindow Event

Pascal

```
property OnFrameNewWindow: TWebBrowser_V1FrameNewWindow;
```

C++

```
__property TWebBrowser_V1FrameNewWindow OnFrameNewWindow;
```

Description

This is OnFrameNewWindow, a member of class TWebBrowser_V1.

7.1.13.4.13 TWebBrowser_V1.OnNavigateComplete Event

Pascal

```
property OnNavigateComplete: TWebBrowser_V1NavigateComplete;
```

C++

```
__property TWebBrowser_V1NavigateComplete OnNavigateComplete;
```

Description

This is OnNavigateComplete, a member of class TWebBrowser_V1.

7.1.13.4.14 TWebBrowser_V1.OnNewWindow Event

Pascal

```
property OnNewWindow: TWebBrowser_V1NewWindow;
```

C++

```
__property TWebBrowser_V1NewWindow OnNewWindow;
```

Description

This is OnNewWindow, a member of class TWebBrowser_V1.

7.1.13.4.15 TWebBrowser_V1.OnProgressChange Event

Pascal

```
property OnProgressChange: TWebBrowser_V1ProgressChange;
```

C++

```
__property TWebBrowser_V1ProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TWebBrowser_V1.

7.1.13.4.16 TWebBrowser_V1.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TWebBrowser_V1PropertyChange;
```

C++

```
__property TWebBrowser_V1PropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TWebBrowser_V1.

7.1.13.4.17 TWebBrowser_V1.OnQuit Event

Pascal

```
property OnQuit: TWebBrowser_V1Quit;
```

C++

```
__property TWebBrowser_V1Quit OnQuit;
```

Description

This is OnQuit, a member of class TWebBrowser_V1.

7.1.13.4.18 TWebBrowser_V1.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

7.1.13.4.19 TWebBrowser_V1.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TWebBrowser_V1StatusTextChange;
```

C++

```
__property TWebBrowser_V1StatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TWebBrowser_V1.

7.1.13.4.20 TWebBrowser_V1.OnTitleChange Event

Pascal

```
property OnTitleChange: TWebBrowser_V1TitleChange;
```

C++

```
__property TWebBrowser_V1TitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TWebBrowser_V1.

7.1.13.4.21 TWebBrowser_V1.OnWindowActivate Event

Pascal

```
property OnWindowActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowActivate;
```

Description

This is OnWindowActivate, a member of class TWebBrowser_V1.

7.1.13.4.22 TWebBrowser_V1.OnWindowMove Event

Pascal

```
property OnWindowMove: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowMove;
```

Description

This is OnWindowMove, a member of class TWebBrowser_V1.

7.1.13.4.23 TWebBrowser_V1.OnWindowResize Event

Pascal

```
property OnWindowResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowResize;
```

Description

This is OnWindowResize, a member of class TWebBrowser_V1.

7.2 Interfaces

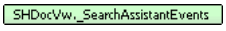
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 571)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 573)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 574)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 576)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 581)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 590)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 594)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 598)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 601)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 611)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 613)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 621)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 624)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 632)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 639)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 641)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 643)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 645)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 650)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 654)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 663)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 671)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 675)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 679)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 683)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 687)	This is class SHDocVw.IWebBrowser.
	IWebBrowser2 (see page 698)	This is class SHDocVw.IWebBrowser2.
	IWebBrowser2Disp (see page 707)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 720)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 729)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 739)	This is class SHDocVw.IWebBrowserDisp.

7.2.1 _SearchAssistantEvents Interface

Class Hierarchy



Pascal

```
_SearchAssistantEvents = dispinterface;
```

C++

```
class _SearchAssistantEvents;
```

File

SHDocVw

Description

This is class SHDocVw._SearchAssistantEvents.

7.2.1.1 _SearchAssistantEvents Members

The following tables list the members exposed by _SearchAssistantEvents.

Public Methods

	Name	Description
	OnNewSearch (see page 572)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 572)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

7.2.1.2 _SearchAssistantEvents Methods

The methods of the _SearchAssistantEvents class are listed here.

Public Methods

	Name	Description
	OnNewSearch (see page 572)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 572)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

7.2.1.2.1 _SearchAssistantEvents.OnNewSearch Method

Pascal

```
procedure OnNewSearch;
```

C++

```
__fastcall OnNewSearch();
```

Description

This is OnNewSearch, a member of class _SearchAssistantEvents.

7.2.1.2.2 _SearchAssistantEvents.OnNextMenuSelect Method

Pascal

```
procedure OnNextMenuSelect(idItem: Integer);
```

C++

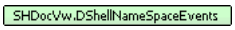
```
__fastcall OnNextMenuSelect(int idItem);
```

Description

This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

7.2.2 DShellNameSpaceEvents Interface

Class Hierarchy



Pascal

```
DShellNameSpaceEvents = dispinterface;
```

C++

```
class DShellNameSpaceEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellNameSpaceEvents.

7.2.2.1 DShellNameSpaceEvents Members

The following tables list the members exposed by DShellNameSpaceEvents.

Public Methods

	Name	Description
	DoubleClick (see page 573)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 574)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 574)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 574)	This is SelectionChange, a member of class DShellNameSpaceEvents.

7.2.2.2 DShellNameSpaceEvents Methods

The methods of the DShellNameSpaceEvents class are listed here.

Public Methods

	Name	Description
	DoubleClick (see page 573)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 574)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 574)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 574)	This is SelectionChange, a member of class DShellNameSpaceEvents.

7.2.2.2.1 DShellNameSpaceEvents.DoubleClick Method

Pascal

```
procedure DoubleClick;
```

C++

```
__fastcall DoubleClick();
```

Description

This is DoubleClick, a member of class DShellNameSpaceEvents.

7.2.2.2.2 DShellNameSpaceEvents.FavoritesSelectionChange Method

Pascal

```
procedure FavoritesSelectionChange(cItems: Integer; hItem: Integer; const strName: WideString;
const strUrl: WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline:
Integer);
```

C++

```
__fastcall FavoritesSelectionChange(int cItems, int hItem, const BSTR strName, const BSTR
strUrl, int cVisits, const BSTR strDate, int fAvailableOffline);
```

Description

This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.

7.2.2.2.3 DShellNameSpaceEvents.Initialized Method

Pascal

```
procedure Initialized;
```

C++

```
__fastcall Initialized();
```

Description

This is Initialized, a member of class DShellNameSpaceEvents.

7.2.2.2.4 DShellNameSpaceEvents.SelectionChange Method

Pascal

```
procedure SelectionChange;
```

C++

```
__fastcall SelectionChange();
```

Description

This is SelectionChange, a member of class DShellNameSpaceEvents.

7.2.3 DShellWindowsEvents Interface

Class Hierarchy

Pascal

```
DShellWindowsEvents = dispinterface;
```

C++

```
class DShellWindowsEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellWindowsEvents.

7.2.3.1 DShellWindowsEvents Members

The following tables list the members exposed by DShellWindowsEvents.

Public Methods

	Name	Description
	WindowRegistered (see page 575)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 575)	This is WindowRevoked, a member of class DShellWindowsEvents.

7.2.3.2 DShellWindowsEvents Methods

The methods of the DShellWindowsEvents class are listed here.

Public Methods

	Name	Description
	WindowRegistered (see page 575)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 575)	This is WindowRevoked, a member of class DShellWindowsEvents.

7.2.3.2.1 DShellWindowsEvents.WindowRegistered Method

Pascal

```
procedure WindowRegistered(lCookie: Integer);
```

C++

```
__fastcall WindowRegistered(int lCookie);
```

Description

This is WindowRegistered, a member of class DShellWindowsEvents.

7.2.3.2.2 DShellWindowsEvents.WindowRevoked Method

Pascal

```
procedure WindowRevoked(lCookie: Integer);
```

C++

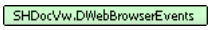
```
__fastcall WindowRevoked(int lCookie);
```

Description

This is WindowRevoked, a member of class DShellWindowsEvents.

7.2.4 DWebBrowserEvents Interface

Class Hierarchy



Pascal

```
DWebBrowserEvents = dispinterface;
```

C++

```
class DWebBrowserEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents.

7.2.4.1 DWebBrowserEvents Members

The following tables list the members exposed by DWebBrowserEvents.

Public Methods

	Name	Description
	BeforeNavigate (see page 577)	This is BeforeNavigate, a member of class DWebBrowserEvents.
	CommandStateChange (see page 577)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 577)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 578)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 578)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 578)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 578)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 579)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 579)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 579)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 579)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 579)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 580)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 580)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 580)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 580)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 581)	This is WindowResize, a member of class DWebBrowserEvents.

7.2.4.2 DWebBrowserEvents Methods

The methods of the DWebBrowserEvents class are listed here.

Public Methods

	Name	Description
	BeforeNavigate (see page 577)	This is BeforeNavigate, a member of class DWebBrowserEvents.

	CommandStateChange (see page 577)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 577)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 578)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 578)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 578)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 578)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 579)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 579)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 579)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 579)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 579)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 580)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 580)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 580)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 580)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 581)	This is WindowResize, a member of class DWebBrowserEvents.

7.2.4.2.1 DWebBrowserEvents.BeforeNavigate Method

Pascal

```
procedure BeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is BeforeNavigate, a member of class DWebBrowserEvents.

7.2.4.2.2 DWebBrowserEvents.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents.

7.2.4.2.3 DWebBrowserEvents.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents.

7.2.4.2.4 DWebBrowserEvents.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents.

7.2.4.2.5 DWebBrowserEvents.FrameBeforeNavigate Method

Pascal

```
procedure FrameBeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall FrameBeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName,
OleVariant PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is FrameBeforeNavigate, a member of class DWebBrowserEvents.

7.2.4.2.6 DWebBrowserEvents.FrameNavigateComplete Method

Pascal

```
procedure FrameNavigateComplete(const URL: WideString);
```

C++

```
__fastcall FrameNavigateComplete(const BSTR URL);
```

Description

This is FrameNavigateComplete, a member of class DWebBrowserEvents.

7.2.4.2.7 DWebBrowserEvents.FrameNewWindow Method

Pascal

```
procedure FrameNewWindow(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall FrameNewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is FrameNewWindow, a member of class DWebBrowserEvents.

7.2.4.2.8 DWebBrowserEvents.NavigateComplete Method

Pascal

```
procedure NavigateComplete(const URL: WideString);
```

C++

```
__fastcall NavigateComplete(const BSTR URL);
```

Description

This is NavigateComplete, a member of class DWebBrowserEvents.

7.2.4.2.9 DWebBrowserEvents.NewWindow Method

Pascal

```
procedure NewWindow(const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall NewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is NewWindow, a member of class DWebBrowserEvents.

7.2.4.2.10 DWebBrowserEvents.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents.

7.2.4.2.11 DWebBrowserEvents.PropertyChange Method

Pascal

```
procedure PropertyChange(const Property_: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR Property_);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents.

7.2.4.2.12 DWebBrowserEvents.Quit Method

Pascal

```
procedure Quit(var Cancel: WordBool);
```

C++

```
__fastcall Quit(WordBool Cancel);
```

Description

This is Quit, a member of class DWebBrowserEvents.

7.2.4.2.13 DWebBrowserEvents.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents.

7.2.4.2.14 DWebBrowserEvents.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents.

7.2.4.2.15 DWebBrowserEvents.WindowActivate Method

Pascal

```
procedure WindowActivate;
```

C++

```
__fastcall WindowActivate();
```

Description

This is WindowActivate, a member of class DWebBrowserEvents.

7.2.4.2.16 DWebBrowserEvents.WindowMove Method

Pascal

```
procedure WindowMove;
```

C++

```
__fastcall WindowMove();
```

Description

This is WindowMove, a member of class DWebBrowserEvents.

7.2.4.2.17 DWebBrowserEvents.WindowResize Method

Pascal

```
procedure WindowResize;
```

C++

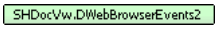
```
__fastcall WindowResize();
```

Description

This is WindowResize, a member of class DWebBrowserEvents.

7.2.5 DWebBrowserEvents2 Interface

Class Hierarchy



Pascal

```
DWebBrowserEvents2 = dispinterface;
```

C++

```
class DWebBrowserEvents2;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents2.

7.2.5.1 DWebBrowserEvents2 Members

The following tables list the members exposed by DWebBrowserEvents2.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 583)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 583)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 583)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 583)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 584)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 584)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 584)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 584)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 584)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 585)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 585)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 585)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 585)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 586)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 586)	This is OnTheaterMode, a member of class DWebBrowserEvents2.

	OnToolBar (see page 586)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 586)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 586)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 587)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 587)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 587)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 587)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 588)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 588)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 588)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 588)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 588)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 589)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 589)	This is WindowSetLeft, a member of class DWebBrowserEvents2.
	WindowSetResizable (see page 589)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 589)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 590)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

7.2.5.2 DWebBrowserEvents2 Methods

The methods of the DWebBrowserEvents2 class are listed here.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 583)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 583)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 583)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 583)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 584)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 584)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 584)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 584)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 584)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 585)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 585)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 585)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 585)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 586)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 586)	This is OnTheaterMode, a member of class DWebBrowserEvents2.
	OnToolBar (see page 586)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 586)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 586)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 587)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 587)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 587)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 587)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 588)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 588)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 588)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 588)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 588)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 589)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 589)	This is WindowSetLeft, a member of class DWebBrowserEvents2.

	WindowSetResizable (see page 589)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 589)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 590)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

7.2.5.2.1 DWebBrowserEvents2.BeforeNavigate2 Method

Pascal

```
procedure BeforeNavigate2(const pDisp: IDispatch; const URL: OleVariant; const Flags:
OleVariant; const TargetFrameName: OleVariant; const PostData: OleVariant; const Headers:
OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate2(const IDispatch pDisp, const OleVariant URL, const OleVariant
Flags, const OleVariant TargetFrameName, const OleVariant PostData, const OleVariant Headers,
WordBool Cancel);
```

Description

This is BeforeNavigate2, a member of class DWebBrowserEvents2.

7.2.5.2.2 DWebBrowserEvents2.ClientToHostWindow Method

Pascal

```
procedure ClientToHostWindow(var CX: Integer; var CY: Integer);
```

C++

```
__fastcall ClientToHostWindow(int CX, int CY);
```

Description

This is ClientToHostWindow, a member of class DWebBrowserEvents2.

7.2.5.2.3 DWebBrowserEvents2.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents2.

7.2.5.2.4 DWebBrowserEvents2.DocumentComplete Method

Pascal

```
procedure DocumentComplete(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall DocumentComplete(const IDispatch pDisp, OleVariant URL);
```

Description

This is DocumentComplete, a member of class DWebBrowserEvents2.

7.2.5.2.5 DWebBrowserEvents2.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents2.

7.2.5.2.6 DWebBrowserEvents2.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents2.

7.2.5.2.7 DWebBrowserEvents2.FileDownload Method

Pascal

```
procedure FileDownload(var Cancel: WordBool);
```

C++

```
__fastcall FileDownload(WordBool Cancel);
```

Description

This is FileDownload, a member of class DWebBrowserEvents2.

7.2.5.2.8 DWebBrowserEvents2.NavigateComplete2 Method

Pascal

```
procedure NavigateComplete2(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall NavigateComplete2(const IDispatch pDisp, OleVariant URL);
```

Description

This is NavigateComplete2, a member of class DWebBrowserEvents2.

7.2.5.2.9 DWebBrowserEvents2.NavigateError Method

Pascal

```
procedure NavigateError(const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant;
var StatusCode: OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall NavigateError(const IDispatch pDisp, OleVariant URL, OleVariant Frame, OleVariant StatusCode, WordBool Cancel);
```

Description

This is NavigateError, a member of class DWebBrowserEvents2.

7.2.5.2.10 DWebBrowserEvents2.NewWindow2 Method

Pascal

```
procedure NewWindow2(var ppDisp: IDispatch; var Cancel: WordBool);
```

C++

```
__fastcall NewWindow2(IDispatch ppDisp, WordBool Cancel);
```

Description

This is NewWindow2, a member of class DWebBrowserEvents2.

7.2.5.2.11 DWebBrowserEvents2.OnFullScreen Method

Pascal

```
procedure OnFullScreen(FullScreen: WordBool);
```

C++

```
__fastcall OnFullScreen(WordBool FullScreen);
```

Description

This is OnFullScreen, a member of class DWebBrowserEvents2.

7.2.5.2.12 DWebBrowserEvents2.OnMenuBar Method

Pascal

```
procedure OnMenuBar(MenuBar: WordBool);
```

C++

```
__fastcall OnMenuBar(WordBool MenuBar);
```

Description

This is OnMenuBar, a member of class DWebBrowserEvents2.

7.2.5.2.13 DWebBrowserEvents2.OnQuit Method

Pascal

```
procedure OnQuit;
```

C++

```
__fastcall OnQuit();
```

Description

This is OnQuit, a member of class DWebBrowserEvents2.

7.2.5.2.14 DWebBrowserEvents2.OnStatusBar Method

Pascal

```
procedure OnStatusBar(StatusBar: WordBool);
```

C++

```
__fastcall OnStatusBar(WordBool StatusBar);
```

Description

This is OnStatusBar, a member of class DWebBrowserEvents2.

7.2.5.2.15 DWebBrowserEvents2.OnTheaterMode Method

Pascal

```
procedure OnTheaterMode(TheaterMode: WordBool);
```

C++

```
__fastcall OnTheaterMode(WordBool TheaterMode);
```

Description

This is OnTheaterMode, a member of class DWebBrowserEvents2.

7.2.5.2.16 DWebBrowserEvents2.OnToolBar Method

Pascal

```
procedure OnToolBar(ToolBar: WordBool);
```

C++

```
__fastcall OnToolBar(WordBool ToolBar);
```

Description

This is OnToolBar, a member of class DWebBrowserEvents2.

7.2.5.2.17 DWebBrowserEvents2.OnVisible Method

Pascal

```
procedure OnVisible(Visible: WordBool);
```

C++

```
__fastcall OnVisible(WordBool Visible);
```

Description

This is OnVisible, a member of class DWebBrowserEvents2.

7.2.5.2.18 DWebBrowserEvents2.PrintTemplateInstantiation Method

Pascal

```
procedure PrintTemplateInstantiation(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateInstantiation(const IDispatch pDisp);
```

Description

This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.

7.2.5.2.19 DWebBrowserEvents2.PrintTemplateTeardown Method

Pascal

```
procedure PrintTemplateTeardown(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateTeardown(const IDispatch pDisp);
```

Description

This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.

7.2.5.2.20 DWebBrowserEvents2.PrivacyImpactedStateChange Method

Pascal

```
procedure PrivacyImpactedStateChange(bImpacted: WordBool);
```

C++

```
__fastcall PrivacyImpactedStateChange(WordBool bImpacted);
```

Description

This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.

7.2.5.2.21 DWebBrowserEvents2.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents2.

7.2.5.2.22 DWebBrowserEvents2.PropertyChange Method

Pascal

```
procedure PropertyChange(const szProperty: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR szProperty);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents2.

7.2.5.2.23 DWebBrowserEvents2.SetSecureLockIcon Method

Pascal

```
procedure SetSecureLockIcon(SecureLockIcon: Integer);
```

C++

```
__fastcall SetSecureLockIcon(int SecureLockIcon);
```

Description

This is SetSecureLockIcon, a member of class DWebBrowserEvents2.

7.2.5.2.24 DWebBrowserEvents2.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents2.

7.2.5.2.25 DWebBrowserEvents2.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents2.

7.2.5.2.26 DWebBrowserEvents2.UpdatePageStatus Method

Pascal

```
procedure UpdatePageStatus(const pDisp: IDispatch; var nPage: OleVariant; var fDone:
OleVariant);
```

C++

```
__fastcall UpdatePageStatus(const IDispatch pDisp, OleVariant nPage, OleVariant fDone);
```

Description

This is UpdatePageStatus, a member of class DWebBrowserEvents2.

7.2.5.2.27 DWebBrowserEvents2.WindowClosing Method

Pascal

```
procedure WindowClosing(IsChildWindow: WordBool; var Cancel: WordBool);
```

C++

```
__fastcall WindowClosing(WordBool IsChildWindow, WordBool Cancel);
```

Description

This is WindowClosing, a member of class DWebBrowserEvents2.

7.2.5.2.28 DWebBrowserEvents2.WindowSetHeight Method

Pascal

```
procedure WindowSetHeight(Height: Integer);
```

C++

```
__fastcall WindowSetHeight(int Height);
```

Description

This is WindowSetHeight, a member of class DWebBrowserEvents2.

7.2.5.2.29 DWebBrowserEvents2.WindowSetLeft Method

Pascal

```
procedure WindowSetLeft(Left: Integer);
```

C++

```
__fastcall WindowSetLeft(int Left);
```

Description

This is WindowSetLeft, a member of class DWebBrowserEvents2.

7.2.5.2.30 DWebBrowserEvents2.WindowSetResizable Method

Pascal

```
procedure WindowSetResizable(Resizable: WordBool);
```

C++

```
__fastcall WindowSetResizable(WordBool Resizable);
```

Description

This is WindowSetResizable, a member of class DWebBrowserEvents2.

7.2.5.2.31 DWebBrowserEvents2.WindowSetTop Method

Pascal

```
procedure WindowSetTop(Top: Integer);
```

C++

```
__fastcall WindowSetTop(int Top);
```

Description

This is WindowSetTop, a member of class DWebBrowserEvents2.

7.2.5.2.32 DWebBrowserEvents2.WindowSetWidth Method

Pascal

```
procedure WindowSetWidth(Width: Integer);
```

C++

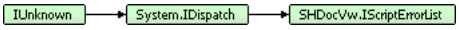
```
__fastcall WindowSetWidth(int Width);
```

Description

This is WindowSetWidth, a member of class DWebBrowserEvents2.

7.2.6 IScriptErrorList Interface

Class Hierarchy



Pascal

```
IScriptErrorList = interface(IDispatch);
```

C++

```
__interface IScriptErrorList : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorList.

7.2.6.1 IScriptErrorList Members

The following tables list the members exposed by IScriptErrorList.

Public Methods

	Name	Description
	advanceError (see page 591)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 591)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 591)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 592)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 592)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 592)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 592)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 593)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 593)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 593)	This is getErrorUrl, a member of class IScriptErrorList.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	getPerErrorDisplay (see page 593)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.

	retreatError (see page 593)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 594)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 594)	This is setPerErrorDisplay, a member of class IScriptErrorList.

7.2.6.2 IScriptErrorList Methods

The methods of the IScriptErrorList class are listed here.

Public Methods

	Name	Description
	advanceError (see page 591)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 591)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 591)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 592)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 592)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 592)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 592)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 593)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 593)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 593)	This is getErrorUrl, a member of class IScriptErrorList.
	getPerErrorDisplay (see page 593)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	retreatError (see page 593)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 594)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 594)	This is setPerErrorDisplay, a member of class IScriptErrorList.

7.2.6.2.1 IScriptErrorList.advanceError Method

Pascal

```
procedure advanceError; safecall;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorList.

7.2.6.2.2 IScriptErrorList.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer; safecall;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorList.

7.2.6.2.3 IScriptErrorList.canRetreatError Method

Pascal

```
function canRetreatError: Integer; safecall;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorList.

7.2.6.2.4 IScriptErrorList.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer; safecall;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorList.

7.2.6.2.5 IScriptErrorList.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer; safecall;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorList.

7.2.6.2.6 IScriptErrorList.getErrorChar Method

Pascal

```
function getErrorChar: Integer; safecall;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorList.

7.2.6.2.7 IScriptErrorList.getErrorCode Method

Pascal

```
function getErrorCode: Integer; safecall;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorList.

7.2.6.2.8 IScriptErrorList.getErrorLine Method

Pascal

```
function getErrorLine: Integer; safecall;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorList.

7.2.6.2.9 IScriptErrorList.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorList.

7.2.6.2.10 IScriptErrorList.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorList.

7.2.6.2.11 IScriptErrorList.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer; safecall;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorList.

7.2.6.2.12 IScriptErrorList.retreatError Method

Pascal

```
procedure retreatError; safecall;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorList.

7.2.6.2.13 IScriptErrorList.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer); safecall;
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorList.

7.2.6.2.14 IScriptErrorList.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer); safecall;
```

C++

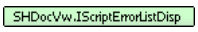
```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorList.

7.2.7 IScriptErrorListDisp Interface

Class Hierarchy



Pascal

```
IScriptErrorListDisp = dispinterface;
```

C++

```
class IScriptErrorListDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorListDisp.

7.2.7.1 IScriptErrorListDisp Members

The following tables list the members exposed by IScriptErrorListDisp.

Public Methods

	Name	Description
	advanceError (see page 595)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 596)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 596)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 596)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 596)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 596)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 597)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 597)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 597)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 597)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 598)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 598)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 598)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 598)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

7.2.7.2 IScriptErrorListDisp Methods

The methods of the IScriptErrorListDisp class are listed here.

Public Methods

	Name	Description
	advanceError (see page 595)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 596)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 596)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 596)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 596)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 596)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 597)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 597)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 597)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 597)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 598)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 598)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 598)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 598)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

7.2.7.2.1 IScriptErrorListDisp.advanceError Method

Pascal

```
procedure advanceError;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorListDisp.

7.2.7.2.2 IScriptErrorListDisp.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorListDisp.

7.2.7.2.3 IScriptErrorListDisp.canRetreatError Method

Pascal

```
function canRetreatError: Integer;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorListDisp.

7.2.7.2.4 IScriptErrorListDisp.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.

7.2.7.2.5 IScriptErrorListDisp.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.

7.2.7.2.6 IScriptErrorListDisp.getErrorChar Method

Pascal

```
function getErrorChar: Integer;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorListDisp.

7.2.7.2.7 IScriptErrorListDisp.getErrorCode Method

Pascal

```
function getErrorCode: Integer;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorListDisp.

7.2.7.2.8 IScriptErrorListDisp.getErrorLine Method

Pascal

```
function getErrorLine: Integer;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorListDisp.

7.2.7.2.9 IScriptErrorListDisp.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorListDisp.

7.2.7.2.10 IScriptErrorListDisp.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorListDisp.

7.2.7.2.11 IScriptErrorListDisp.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorListDisp.

7.2.7.2.12 IScriptErrorListDisp.retreatError Method

Pascal

```
procedure retreatError;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorListDisp.

7.2.7.2.13 IScriptErrorListDisp.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer);
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.

7.2.7.2.14 IScriptErrorListDisp.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer);
```

C++

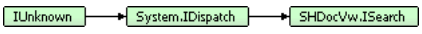
```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

7.2.8 ISearch Interface

Class Hierarchy



Pascal

```
ISearch = interface(IDispatch);
```

C++

```
__interface ISearch : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearch.

7.2.8.1 ISearch Members

The following tables list the members exposed by ISearch.

Public Methods

	Name	Description
	Get_Id (see page 599)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 600)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 600)	This is Get_URL, a member of class ISearch.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.

Public Properties

	Name	Description
R	Id (see page 600)	This is Id, a member of class ISearch.
R	Title (see page 600)	This is Title, a member of class ISearch.
R	URL (see page 601)	This is URL, a member of class ISearch.

7.2.8.2 ISearch Methods

The methods of the ISearch class are listed here.

Public Methods

	Name	Description
	Get_Id (see page 599)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 600)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 600)	This is Get_URL, a member of class ISearch.

7.2.8.2.1 ISearch.Get_Id Method

Pascal

```
function Get_Id: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Id();
```

Description

This is Get_Id, a member of class ISearch.

7.2.8.2.2 ISearch.Get_Title Method

Pascal

```
function Get_Title: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Title();
```

Description

This is Get_Title, a member of class ISearch.

7.2.8.2.3 ISearch.Get_URL Method

Pascal

```
function Get_URL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_URL();
```

Description

This is Get_URL, a member of class ISearch.

7.2.8.3 ISearch Properties

The properties of the ISearch class are listed here.

Public Properties

	Name	Description
	Id (see page 600)	This is Id, a member of class ISearch.
	Title (see page 600)	This is Title, a member of class ISearch.
	URL (see page 601)	This is URL, a member of class ISearch.

7.2.8.3.1 ISearch.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearch.

7.2.8.3.2 ISearch.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearch.

7.2.8.3.3 ISearch.URL Property

Pascal

```
property URL: WideString;
```

C++

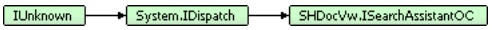
```
__property BSTR URL;
```

Description

This is URL, a member of class ISearch.

7.2.9 ISearchAssistantOC Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC = interface(IDispatch);
```

C++

```
__interface ISearchAssistantOC : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC.

7.2.9.1 ISearchAssistantOC Members

The following tables list the members exposed by ISearchAssistantOC.

Public Methods

	Name	Description
	AddNextMenuItem (see page 603)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 603)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 603)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 604)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 604)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 604)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 604)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 604)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 605)	This is Get_ASSetting, a member of class ISearchAssistantOC.

	Get_InWebFolder (see page 605)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 605)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 605)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 606)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_Version (see page 606)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 606)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 606)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 606)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 607)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 607)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 607)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 607)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 608)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 608)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 608)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 608)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 608)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 609)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 609)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 609)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 610)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 610)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 610)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 610)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 611)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 611)	This is Version, a member of class ISearchAssistantOC.

7.2.9.2 ISearchAssistantOC Methods

The methods of the ISearchAssistantOC class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 603)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 603)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 603)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 604)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 604)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 604)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 604)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 604)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 605)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 605)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 605)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 605)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 606)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.

	Get_Version (see page 606)	This is Get_Version, a member of class ISearchAssistantOC.
	GetProperty (see page 606)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 606)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	IsRestricted (see page 606)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 607)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 607)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 607)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 607)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 608)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 608)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 608)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 608)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 608)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 609)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

7.2.9.2.1 ISearchAssistantOC.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer); safecall;
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC.

7.2.9.2.2 ISearchAssistantOC.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString; safecall;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC.

7.2.9.2.3 ISearchAssistantOC.FindComputer Method

Pascal

```
procedure FindComputer; safecall;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC.

7.2.9.2.4 ISearchAssistantOC.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders; safecall;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC.

7.2.9.2.5 ISearchAssistantOC.FindOnWeb Method

Pascal

```
procedure FindOnWeb; safecall;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC.

7.2.9.2.6 ISearchAssistantOC.FindPeople Method

Pascal

```
procedure FindPeople; safecall;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC.

7.2.9.2.7 ISearchAssistantOC.FindPrinter Method

Pascal

```
procedure FindPrinter; safecall;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC.

7.2.9.2.8 ISearchAssistantOC.Get_ASProvider Method

Pascal

```
function Get_ASProvider: WideString; safecall;
```

C++

```
__fastcall BSTR Get_ASProvider();
```

Description

This is Get_ASProvider, a member of class ISearchAssistantOC.

7.2.9.2.9 ISearchAssistantOC.Get_ASSetting Method

Pascal

```
function Get_ASSetting: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ASSetting();
```

Description

This is Get_ASSetting, a member of class ISearchAssistantOC.

7.2.9.2.10 ISearchAssistantOC.Get_InWebFolder Method

Pascal

```
function Get_InWebFolder: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_InWebFolder();
```

Description

This is Get_InWebFolder, a member of class ISearchAssistantOC.

7.2.9.2.11 ISearchAssistantOC.Get_SearchAssistantDefault Method

Pascal

```
function Get_SearchAssistantDefault: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchAssistantDefault();
```

Description

This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.

7.2.9.2.12 ISearchAssistantOC.Get_Searches Method

Pascal

```
function Get_Searches: ISearches; safecall;
```

C++

```
__fastcall ISearches Get_Searches();
```

Description

This is Get_Searches, a member of class ISearchAssistantOC.

7.2.9.2.13 ISearchAssistantOC.Get_ShellFeaturesEnabled Method

Pascal

```
function Get_ShellFeaturesEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShellFeaturesEnabled();
```

Description

This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.

7.2.9.2.14 ISearchAssistantOC.Get_Version Method

Pascal

```
function Get_Version: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Version();
```

Description

This is Get_Version, a member of class ISearchAssistantOC.

7.2.9.2.15 ISearchAssistantOC.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString; safecall;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC.

7.2.9.2.16 ISearchAssistantOC.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
safecall;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC.

7.2.9.2.17 ISearchAssistantOC.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC.

7.2.9.2.18 ISearchAssistantOC.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch; safecall;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.

7.2.9.2.19 ISearchAssistantOC.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate; safecall;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC.

7.2.9.2.20 ISearchAssistantOC.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged; safecall;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.

7.2.9.2.21 ISearchAssistantOC.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString); safecall;
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC.

7.2.9.2.22 ISearchAssistantOC.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString); safecall;
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC.

7.2.9.2.23 ISearchAssistantOC.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu; safecall;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC.

7.2.9.2.24 ISearchAssistantOC.Set_ASProvider Method

Pascal

```
procedure Set_ASProvider(const pProvider: WideString); safecall;
```

C++

```
__fastcall Set_ASProvider(const BSTR pProvider);
```

Description

This is Set_ASProvider, a member of class ISearchAssistantOC.

7.2.9.2.25 ISearchAssistantOC.Set_ASSetting Method

Pascal

```
procedure Set_ASSetting(pSetting: SYSINT); safecall;
```

C++

```
__fastcall Set_ASSetting(SYSINT pSetting);
```

Description

This is Set_ASSetting, a member of class ISearchAssistantOC.

7.2.9.2.26 ISearchAssistantOC.Set_EventHandled Method

Pascal

```
procedure Set_EventHandled(Param1: WordBool); safecall;
```

C++

```
__fastcall Set_EventHandled(WordBool Param1);
```

Description

This is Set_EventHandled, a member of class ISearchAssistantOC.

7.2.9.2.27 ISearchAssistantOC.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString); safecall;
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

7.2.9.3 ISearchAssistantOC Properties

The properties of the ISearchAssistantOC class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 609)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 609)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 610)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 610)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 610)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 610)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 611)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 611)	This is Version, a member of class ISearchAssistantOC.

7.2.9.3.1 ISearchAssistantOC.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC.

7.2.9.3.2 ISearchAssistantOC.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC.

7.2.9.3.3 ISearchAssistantOC.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC.

7.2.9.3.4 ISearchAssistantOC.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC.

7.2.9.3.5 ISearchAssistantOC.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC.

7.2.9.3.6 ISearchAssistantOC.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC.

7.2.9.3.7 ISearchAssistantOC.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.

7.2.9.3.8 ISearchAssistantOC.Version Property

Pascal

```
property Version: SYSINT;
```

C++

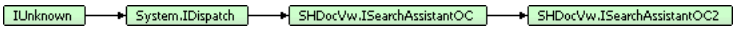
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC.

7.2.10 ISearchAssistantOC2 Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC2 = interface(ISearchAssistantOC);
```

C++

```
__interface ISearchAssistantOC2 : public ISearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2.

7.2.10.1 ISearchAssistantOC2 Members

The following tables list the members exposed by ISearchAssistantOC2.

Public Methods

	Name	Description
	AddNextMenuItem (see page 603)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 603)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 603)	This is FindComputer, a member of class ISearchAssistantOC.

	FindFilesOrFolders (see page 604)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 604)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 604)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 604)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 604)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 605)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 605)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 605)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 605)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 606)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 613)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_Version (see page 606)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 606)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 606)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 606)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 607)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 607)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 607)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 607)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 608)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 608)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 608)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 608)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 608)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 609)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 609)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 609)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 610)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 610)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 610)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 610)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 611)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 613)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
R	Version (see page 611)	This is Version, a member of class ISearchAssistantOC.

7.2.10.2 ISearchAssistantOC2 Methods

The methods of the ISearchAssistantOC2 class are listed here.

Public Methods

	Name	Description
	Get_ShowFindPrinter (see page 613)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

7.2.10.2.1 ISearchAssistantOC2.Get_ShowFindPrinter Method

Pascal

```
function Get_ShowFindPrinter: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShowFindPrinter();
```

Description

This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

7.2.10.3 ISearchAssistantOC2 Properties

The properties of the ISearchAssistantOC2 class are listed here.

Public Properties

	Name	Description
	ShowFindPrinter (see page 613)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.

7.2.10.3.1 ISearchAssistantOC2.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

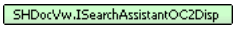
```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2.

7.2.11 ISearchAssistantOC2Disp Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC2Disp = dispinterface;
```

C++

```
class ISearchAssistantOC2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2Disp.

7.2.11.1 ISearchAssistantOC2Disp Members

The following tables list the members exposed by ISearchAssistantOC2Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 615)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 615)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 615)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 615)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 616)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 616)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 616)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 616)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 617)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.
	IsRestricted (see page 617)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 617)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 617)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 617)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 618)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 618)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 618)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 618)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

Public Properties

	Name	Description
	ASProvider (see page 619)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 619)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 619)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 620)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 620)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 620)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 620)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 620)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 621)	This is Version, a member of class ISearchAssistantOC2Disp.

7.2.11.2 ISearchAssistantOC2Disp Methods

The methods of the ISearchAssistantOC2Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 615)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 615)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 615)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 615)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 616)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 616)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 616)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 616)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 617)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.

	IsRestricted (see page 617)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 617)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 617)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 617)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 618)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 618)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 618)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 618)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

7.2.11.2.1 ISearchAssistantOC2Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.

7.2.11.2.2 ISearchAssistantOC2Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC2Disp.

7.2.11.2.3 ISearchAssistantOC2Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC2Disp.

7.2.11.2.4 ISearchAssistantOC2Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.

7.2.11.2.5 ISearchAssistantOC2Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC2Disp.

7.2.11.2.6 ISearchAssistantOC2Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC2Disp.

7.2.11.2.7 ISearchAssistantOC2Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC2Disp.

7.2.11.2.8 ISearchAssistantOC2Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC2Disp.

7.2.11.2.9 ISearchAssistantOC2Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.

7.2.11.2.10 ISearchAssistantOC2Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC2Disp.

7.2.11.2.11 ISearchAssistantOC2Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.

7.2.11.2.12 ISearchAssistantOC2Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.

7.2.11.2.13 ISearchAssistantOC2Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.

7.2.11.2.14 ISearchAssistantOC2Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC2Disp.

7.2.11.2.15 ISearchAssistantOC2Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC2Disp.

7.2.11.2.16 ISearchAssistantOC2Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.

7.2.11.2.17 ISearchAssistantOC2Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

7.2.11.3 ISearchAssistantOC2Disp Properties

The properties of the ISearchAssistantOC2Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 619)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 619)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 619)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 620)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 620)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 620)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 620)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 620)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 621)	This is Version, a member of class ISearchAssistantOC2Disp.

7.2.11.3.1 ISearchAssistantOC2Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC2Disp.

7.2.11.3.2 ISearchAssistantOC2Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC2Disp.

7.2.11.3.3 ISearchAssistantOC2Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC2Disp.

7.2.11.3.4 ISearchAssistantOC2Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC2Disp.

7.2.11.3.5 ISearchAssistantOC2Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.

7.2.11.3.6 ISearchAssistantOC2Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC2Disp.

7.2.11.3.7 ISearchAssistantOC2Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.

7.2.11.3.8 ISearchAssistantOC2Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.

7.2.11.3.9 ISearchAssistantOC2Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC2Disp.

7.2.12 ISearchAssistantOC3 Interface

Class Hierarchy

Pascal

```
ISearchAssistantOC3 = interface(ISearchAssistantOC2);
```

C++

```
__interface ISearchAssistantOC3 : public ISearchAssistantOC2;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3.

7.2.12.1 ISearchAssistantOC3 Members

The following tables list the members exposed by ISearchAssistantOC3.

Public Methods

	Name	Description
	AddNextMenuItem (see page 603)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 603)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 603)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 604)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 604)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 604)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 604)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 604)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 605)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 605)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 605)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.

	Get_SearchCompanionAvailable (see page 623)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_Searches (see page 605)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 606)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 613)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_UseSearchCompanion (see page 623)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Get_Version (see page 606)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 606)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 606)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 606)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 607)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 607)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 607)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 607)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 608)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 608)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 608)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 608)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 608)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	Set_UseSearchCompanion (see page 623)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.
	SetDefaultSearchUrl (see page 609)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 609)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 609)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 610)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 610)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 610)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	SearchCompanionAvailable (see page 623)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
R	Searches (see page 610)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 611)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 613)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
	UseSearchCompanion (see page 624)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.
R	Version (see page 611)	This is Version, a member of class ISearchAssistantOC.

7.2.12.2 ISearchAssistantOC3 Methods

The methods of the ISearchAssistantOC3 class are listed here.

Public Methods

	Name	Description
	Get_SearchCompanionAvailable (see page 623)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_UseSearchCompanion (see page 623)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Set_UseSearchCompanion (see page 623)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

7.2.12.2.1 ISearchAssistantOC3.Get_SearchCcompanionAvailable Method

Pascal

```
function Get_SearchCompanionAvailable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchCompanionAvailable();
```

Description

This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.

7.2.12.2.2 ISearchAssistantOC3.Get_UseSearchCompanion Method

Pascal

```
function Get_UseSearchCompanion: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_UseSearchCompanion();
```

Description

This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.

7.2.12.2.3 ISearchAssistantOC3.Set_UseSearchCompanion Method

Pascal

```
procedure Set_UseSearchCompanion(pbUseSC: WordBool); safecall;
```

C++

```
__fastcall Set_UseSearchCompanion(WordBool pbUseSC);
```

Description

This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

7.2.12.3 ISearchAssistantOC3 Properties

The properties of the ISearchAssistantOC3 class are listed here.

Public Properties

	Name	Description
	SearchCompanionAvailable (see page 623)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	UseSearchCompanion (see page 624)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.

7.2.12.3.1 ISearchAssistantOC3.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCcompanionAvailable, a member of class ISearchAssistantOC3.

7.2.12.3.2 ISearchAssistantOC3.UseSearchCompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

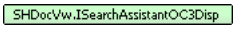
```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3.

7.2.13 ISearchAssistantOC3Disp Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC3Disp = dispinterface;
```

C++

```
class ISearchAssistantOC3Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3Disp.

7.2.13.1 ISearchAssistantOC3Disp Members

The following tables list the members exposed by ISearchAssistantOC3Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 625)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 626)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 626)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 626)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 626)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 627)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 627)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 627)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 627)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 627)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 628)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

	NETDetectNextNavigate (see page 628)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 628)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 628)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 629)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 629)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 629)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

Public Properties

	Name	Description
	ASProvider (see page 630)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 630)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 630)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 630)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 630)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 631)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 631)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 631)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 631)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 632)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.
	Version (see page 632)	This is Version, a member of class ISearchAssistantOC3Disp.

7.2.13.2 ISearchAssistantOC3Disp Methods

The methods of the ISearchAssistantOC3Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 625)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 626)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 626)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 626)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 626)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 627)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 627)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 627)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 627)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 627)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 628)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.
	NETDetectNextNavigate (see page 628)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 628)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 628)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 629)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 629)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 629)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

7.2.13.2.1 ISearchAssistantOC3Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.

7.2.13.2.2 ISearchAssistantOC3Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC3Disp.

7.2.13.2.3 ISearchAssistantOC3Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC3Disp.

7.2.13.2.4 ISearchAssistantOC3Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.

7.2.13.2.5 ISearchAssistantOC3Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC3Disp.

7.2.13.2.6 ISearchAssistantOC3Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC3Disp.

7.2.13.2.7 ISearchAssistantOC3Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC3Disp.

7.2.13.2.8 ISearchAssistantOC3Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC3Disp.

7.2.13.2.9 ISearchAssistantOC3Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.

7.2.13.2.10 ISearchAssistantOC3Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC3Disp.

7.2.13.2.11 ISearchAssistantOC3Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

7.2.13.2.12 ISearchAssistantOC3Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.

7.2.13.2.13 ISearchAssistantOC3Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.

7.2.13.2.14 ISearchAssistantOC3Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC3Disp.

7.2.13.2.15 ISearchAssistantOC3Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC3Disp.

7.2.13.2.16 ISearchAssistantOC3Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.

7.2.13.2.17 ISearchAssistantOC3Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

7.2.13.3 ISearchAssistantOC3Disp Properties

The properties of the ISearchAssistantOC3Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 630)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 630)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 630)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 630)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 630)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 631)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 631)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 631)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 631)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 632)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.

	Version (see page 632)	This is Version, a member of class ISearchAssistantOC3Disp.

7.2.13.3.1 ISearchAssistantOC3Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC3Disp.

7.2.13.3.2 ISearchAssistantOC3Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC3Disp.

7.2.13.3.3 ISearchAssistantOC3Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC3Disp.

7.2.13.3.4 ISearchAssistantOC3Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC3Disp.

7.2.13.3.5 ISearchAssistantOC3Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.

7.2.13.3.6 ISearchAssistantOC3Disp.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.

7.2.13.3.7 ISearchAssistantOC3Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC3Disp.

7.2.13.3.8 ISearchAssistantOC3Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.

7.2.13.3.9 ISearchAssistantOC3Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.

7.2.13.3.10 ISearchAssistantOC3Disp.UseSearchCcompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.

7.2.13.3.11 ISearchAssistantOC3Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

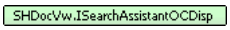
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC3Disp.

7.2.14 ISearchAssistantOCDisp Interface

Class Hierarchy



Pascal

```
ISearchAssistantOCDisp = dispinterface;
```

C++

```
class ISearchAssistantOCDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOCDisp.

7.2.14.1 ISearchAssistantOCDisp Members

The following tables list the members exposed by ISearchAssistantOCDisp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 634)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 634)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 634)	This is FindComputer, a member of class ISearchAssistantOCDisp.

	FindFilesOrFolders (see page 634)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 634)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 635)	This is FindPeople, a member of class ISearchAssistantOCDisp.
	FindPrinter (see page 635)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 635)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 635)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 636)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 636)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 636)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 636)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 636)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 637)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 637)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 637)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

Public Properties

	Name	Description
	ASProvider (see page 638)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 638)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 638)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 638)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 638)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 639)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 639)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.
	Version (see page 639)	This is Version, a member of class ISearchAssistantOCDisp.

7.2.14.2 ISearchAssistantOCDisp Methods

The methods of the ISearchAssistantOCDisp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 634)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 634)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 634)	This is FindComputer, a member of class ISearchAssistantOCDisp.
	FindFilesOrFolders (see page 634)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 634)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 635)	This is FindPeople, a member of class ISearchAssistantOCDisp.
	FindPrinter (see page 635)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 635)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 635)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 636)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 636)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 636)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 636)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 636)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 637)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 637)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 637)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

7.2.14.2.1 ISearchAssistantOCDisp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.

7.2.14.2.2 ISearchAssistantOCDisp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOCDisp.

7.2.14.2.3 ISearchAssistantOCDisp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOCDisp.

7.2.14.2.4 ISearchAssistantOCDisp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.

7.2.14.2.5 ISearchAssistantOCDisp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOCDisp.

7.2.14.2.6 ISearchAssistantOCDisp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOCDisp.

7.2.14.2.7 ISearchAssistantOCDisp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOCDisp.

7.2.14.2.8 ISearchAssistantOCDisp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOCDisp.

7.2.14.2.9 ISearchAssistantOCDisp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.

7.2.14.2.10 ISearchAssistantOCDisp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOCDisp.

7.2.14.2.11 ISearchAssistantOCDisp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.

7.2.14.2.12 ISearchAssistantOCDisp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.

7.2.14.2.13 ISearchAssistantOCDisp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.

7.2.14.2.14 ISearchAssistantOCDisp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOCDisp.

7.2.14.2.15 ISearchAssistantOCDisp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOCDisp.

7.2.14.2.16 ISearchAssistantOCDisp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOCDisp.

7.2.14.2.17 ISearchAssistantOCDisp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

7.2.14.3 ISearchAssistantOCDisp Properties

The properties of the ISearchAssistantOCDisp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 638)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 638)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 638)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 638)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 638)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 639)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 639)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.

	Version (see page 639)	This is Version, a member of class ISearchAssistantOCDisp.

7.2.14.3.1 ISearchAssistantOCDisp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOCDisp.

7.2.14.3.2 ISearchAssistantOCDisp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOCDisp.

7.2.14.3.3 ISearchAssistantOCDisp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOCDisp.

7.2.14.3.4 ISearchAssistantOCDisp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOCDisp.

7.2.14.3.5 ISearchAssistantOCDisp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.

7.2.14.3.6 ISearchAssistantOCDisp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOCDisp.

7.2.14.3.7 ISearchAssistantOCDisp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.

7.2.14.3.8 ISearchAssistantOCDisp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

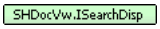
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOCDisp.

7.2.15 ISearchDisp Interface

Class Hierarchy



Pascal

```
ISearchDisp = dispinterface;
```

C++

```
class ISearchDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchDisp.

7.2.15.1 ISearchDisp Members

The following tables list the members exposed by ISearchDisp.

Public Properties

	Name	Description
	Id (see page 640)	This is Id, a member of class ISearchDisp.
	Title (see page 640)	This is Title, a member of class ISearchDisp.
	URL (see page 641)	This is URL, a member of class ISearchDisp.

7.2.15.2 ISearchDisp Properties

The properties of the ISearchDisp class are listed here.

Public Properties

	Name	Description
	Id (see page 640)	This is Id, a member of class ISearchDisp.
	Title (see page 640)	This is Title, a member of class ISearchDisp.
	URL (see page 641)	This is URL, a member of class ISearchDisp.

7.2.15.2.1 ISearchDisp.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearchDisp.

7.2.15.2.2 ISearchDisp.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearchDisp.

7.2.15.2.3 ISearchDisp.URL Property

Pascal

```
property URL: WideString;
```

C++

```
__property BSTR URL;
```

Description

This is URL, a member of class ISearchDisp.

7.2.16 ISearches Interface

Class Hierarchy

Pascal

```
ISearches = interface(IDispatch);
```

C++

```
__interface ISearches : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearches.

7.2.16.1 ISearches Members

The following tables list the members exposed by ISearches.

Public Methods

	Name	Description
	_NewEnum (see page 642)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 642)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 642)	This is Get_Default, a member of class ISearches.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Item (see page 642)	This is Item, a member of class ISearches.

Public Properties

	Name	Description
R	Count (see page 643)	This is Count, a member of class ISearches.
R	Default (see page 643)	This is Default, a member of class ISearches.

7.2.16.2 ISearches Methods

The methods of the ISearches class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 642)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 642)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 642)	This is Get_Default, a member of class ISearches.
	Item (see page 642)	This is Item, a member of class ISearches.

7.2.16.2.1 ISearches._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearches.

7.2.16.2.2 ISearches.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class ISearches.

7.2.16.2.3 ISearches.Get_Default Method

Pascal

```
function Get_Default: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Default();
```

Description

This is Get_Default, a member of class ISearches.

7.2.16.2.4 ISearches.Item Method

Pascal

```
function Item(index: OleVariant): ISearch; safecall;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearches.

7.2.16.3 ISearches Properties

The properties of the ISearches class are listed here.

Public Properties

	Name	Description
	Count (see page 643)	This is Count, a member of class ISearches.
	Default (see page 643)	This is Default, a member of class ISearches.

7.2.16.3.1 ISearches.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearches.

7.2.16.3.2 ISearches.Default Property

Pascal

```
property Default: WideString;
```

C++

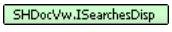
```
__property BSTR Default;
```

Description

This is Default, a member of class ISearches.

7.2.17 ISearchesDisp Interface

Class Hierarchy



Pascal

```
ISearchesDisp = dispinterface;
```

C++

```
class ISearchesDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchesDisp.

7.2.17.1 ISearchesDisp Members

The following tables list the members exposed by ISearchesDisp.

Public Methods

	Name	Description
	_NewEnum (see page 644)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 644)	This is Item, a member of class ISearchesDisp.

Public Properties

	Name	Description
	Count (see page 645)	This is Count, a member of class ISearchesDisp.
	Default (see page 645)	This is Default, a member of class ISearchesDisp.

7.2.17.2 ISearchesDisp Methods

The methods of the ISearchesDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 644)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 644)	This is Item, a member of class ISearchesDisp.

7.2.17.2.1 ISearchesDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearchesDisp.

7.2.17.2.2 ISearchesDisp.Item Method

Pascal

```
function Item(index: OleVariant): ISearch;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearchesDisp.

7.2.17.3 ISearchesDisp Properties

The properties of the ISearchesDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 645)	This is Count, a member of class ISearchesDisp.
	Default (see page 645)	This is Default, a member of class ISearchesDisp.

7.2.17.3.1 ISearchesDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearchesDisp.

7.2.17.3.2 ISearchesDisp.Default Property

Pascal

```
property Default: WideString;
```

C++

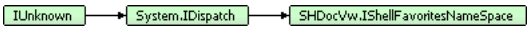
```
__property BSTR Default;
```

Description

This is Default, a member of class ISearchesDisp.

7.2.18 IShellFavoritesNameSpace Interface

Class Hierarchy



Pascal

```
IShellFavoritesNameSpace = interface(IDispatch);
```

C++

```
__interface IShellFavoritesNameSpace : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpace.

7.2.18.1 IShellFavoritesNameSpace Members

The following tables list the members exposed by IShellFavoritesNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 647)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 647)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 647)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 647)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Import (see page 647)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	InvokeContextMenuCommand (see page 648)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 648)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 648)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 648)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 649)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 649)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 649)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	Synchronize (see page 649)	This is Synchronize, a member of class IShellFavoritesNameSpace.

Public Properties

	Name	Description
R	SubscriptionsEnabled (see page 650)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

7.2.18.2 IShellFavoritesNameSpace Methods

The methods of the IShellFavoritesNameSpace class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 647)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 647)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 647)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 647)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	Import (see page 647)	This is Import, a member of class IShellFavoritesNameSpace.
	InvokeContextMenuCommand (see page 648)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 648)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 648)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 648)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 649)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 649)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 649)	This is SetRoot, a member of class IShellFavoritesNameSpace.

	Synchronize (see page 649)	This is Synchronize, a member of class IShellFavoritesNameSpace.

7.2.18.2.1 IShellFavoritesNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

7.2.18.2.2 IShellFavoritesNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

7.2.18.2.3 IShellFavoritesNameSpace.Export Method

Pascal

```
procedure Export; safecall;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpace.

7.2.18.2.4 IShellFavoritesNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

7.2.18.2.5 IShellFavoritesNameSpace.Import Method

Pascal

```
procedure Import; safecall;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpace.

7.2.18.2.6 IShellFavoritesNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString); safecall;
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.

7.2.18.2.7 IShellFavoritesNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown; safecall;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.

7.2.18.2.8 IShellFavoritesNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo; safecall;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.

7.2.18.2.9 IShellFavoritesNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp; safecall;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.

7.2.18.2.10 IShellFavoritesNameSpace.NewFolder Method

Pascal

```
procedure NewFolder; safecall;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpace.

7.2.18.2.11 IShellFavoritesNameSpace.ResetSort Method

Pascal

```
procedure ResetSort; safecall;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpace.

7.2.18.2.12 IShellFavoritesNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString); safecall;
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpace.

7.2.18.2.13 IShellFavoritesNameSpace.Synchronize Method

Pascal

```
procedure Synchronize; safecall;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpace.

7.2.18.3 IShellFavoritesNameSpace Properties

The properties of the IShellFavoritesNameSpace class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 650)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

7.2.18.3.1 IShellFavoritesNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

7.2.19 IShellFavoritesNameSpaceDisp Interface

Class Hierarchy

Pascal

```
IShellFavoritesNameSpaceDisp = dispinterface;
```

C++

```
class IShellFavoritesNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpaceDisp.

7.2.19.1 IShellFavoritesNameSpaceDisp Members

The following tables list the members exposed by IShellFavoritesNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 651)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 651)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 652)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 652)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 652)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 652)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 653)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 653)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 653)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.

	ResetSort (see page 653)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 653)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 654)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 654)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2 IShellFavoritesNameSpaceDisp Methods

The methods of the IShellFavoritesNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 651)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 651)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 652)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 652)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 652)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 652)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 653)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 653)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 653)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.
	ResetSort (see page 653)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 653)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 654)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.1 IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.2 IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.3 IShellFavoritesNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.4 IShellFavoritesNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.5 IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.6 IShellFavoritesNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.7 IShellFavoritesNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.8 IShellFavoritesNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.9 IShellFavoritesNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.10 IShellFavoritesNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.11 IShellFavoritesNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.2.12 IShellFavoritesNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.3 IShellFavoritesNameSpaceDisp Properties

The properties of the IShellFavoritesNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 654)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

7.2.19.3.1 IShellFavoritesNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

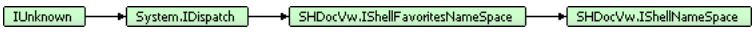
```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

7.2.20 IShellNameSpace Interface

Class Hierarchy



Pascal

```
IShellNameSpace = interface(IShellFavoritesNameSpace);
```

C++

```
__interface IShellNameSpace : public IShellFavoritesNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpace.

7.2.20.1 IShellNameSpace Members

The following tables list the members exposed by IShellNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 647)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 647)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Expand (see page 656)	This is Expand, a member of class IShellNameSpace.
	Export (see page 647)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_Columns (see page 657)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 657)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 657)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 657)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 657)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 658)	This is Get_Mode, a member of class IShellNameSpace.
	Get_Root (see page 658)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 658)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_SubscriptionsEnabled (see page 647)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	Get_TVFlags (see page 658)	This is Get_TVFlags, a member of class IShellNameSpace.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Import (see page 647)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	InvokeContextMenuCommand (see page 648)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 648)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 648)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 648)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 649)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 649)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SelectedItems (see page 659)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 659)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 659)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 659)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 659)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 660)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 660)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 660)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 660)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetRoot (see page 649)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	SetViewType (see page 661)	This is SetViewType, a member of class IShellNameSpace.
	Synchronize (see page 649)	This is Synchronize, a member of class IShellFavoritesNameSpace.
	UnselectAll (see page 661)	This is UnselectAll, a member of class IShellNameSpace.

Public Properties

	Name	Description
	Columns (see page 661)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 662)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 662)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 662)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 662)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 662)	This is Mode, a member of class IShellNameSpace.
	Root (see page 663)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 663)	This is SelectedItem, a member of class IShellNameSpace.
R	SubscriptionsEnabled (see page 650)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	TVFlags (see page 663)	This is TVFlags, a member of class IShellNameSpace.

7.2.20.2 IShellNameSpace Methods

The methods of the IShellNameSpace class are listed here.

Public Methods

	Name	Description
	Expand (see page 656)	This is Expand, a member of class IShellNameSpace.
	Get_Columns (see page 657)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 657)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 657)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 657)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 657)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 658)	This is Get_Mode, a member of class IShellNameSpace.
	Get_Root (see page 658)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 658)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_TVFlags (see page 658)	This is Get_TVFlags, a member of class IShellNameSpace.
	SelectedItems (see page 659)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 659)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 659)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 659)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 659)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 660)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 660)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 660)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 660)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetViewType (see page 661)	This is SetViewType, a member of class IShellNameSpace.
	UnselectAll (see page 661)	This is UnselectAll, a member of class IShellNameSpace.

7.2.20.2.1 IShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT); safecall;
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpace.

7.2.20.2.2 IShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class IShellNameSpace.

7.2.20.2.3 IShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class IShellNameSpace.

7.2.20.2.4 IShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class IShellNameSpace.

7.2.20.2.5 IShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer; safecall;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class IShellNameSpace.

7.2.20.2.6 IShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class IShellNameSpace.

7.2.20.2.7 IShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT; safecall;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class IShellNameSpace.

7.2.20.2.8 IShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class IShellNameSpace.

7.2.20.2.9 IShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class IShellNameSpace.

7.2.20.2.10 IShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class IShellNameSpace.

7.2.20.2.11 IShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch; safecall;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpace.

7.2.20.2.12 IShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString); safecall;
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class IShellNameSpace.

7.2.20.2.13 IShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT); safecall;
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class IShellNameSpace.

7.2.20.2.14 IShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer); safecall;
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class IShellNameSpace.

7.2.20.2.15 IShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class IShellNameSpace.

7.2.20.2.16 IShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT); safecall;
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class IShellNameSpace.

7.2.20.2.17 IShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant); safecall;
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class IShellNameSpace.

7.2.20.2.18 IShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch); safecall;
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class IShellNameSpace.

7.2.20.2.19 IShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class IShellNameSpace.

7.2.20.2.20 IShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT); safecall;
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpace.

7.2.20.2.21 IShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll; safecall;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpace.

7.2.20.3 IShellNameSpace Properties

The properties of the IShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 661)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 662)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 662)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 662)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 662)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 662)	This is Mode, a member of class IShellNameSpace.
	Root (see page 663)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 663)	This is SelectedItem, a member of class IShellNameSpace.
	TVFlags (see page 663)	This is TVFlags, a member of class IShellNameSpace.

7.2.20.3.1 IShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpace.

7.2.20.3.2 IShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpace.

7.2.20.3.3 IShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpace.

7.2.20.3.4 IShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpace.

7.2.20.3.5 IShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpace.

7.2.20.3.6 IShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpace.

7.2.20.3.7 IShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpace.

7.2.20.3.8 IShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpace.

7.2.20.3.9 IShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

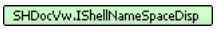
```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpace.

7.2.21 IShellNameSpaceDisp Interface

Class Hierarchy



Pascal

```
IShellNameSpaceDisp = dispinterface;
```

C++

```
class IShellNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpaceDisp.

7.2.21.1 IShellNameSpaceDisp Members

The following tables list the members exposed by IShellNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 665)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 665)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 665)	This is Expand, a member of class IShellNameSpaceDisp.
	Export (see page 665)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 666)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 666)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 666)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 666)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 667)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 667)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 667)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 667)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 667)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 668)	This is SetViewType, a member of class IShellNameSpaceDisp.
	Synchronize (see page 668)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 668)	This is UnselectAll, a member of class IShellNameSpaceDisp.

Public Properties

	Name	Description
	Columns (see page 669)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 669)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 669)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 669)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 669)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 670)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 670)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 670)	This is SelectedItem, a member of class IShellNameSpaceDisp.
	SubscriptionsEnabled (see page 670)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 671)	This is TVFlags, a member of class IShellNameSpaceDisp.

7.2.21.2 IShellNameSpaceDisp Methods

The methods of the IShellNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 665)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 665)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 665)	This is Expand, a member of class IShellNameSpaceDisp.

	Export (see page 665)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 666)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 666)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 666)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 666)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 667)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 667)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 667)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 667)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 667)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 668)	This is SetViewType, a member of class IShellNameSpaceDisp.
	Synchronize (see page 668)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 668)	This is UnselectAll, a member of class IShellNameSpaceDisp.

7.2.21.2.1 IShellNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.

7.2.21.2.2 IShellNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.

7.2.21.2.3 IShellNameSpaceDisp.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpaceDisp.

7.2.21.2.4 IShellNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellNameSpaceDisp.

7.2.21.2.5 IShellNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellNameSpaceDisp.

7.2.21.2.6 IShellNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.

7.2.21.2.7 IShellNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellNameSpaceDisp.

7.2.21.2.8 IShellNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellNameSpaceDisp.

7.2.21.2.9 IShellNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellNameSpaceDisp.

7.2.21.2.10 IShellNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellNameSpaceDisp.

7.2.21.2.11 IShellNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellNameSpaceDisp.

7.2.21.2.12 IShellNameSpaceDisp.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpaceDisp.

7.2.21.2.13 IShellNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellNameSpaceDisp.

7.2.21.2.14 IShellNameSpaceDisp.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpaceDisp.

7.2.21.2.15 IShellNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellNameSpaceDisp.

7.2.21.2.16 IShellNameSpaceDisp.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpaceDisp.

7.2.21.3 IShellNameSpaceDisp Properties

The properties of the IShellNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	Columns (see page 669)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 669)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 669)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 669)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 669)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 670)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 670)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 670)	This is SelectedItem, a member of class IShellNameSpaceDisp.

	SubscriptionsEnabled (see page 670)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 671)	This is TVFlags, a member of class IShellNameSpaceDisp.

7.2.21.3.1 IShellNameSpaceDisp.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpaceDisp.

7.2.21.3.2 IShellNameSpaceDisp.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpaceDisp.

7.2.21.3.3 IShellNameSpaceDisp.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpaceDisp.

7.2.21.3.4 IShellNameSpaceDisp.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpaceDisp.

7.2.21.3.5 IShellNameSpaceDisp.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpaceDisp.

7.2.21.3.6 IShellNameSpaceDisp.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpaceDisp.

7.2.21.3.7 IShellNameSpaceDisp.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpaceDisp.

7.2.21.3.8 IShellNameSpaceDisp.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpaceDisp.

7.2.21.3.9 IShellNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.

7.2.21.3.10 IShellNameSpaceDisp.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

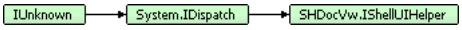
```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpaceDisp.

7.2.22 IShellUIHelper Interface

Class Hierarchy



Pascal

```
IShellUIHelper = interface(IDispatch);
```

C++

```
__interface IShellUIHelper : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelper.

7.2.22.1 IShellUIHelper Members

The following tables list the members exposed by IShellUIHelper.

Public Methods

	Name	Description
	AddChannel (see page 672)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 672)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 672)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 673)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 673)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 673)	This is AutoScan, a member of class IShellUIHelper.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	ImportExportFavorites (see page 673)	This is ImportExportFavorites, a member of class IShellUIHelper.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	IsSubscribed (see page 674)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 674)	This is NavigateAndFind, a member of class IShellUIHelper.
	RefreshOfflineDesktop (see page 674)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 674)	This is ResetFirstBootMode, a member of class IShellUIHelper.

	ResetSafeMode (see page 674)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 675)	This is ShowBrowserUI, a member of class IShellUIHelper.

7.2.22.2 IShellUIHelper Methods

The methods of the IShellUIHelper class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 672)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 672)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 672)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 673)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 673)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 673)	This is AutoScan, a member of class IShellUIHelper.
	ImportExportFavorites (see page 673)	This is ImportExportFavorites, a member of class IShellUIHelper.
	IsSubscribed (see page 674)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 674)	This is NavigateAndFind, a member of class IShellUIHelper.
	RefreshOfflineDesktop (see page 674)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 674)	This is ResetFirstBootMode, a member of class IShellUIHelper.
	ResetSafeMode (see page 674)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 675)	This is ShowBrowserUI, a member of class IShellUIHelper.

7.2.22.2.1 IShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString); safecall;
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelper.

7.2.22.2.2 IShellUIHelper.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); safecall;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelper.

7.2.22.2.3 IShellUIHelper.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); safecall;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelper.

7.2.22.2.4 IShellUIHelper.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelper.

7.2.22.2.5 IShellUIHelper.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelper.

7.2.22.2.6 IShellUIHelper.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelper.

7.2.22.2.7 IShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString); safecall;
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelper.

7.2.22.2.8 IShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelper.

7.2.22.2.9 IShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelper.

7.2.22.2.10 IShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop; safecall;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelper.

7.2.22.2.11 IShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode; safecall;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelper.

7.2.22.2.12 IShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode; safecall;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelper.

7.2.22.2.13 IShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelper.

7.2.23 IShellUIHelperDisp Interface

Class Hierarchy

SHDocVw.IShellUIHelperDisp

Pascal

```
IShellUIHelperDisp = dispinterface;
```

C++

```
class IShellUIHelperDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelperDisp.

7.2.23.1 IShellUIHelperDisp Members

The following tables list the members exposed by IShellUIHelperDisp.

Public Methods

	Name	Description
	AddChannel (see page 676)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 676)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 677)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 677)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 677)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.
	AutoScan (see page 677)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 677)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 678)	This is IsSubscribed, a member of class IShellUIHelperDisp.

	NavigateAndFind (see page 678)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 678)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 678)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 679)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 679)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

7.2.23.2 IShellUIHelperDisp Methods

The methods of the IShellUIHelperDisp class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 676)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 676)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 677)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 677)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 677)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.
	AutoScan (see page 677)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 677)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 678)	This is IsSubscribed, a member of class IShellUIHelperDisp.
	NavigateAndFind (see page 678)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 678)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 678)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 679)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 679)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

7.2.23.2.1 IShellUIHelperDisp.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelperDisp.

7.2.23.2.2 IShellUIHelperDisp.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant);
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelperDisp.

7.2.23.2.3 IShellUIHelperDisp.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant);
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelperDisp.

7.2.23.2.4 IShellUIHelperDisp.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant);
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelperDisp.

7.2.23.2.5 IShellUIHelperDisp.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant);
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.

7.2.23.2.6 IShellUIHelperDisp.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant);
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelperDisp.

7.2.23.2.7 IShellUIHelperDisp.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelperDisp.

7.2.23.2.8 IShellUIHelperDisp.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelperDisp.

7.2.23.2.9 IShellUIHelperDisp.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelperDisp.

7.2.23.2.10 IShellUIHelperDisp.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.

7.2.23.2.11 IShellUIHelperDisp.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelperDisp.

7.2.23.2.12 IShellUIHelperDisp.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelperDisp.

7.2.23.2.13 IShellUIHelperDisp.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelperDisp.

7.2.24 IShellWindows Interface

Class Hierarchy

Pascal

```
IShellWindows = interface(IDispatch);
```

C++

```
__interface IShellWindows : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindows.

7.2.24.1 IShellWindows Members

The following tables list the members exposed by IShellWindows.

Public Methods

	Name	Description
	_NewEnum (see page 680)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 680)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 681)	This is Get_Count, a member of class IShellWindows.

	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Item (see page 681)	This is Item, a member of class IShellWindows.
	OnActivated (see page 681)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 681)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 682)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 682)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 682)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 682)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 682)	This is Revoke, a member of class IShellWindows.

Public Properties

	Name	Description
R	Count (see page 683)	This is Count, a member of class IShellWindows.

7.2.24.2 IShellWindows Methods

The methods of the IShellWindows class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 680)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 680)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 681)	This is Get_Count, a member of class IShellWindows.
	Item (see page 681)	This is Item, a member of class IShellWindows.
	OnActivated (see page 681)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 681)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 682)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 682)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 682)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 682)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 682)	This is Revoke, a member of class IShellWindows.

7.2.24.2.1 IShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindows.

7.2.24.2.2 IShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch; safecall;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindows.

7.2.24.2.3 IShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class IShellWindows.

7.2.24.2.4 IShellWindows.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch; safecall;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindows.

7.2.24.2.5 IShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool); safecall;
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindows.

7.2.24.2.6 IShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown); safecall;
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindows.

7.2.24.2.7 IShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant); safecall;
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindows.

7.2.24.2.8 IShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool); safecall;
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindows.

7.2.24.2.9 IShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer); safecall;
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindows.

7.2.24.2.10 IShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer); safecall;
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindows.

7.2.24.2.11 IShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer); safecall;
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindows.

7.2.24.3 IShellWindows Properties

The properties of the IShellWindows class are listed here.

Public Properties

	Name	Description
	Count (see page 683)	This is Count, a member of class IShellWindows.

7.2.24.3.1 IShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IShellWindows.

7.2.25 IShellWindowsDisp Interface

Class Hierarchy

SHDocVw.IShellWindowsDisp

Pascal

```
IShellWindowsDisp = dispinterface;
```

C++

```
class IShellWindowsDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindowsDisp.

7.2.25.1 IShellWindowsDisp Members

The following tables list the members exposed by IShellWindowsDisp.

Public Methods

	Name	Description
	_NewEnum (see page 684)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 684)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 685)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 685)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 685)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 685)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 686)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 686)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 686)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 686)	This is Revoke, a member of class IShellWindowsDisp.

Public Properties

	Name	Description
	Count (see page 687)	This is Count, a member of class IShellWindowsDisp.

7.2.25.2 IShellWindowsDisp Methods

The methods of the IShellWindowsDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 684)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 684)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 685)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 685)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 685)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 685)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 686)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 686)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 686)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 686)	This is Revoke, a member of class IShellWindowsDisp.

7.2.25.2.1 IShellWindowsDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindowsDisp.

7.2.25.2.2 IShellWindowsDisp.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindowsDisp.

7.2.25.2.3 IShellWindowsDisp.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindowsDisp.

7.2.25.2.4 IShellWindowsDisp.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindowsDisp.

7.2.25.2.5 IShellWindowsDisp.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindowsDisp.

7.2.25.2.6 IShellWindowsDisp.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindowsDisp.

7.2.25.2.7 IShellWindowsDisp.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindowsDisp.

7.2.25.2.8 IShellWindowsDisp.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindowsDisp.

7.2.25.2.9 IShellWindowsDisp.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindowsDisp.

7.2.25.2.10 IShellWindowsDisp.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindowsDisp.

7.2.25.3 IShellWindowsDisp Properties

The properties of the IShellWindowsDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 687)	This is Count, a member of class IShellWindowsDisp.

7.2.25.3.1 IShellWindowsDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IShellWindowsDisp.

7.2.26 IWebBrowser Interface

Class Hierarchy

Pascal

```
IWebBrowser = interface(IDispatch);
```

C++

```
__interface IWebBrowser : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser.

7.2.26.1 IWebBrowser Members

The following tables list the members exposed by IWebBrowser.

Public Methods

	Name	Description
	Get_Application (see page 689)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 689)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 689)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 689)	This is Get_Document, a member of class IWebBrowser.
	Get_Height (see page 690)	This is Get_Height, a member of class IWebBrowser.
	Get_Left (see page 690)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 690)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 690)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_Parent (see page 691)	This is Get_Parent, a member of class IWebBrowser.
	Get_Top (see page 691)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 691)	This is Get_TopLevelContainer, a member of class IWebBrowser.

	Get_type_ (see page 691)	This is Get_type_, a member of class IWebBrowser.
	Get_Width (see page 691)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 692)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 692)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 692)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 692)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Navigate (see page 693)	This is Navigate, a member of class IWebBrowser.
	Refresh (see page 693)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 693)	This is Refresh2, a member of class IWebBrowser.
	Set_Height (see page 693)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 693)	This is Set_Left, a member of class IWebBrowser.
	Set_Top (see page 694)	This is Set_Top, a member of class IWebBrowser.
	Set_Width (see page 694)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 694)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
R	Application (see page 695)	This is Application, a member of class IWebBrowser.
R	Busy (see page 695)	This is Busy, a member of class IWebBrowser.
R	Container (see page 695)	This is Container, a member of class IWebBrowser.
R	Document (see page 695)	This is Document, a member of class IWebBrowser.
	Height (see page 696)	This is Height, a member of class IWebBrowser.
	Left (see page 696)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 696)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 696)	This is LocationURL, a member of class IWebBrowser.
R	Parent (see page 696)	This is Parent, a member of class IWebBrowser.
	Top (see page 697)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 697)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 697)	This is type_, a member of class IWebBrowser.
	Width (see page 697)	This is Width, a member of class IWebBrowser.

7.2.26.2 IWebBrowser Methods

The methods of the IWebBrowser class are listed here.

Public Methods

	Name	Description
	Get_Application (see page 689)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 689)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 689)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 689)	This is Get_Document, a member of class IWebBrowser.
	Get_Height (see page 690)	This is Get_Height, a member of class IWebBrowser.
	Get_Left (see page 690)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 690)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 690)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_Parent (see page 691)	This is Get_Parent, a member of class IWebBrowser.
	Get_Top (see page 691)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 691)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 691)	This is Get_type_, a member of class IWebBrowser.

	Get_Width (see page 691)	This is Get_Width, a member of class IWebBrowser.
	GoBack (see page 692)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 692)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 692)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 692)	This is GoSearch, a member of class IWebBrowser.
	Navigate (see page 693)	This is Navigate, a member of class IWebBrowser.
	Refresh (see page 693)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 693)	This is Refresh2, a member of class IWebBrowser.
	Set_Height (see page 693)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 693)	This is Set_Left, a member of class IWebBrowser.
	Set_Top (see page 694)	This is Set_Top, a member of class IWebBrowser.
	Set_Width (see page 694)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 694)	This is Stop, a member of class IWebBrowser.

7.2.26.2.1 IWebBrowser.Get_Application Method

Pascal

```
function Get_Application: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class IWebBrowser.

7.2.26.2.2 IWebBrowser.Get_Busy Method

Pascal

```
function Get_Busy: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

This is Get_Busy, a member of class IWebBrowser.

7.2.26.2.3 IWebBrowser.Get_Container Method

Pascal

```
function Get_Container: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class IWebBrowser.

7.2.26.2.4 IWebBrowser.Get_Document Method

Pascal

```
function Get_Document: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class IWebBrowser.

7.2.26.2.5 IWebBrowser.Get_Height Method

Pascal

```
function Get_Height: Integer; safecall;
```

C++

```
__fastcall int Get_Height();
```

Description

This is Get_Height, a member of class IWebBrowser.

7.2.26.2.6 IWebBrowser.Get_Left Method

Pascal

```
function Get_Left: Integer; safecall;
```

C++

```
__fastcall int Get_Left();
```

Description

This is Get_Left, a member of class IWebBrowser.

7.2.26.2.7 IWebBrowser.Get_LocationName Method

Pascal

```
function Get_LocationName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

This is Get_LocationName, a member of class IWebBrowser.

7.2.26.2.8 IWebBrowser.Get_LocationURL Method

Pascal

```
function Get_LocationURL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

This is Get_LocationURL, a member of class IWebBrowser.

7.2.26.2.9 IWebBrowser.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class IWebBrowser.

7.2.26.2.10 IWebBrowser.Get_Top Method

Pascal

```
function Get_Top: Integer; safecall;
```

C++

```
__fastcall int Get_Top();
```

Description

This is Get_Top, a member of class IWebBrowser.

7.2.26.2.11 IWebBrowser.Get_TopLevelContainer Method

Pascal

```
function Get_TopLevelContainer: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

This is Get_TopLevelContainer, a member of class IWebBrowser.

7.2.26.2.12 IWebBrowser.Get_type_ Method

Pascal

```
function Get_type_: WideString; safecall;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

This is Get_type_, a member of class IWebBrowser.

7.2.26.2.13 IWebBrowser.Get_Width Method

Pascal

```
function Get_Width: Integer; safecall;
```

C++

```
__fastcall int Get_Width();
```

Description

This is Get_Width, a member of class IWebBrowser.

7.2.26.2.14 IWebBrowser.GoBack Method

Pascal

```
procedure GoBack; safecall;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowser.

7.2.26.2.15 IWebBrowser.GoForward Method

Pascal

```
procedure GoForward; safecall;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowser.

7.2.26.2.16 IWebBrowser.GoHome Method

Pascal

```
procedure GoHome; safecall;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowser.

7.2.26.2.17 IWebBrowser.GoSearch Method

Pascal

```
procedure GoSearch; safecall;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowser.

7.2.26.2.18 IWebBrowser.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowser.

7.2.26.2.19 IWebBrowser.Refresh Method

Pascal

```
procedure Refresh; safecall;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowser.

7.2.26.2.20 IWebBrowser.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant); safecall;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowser.

7.2.26.2.21 IWebBrowser.Set_Height Method

Pascal

```
procedure Set_Height(pl: Integer); safecall;
```

C++

```
__fastcall Set_Height(int pl);
```

Description

This is Set_Height, a member of class IWebBrowser.

7.2.26.2.22 IWebBrowser.Set_Left Method

Pascal

```
procedure Set_Left(pl: Integer); safecall;
```

C++

```
__fastcall Set_Left(int pl);
```

Description

This is Set_Left, a member of class IWebBrowser.

7.2.26.2.23 IWebBrowser.Set_Top Method

Pascal

```
procedure Set_Top(pl: Integer); safecall;
```

C++

```
__fastcall Set_Top(int pl);
```

Description

This is Set_Top, a member of class IWebBrowser.

7.2.26.2.24 IWebBrowser.Set_Width Method

Pascal

```
procedure Set_Width(pl: Integer); safecall;
```

C++

```
__fastcall Set_Width(int pl);
```

Description

This is Set_Width, a member of class IWebBrowser.

7.2.26.2.25 IWebBrowser.Stop Method

Pascal

```
procedure Stop; safecall;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowser.

7.2.26.3 IWebBrowser Properties

The properties of the IWebBrowser class are listed here.

Public Properties

	Name	Description
R	Application (see page 695)	This is Application, a member of class IWebBrowser.
R	Busy (see page 695)	This is Busy, a member of class IWebBrowser.
R	Container (see page 695)	This is Container, a member of class IWebBrowser.
R	Document (see page 695)	This is Document, a member of class IWebBrowser.
	Height (see page 696)	This is Height, a member of class IWebBrowser.

	Left (see page 696)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 696)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 696)	This is LocationURL, a member of class IWebBrowser.
R	Parent (see page 696)	This is Parent, a member of class IWebBrowser.
	Top (see page 697)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 697)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 697)	This is type_, a member of class IWebBrowser.
	Width (see page 697)	This is Width, a member of class IWebBrowser.

7.2.26.3.1 IWebBrowser.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowser.

7.2.26.3.2 IWebBrowser.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowser.

7.2.26.3.3 IWebBrowser.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowser.

7.2.26.3.4 IWebBrowser.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowser.

7.2.26.3.5 IWebBrowser.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowser.

7.2.26.3.6 IWebBrowser.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowser.

7.2.26.3.7 IWebBrowser.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowser.

7.2.26.3.8 IWebBrowser.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowser.

7.2.26.3.9 IWebBrowser.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowser.

7.2.26.3.10 IWebBrowser.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowser.

7.2.26.3.11 IWebBrowser.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowser.

7.2.26.3.12 IWebBrowser.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowser.

7.2.26.3.13 IWebBrowser.Width Property

Pascal

```
property Width: Integer;
```

C++

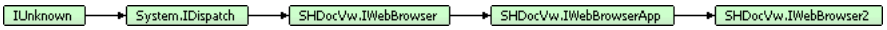
```
__property int Width;
```

Description

This is Width, a member of class IWebBrowser.

7.2.27 IWebBrowser2 Interface

Class Hierarchy



Pascal

```
IWebBrowser2 = interface(IWebBrowserApp);
```

C++

```
__interface IWebBrowser2 : public IWebBrowserApp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser2.

7.2.27.1 IWebBrowser2 Members

The following tables list the members exposed by IWebBrowser2.

Public Methods

	Name	Description
	ClientToWindow (see page 722)	This is ClientToWindow, a member of class IWebBrowserApp.
	ExecWB (see page 700)	This is ExecWB, a member of class IWebBrowser2.
	Get_AddressBar (see page 701)	This is Get_AddressBar, a member of class IWebBrowser2.
	Get_Application (see page 689)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 689)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 689)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 689)	This is Get_Document, a member of class IWebBrowser.
	Get_FullName (see page 722)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 723)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 690)	This is Get_Height, a member of class IWebBrowser.
	Get_HWnd (see page 723)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 690)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 690)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 690)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_MenuBar (see page 723)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 723)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Offline (see page 701)	This is Get_Offline, a member of class IWebBrowser2.
	Get_Parent (see page 691)	This is Get_Parent, a member of class IWebBrowser.
	Get_Path (see page 724)	This is Get_Path, a member of class IWebBrowserApp.
	Get_ReadyState (see page 701)	This is Get_ReadyState, a member of class IWebBrowser2.
	Get_RegisterAsBrowser (see page 701)	This is Get_RegisterAsBrowser, a member of class IWebBrowser2.
	Get_RegisterAsDropTarget (see page 702)	This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.
	Get_Resizable (see page 702)	This is Get_Resizable, a member of class IWebBrowser2.
	Get_Silent (see page 702)	This is Get_Silent, a member of class IWebBrowser2.
	Get_StatusBar (see page 724)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 724)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_TheaterMode (see page 702)	This is Get_TheaterMode, a member of class IWebBrowser2.

	Get_ToolBar (see page 724)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 691)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 691)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 691)	This is Get_type_, a member of class IWebBrowser.
	Get_Visible (see page 724)	This is Get_Visible, a member of class IWebBrowserApp.
	Get_Width (see page 691)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 725)	This is GetProperty, a member of class IWebBrowserApp.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 692)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 692)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 692)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 692)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Navigate (see page 693)	This is Navigate, a member of class IWebBrowser.
	Navigate2 (see page 702)	This is Navigate2, a member of class IWebBrowser2.
	PutProperty (see page 725)	This is PutProperty, a member of class IWebBrowserApp.
	QueryStatusWB (see page 703)	This is QueryStatusWB, a member of class IWebBrowser2.
	Quit (see page 725)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 693)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 693)	This is Refresh2, a member of class IWebBrowser.
	Set_AddressBar (see page 703)	This is Set_AddressBar, a member of class IWebBrowser2.
	Set_FullScreen (see page 725)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 693)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 693)	This is Set_Left, a member of class IWebBrowser.
	Set_MenuBar (see page 726)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_Offline (see page 703)	This is Set_Offline, a member of class IWebBrowser2.
	Set_RegisterAsBrowser (see page 703)	This is Set_RegisterAsBrowser, a member of class IWebBrowser2.
	Set_RegisterAsDropTarget (see page 704)	This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.
	Set_Resizable (see page 704)	This is Set_Resizable, a member of class IWebBrowser2.
	Set_Silent (see page 704)	This is Set_Silent, a member of class IWebBrowser2.
	Set_StatusBar (see page 726)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 726)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_TheaterMode (see page 704)	This is Set_TheaterMode, a member of class IWebBrowser2.
	Set_ToolBar (see page 726)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 694)	This is Set_Top, a member of class IWebBrowser.
	Set_Visible (see page 726)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 694)	This is Set_Width, a member of class IWebBrowser.
	ShowBrowserBar (see page 704)	This is ShowBrowserBar, a member of class IWebBrowser2.
	Stop (see page 694)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
	AddressBar (see page 705)	This is AddressBar, a member of class IWebBrowser2.
R	Application (see page 695)	This is Application, a member of class IWebBrowser.
R	Busy (see page 695)	This is Busy, a member of class IWebBrowser.
R	Container (see page 695)	This is Container, a member of class IWebBrowser.
R	Document (see page 695)	This is Document, a member of class IWebBrowser.
R	FullName (see page 727)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 727)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 696)	This is Height, a member of class IWebBrowser.
R	HWnd (see page 727)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 696)	This is Left, a member of class IWebBrowser.

R	LocationName (see page 696)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 696)	This is LocationURL, a member of class IWebBrowser.
	MenuBar (see page 728)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 728)	This is Name, a member of class IWebBrowserApp.
	Offline (see page 705)	This is Offline, a member of class IWebBrowser2.
R	Parent (see page 696)	This is Parent, a member of class IWebBrowser.
R	Path (see page 728)	This is Path, a member of class IWebBrowserApp.
R	ReadyState (see page 705)	This is ReadyState, a member of class IWebBrowser2.
	RegisterAsBrowser (see page 706)	This is RegisterAsBrowser, a member of class IWebBrowser2.
	RegisterAsDropTarget (see page 706)	This is RegisterAsDropTarget, a member of class IWebBrowser2.
	Resizable (see page 706)	This is Resizable, a member of class IWebBrowser2.
	Silent (see page 706)	This is Silent, a member of class IWebBrowser2.
	StatusBar (see page 728)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 729)	This is StatusText, a member of class IWebBrowserApp.
	TheaterMode (see page 707)	This is TheaterMode, a member of class IWebBrowser2.
	ToolBar (see page 729)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 697)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 697)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 697)	This is type_, a member of class IWebBrowser.
	Visible (see page 729)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 697)	This is Width, a member of class IWebBrowser.

7.2.27.2 IWebBrowser2 Methods

The methods of the IWebBrowser2 class are listed here.

Public Methods

	Name	Description
	ExecWB (see page 700)	This is ExecWB, a member of class IWebBrowser2.
	Get_AddressBar (see page 701)	This is Get_AddressBar, a member of class IWebBrowser2.
	Get_Offline (see page 701)	This is Get_Offline, a member of class IWebBrowser2.
	Get_ReadyState (see page 701)	This is Get_ReadyState, a member of class IWebBrowser2.
	Get_RegisterAsBrowser (see page 701)	This is Get_RegisterAsBrowser, a member of class IWebBrowser2.
	Get_RegisterAsDropTarget (see page 702)	This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.
	Get_Resizable (see page 702)	This is Get_Resizable, a member of class IWebBrowser2.
	Get_Silent (see page 702)	This is Get_Silent, a member of class IWebBrowser2.
	Get_TheaterMode (see page 702)	This is Get_TheaterMode, a member of class IWebBrowser2.
	Navigate2 (see page 702)	This is Navigate2, a member of class IWebBrowser2.
	QueryStatusWB (see page 703)	This is QueryStatusWB, a member of class IWebBrowser2.
	Set_AddressBar (see page 703)	This is Set_AddressBar, a member of class IWebBrowser2.
	Set_Offline (see page 703)	This is Set_Offline, a member of class IWebBrowser2.
	Set_RegisterAsBrowser (see page 703)	This is Set_RegisterAsBrowser, a member of class IWebBrowser2.
	Set_RegisterAsDropTarget (see page 704)	This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.
	Set_Resizable (see page 704)	This is Set_Resizable, a member of class IWebBrowser2.
	Set_Silent (see page 704)	This is Set_Silent, a member of class IWebBrowser2.
	Set_TheaterMode (see page 704)	This is Set_TheaterMode, a member of class IWebBrowser2.
	ShowBrowserBar (see page 704)	This is ShowBrowserBar, a member of class IWebBrowser2.

7.2.27.2.1 IWebBrowser2.ExecWB Method

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); safecall;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class IWebBrowser2.

7.2.27.2.2 IWebBrowser2.Get_AddressBar Method

Pascal

```
function Get_AddressBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

This is Get_AddressBar, a member of class IWebBrowser2.

7.2.27.2.3 IWebBrowser2.Get_Offline Method

Pascal

```
function Get_Offline: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

This is Get_Offline, a member of class IWebBrowser2.

7.2.27.2.4 IWebBrowser2.Get_ReadyState Method

Pascal

```
function Get_ReadyState: tagREADYSTATE; safecall;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

This is Get_ReadyState, a member of class IWebBrowser2.

7.2.27.2.5 IWebBrowser2.Get_RegisterAsBrowser Method

Pascal

```
function Get_RegisterAsBrowser: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

This is Get_RegisterAsBrowser, a member of class IWebBrowser2.

7.2.27.2.6 IWebBrowser2.Get_RegisterAsDropTarget Method

Pascal

```
function Get_RegisterAsDropTarget: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.

7.2.27.2.7 IWebBrowser2.Get_Resizable Method

Pascal

```
function Get_Resizable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

This is Get_Resizable, a member of class IWebBrowser2.

7.2.27.2.8 IWebBrowser2.Get_Silent Method

Pascal

```
function Get_Silent: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

This is Get_Silent, a member of class IWebBrowser2.

7.2.27.2.9 IWebBrowser2.Get_TheaterMode Method

Pascal

```
function Get_TheaterMode: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

This is Get_TheaterMode, a member of class IWebBrowser2.

7.2.27.2.10 IWebBrowser2.Navigate2 Method

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate2, a member of class IWebBrowser2.

7.2.27.2.11 IWebBrowser2.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF; safecall;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class IWebBrowser2.

7.2.27.2.12 IWebBrowser2.Set_AddressBar Method

Pascal

```
procedure Set_AddressBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

This is Set_AddressBar, a member of class IWebBrowser2.

7.2.27.2.13 IWebBrowser2.Set_Offline Method

Pascal

```
procedure Set_Offline(pbOffline: WordBool); safecall;
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

This is Set_Offline, a member of class IWebBrowser2.

7.2.27.2.14 IWebBrowser2.Set_RegisterAsBrowser Method

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

This is Set_RegisterAsBrowser, a member of class IWebBrowser2.

7.2.27.2.15 IWebBrowser2.Set_RegisterAsDropTarget Method

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.

7.2.27.2.16 IWebBrowser2.Set_Resizable Method

Pascal

```
procedure Set_Resizable(Value: WordBool); safecall;
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

This is Set_Resizable, a member of class IWebBrowser2.

7.2.27.2.17 IWebBrowser2.Set_Silent Method

Pascal

```
procedure Set_Silent(pbSilent: WordBool); safecall;
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

This is Set_Silent, a member of class IWebBrowser2.

7.2.27.2.18 IWebBrowser2.Set_TheaterMode Method

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

This is Set_TheaterMode, a member of class IWebBrowser2.

7.2.27.2.19 IWebBrowser2.ShowBrowserBar Method

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); safecall;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class IWebBrowser2.

7.2.27.3 IWebBrowser2 Properties

The properties of the IWebBrowser2 class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 705)	This is AddressBar, a member of class IWebBrowser2.
	Offline (see page 705)	This is Offline, a member of class IWebBrowser2.
R	ReadyState (see page 705)	This is ReadyState, a member of class IWebBrowser2.
	RegisterAsBrowser (see page 706)	This is RegisterAsBrowser, a member of class IWebBrowser2.
	RegisterAsDropTarget (see page 706)	This is RegisterAsDropTarget, a member of class IWebBrowser2.
	Resizable (see page 706)	This is Resizable, a member of class IWebBrowser2.
	Silent (see page 706)	This is Silent, a member of class IWebBrowser2.
	TheaterMode (see page 707)	This is TheaterMode, a member of class IWebBrowser2.

7.2.27.3.1 IWebBrowser2.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class IWebBrowser2.

7.2.27.3.2 IWebBrowser2.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class IWebBrowser2.

7.2.27.3.3 IWebBrowser2.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class IWebBrowser2.

7.2.27.3.4 IWebBrowser2.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class IWebBrowser2.

7.2.27.3.5 IWebBrowser2.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class IWebBrowser2.

7.2.27.3.6 IWebBrowser2.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class IWebBrowser2.

7.2.27.3.7 IWebBrowser2.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class IWebBrowser2.

7.2.27.3.8 IWebBrowser2.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

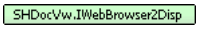
```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class IWebBrowser2.

7.2.28 IWebBrowser2Disp Interface

Class Hierarchy



Pascal

```
IWebBrowser2Disp = dispinterface;
```

C++

```
class IWebBrowser2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser2Disp.

7.2.28.1 IWebBrowser2Disp Members

The following tables list the members exposed by IWebBrowser2Disp.

Public Methods

	Name	Description
	ClientToWindow (see page 709)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 709)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 709)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 709)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 710)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 710)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 710)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 710)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 710)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 711)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 711)	This is QueryStatusWB, a member of class IWebBrowser2Disp.
	Quit (see page 711)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 711)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 712)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 712)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.

	Stop (see page 712)	This is Stop, a member of class IWebBrowser2Disp.

Public Properties

	Name	Description
	AddressBar (see page 713)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 713)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 713)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 714)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 714)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 714)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 714)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 714)	This is Height, a member of class IWebBrowser2Disp.
	HWnd (see page 715)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 715)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 715)	This is LocationName, a member of class IWebBrowser2Disp.
	LocationURL (see page 715)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 716)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 716)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 716)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 716)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 716)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 717)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 717)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.
	RegisterAsDropTarget (see page 717)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 717)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 718)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 718)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 718)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 718)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 718)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 719)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 719)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 719)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 719)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 720)	This is Width, a member of class IWebBrowser2Disp.

7.2.28.2 IWebBrowser2Disp Methods

The methods of the IWebBrowser2Disp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 709)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 709)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 709)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 709)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 710)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 710)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 710)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 710)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 710)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 711)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 711)	This is QueryStatusWB, a member of class IWebBrowser2Disp.

	Quit (see page 711)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 711)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 712)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 712)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.
	Stop (see page 712)	This is Stop, a member of class IWebBrowser2Disp.

7.2.28.2.1 IWebBrowser2Disp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowser2Disp.

7.2.28.2.2 IWebBrowser2Disp.ExecWB Method

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant);
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class IWebBrowser2Disp.

7.2.28.2.3 IWebBrowser2Disp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowser2Disp.

7.2.28.2.4 IWebBrowser2Disp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowser2Disp.

7.2.28.2.5 IWebBrowser2Disp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowser2Disp.

7.2.28.2.6 IWebBrowser2Disp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowser2Disp.

7.2.28.2.7 IWebBrowser2Disp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowser2Disp.

7.2.28.2.8 IWebBrowser2Disp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowser2Disp.

7.2.28.2.9 IWebBrowser2Disp.Navigate2 Method

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate2, a member of class IWebBrowser2Disp.

7.2.28.2.10 IWebBrowser2Disp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowser2Disp.

7.2.28.2.11 IWebBrowser2Disp.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class IWebBrowser2Disp.

7.2.28.2.12 IWebBrowser2Disp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowser2Disp.

7.2.28.2.13 IWebBrowser2Disp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowser2Disp.

7.2.28.2.14 IWebBrowser2Disp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowser2Disp.

7.2.28.2.15 IWebBrowser2Disp.ShowBrowserBar Method

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant);
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class IWebBrowser2Disp.

7.2.28.2.16 IWebBrowser2Disp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowser2Disp.

7.2.28.3 IWebBrowser2Disp Properties

The properties of the IWebBrowser2Disp class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 713)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 713)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 713)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 714)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 714)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 714)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 714)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 714)	This is Height, a member of class IWebBrowser2Disp.
	HWnd (see page 715)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 715)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 715)	This is LocationName, a member of class IWebBrowser2Disp.

	LocationURL (see page 715)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 716)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 716)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 716)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 716)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 716)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 717)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 717)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.
	RegisterAsDropTarget (see page 717)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 717)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 718)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 718)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 718)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 718)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 718)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 719)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 719)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 719)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 719)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 720)	This is Width, a member of class IWebBrowser2Disp.

7.2.28.3.1 IWebBrowser2Disp.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class IWebBrowser2Disp.

7.2.28.3.2 IWebBrowser2Disp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowser2Disp.

7.2.28.3.3 IWebBrowser2Disp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowser2Disp.

7.2.28.3.4 IWebBrowser2Disp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowser2Disp.

7.2.28.3.5 IWebBrowser2Disp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowser2Disp.

7.2.28.3.6 IWebBrowser2Disp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowser2Disp.

7.2.28.3.7 IWebBrowser2Disp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowser2Disp.

7.2.28.3.8 IWebBrowser2Disp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowser2Disp.

7.2.28.3.9 IWebBrowser2Disp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowser2Disp.

7.2.28.3.10 IWebBrowser2Disp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowser2Disp.

7.2.28.3.11 IWebBrowser2Disp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowser2Disp.

7.2.28.3.12 IWebBrowser2Disp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowser2Disp.

7.2.28.3.13 IWebBrowser2Disp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowser2Disp.

7.2.28.3.14 IWebBrowser2Disp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowser2Disp.

7.2.28.3.15 IWebBrowser2Disp.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class IWebBrowser2Disp.

7.2.28.3.16 IWebBrowser2Disp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowser2Disp.

7.2.28.3.17 IWebBrowser2Disp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowser2Disp.

7.2.28.3.18 IWebBrowser2Disp.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class IWebBrowser2Disp.

7.2.28.3.19 IWebBrowser2Disp.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class IWebBrowser2Disp.

7.2.28.3.20 IWebBrowser2Disp.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.

7.2.28.3.21 IWebBrowser2Disp.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class IWebBrowser2Disp.

7.2.28.3.22 IWebBrowser2Disp.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class IWebBrowser2Disp.

7.2.28.3.23 IWebBrowser2Disp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowser2Disp.

7.2.28.3.24 IWebBrowser2Disp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowser2Disp.

7.2.28.3.25 IWebBrowser2Disp.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class IWebBrowser2Disp.

7.2.28.3.26 IWebBrowser2Disp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowser2Disp.

7.2.28.3.27 IWebBrowser2Disp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowser2Disp.

7.2.28.3.28 IWebBrowser2Disp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowser2Disp.

7.2.28.3.29 IWebBrowser2Disp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowser2Disp.

7.2.28.3.30 IWebBrowser2Disp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowser2Disp.

7.2.28.3.31 IWebBrowser2Disp.Width Property

Pascal

```
property Width: Integer;
```

C++

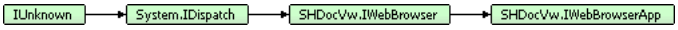
```
__property int Width;
```

Description

This is Width, a member of class IWebBrowser2Disp.

7.2.29 IWebBrowserApp Interface

Class Hierarchy



Pascal

```
IWebBrowserApp = interface(IWebBrowser);
```

C++

```
__interface IWebBrowserApp : public IWebBrowser;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserApp.

7.2.29.1 IWebBrowserApp Members

The following tables list the members exposed by IWebBrowserApp.

Public Methods

	Name	Description
	ClientToWindow (see page 722)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_Application (see page 689)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 689)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 689)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 689)	This is Get_Document, a member of class IWebBrowser.
	Get_FullName (see page 722)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 723)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 690)	This is Get_Height, a member of class IWebBrowser.
	Get_HWnd (see page 723)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 690)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 690)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 690)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_MenuBar (see page 723)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 723)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Parent (see page 691)	This is Get_Parent, a member of class IWebBrowser.

	Get_Path (see page 724)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 724)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 724)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 724)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 691)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 691)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 691)	This is Get_type_, a member of class IWebBrowser.
	Get_Visible (see page 724)	This is Get_Visible, a member of class IWebBrowserApp.
	Get_Width (see page 691)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 725)	This is GetProperty, a member of class IWebBrowserApp.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 692)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 692)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 692)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 692)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Navigate (see page 693)	This is Navigate, a member of class IWebBrowser.
	PutProperty (see page 725)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 725)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 693)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 693)	This is Refresh2, a member of class IWebBrowser.
	Set_FullScreen (see page 725)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 693)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 693)	This is Set_Left, a member of class IWebBrowser.
	Set_MenuBar (see page 726)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 726)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 726)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 726)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 694)	This is Set_Top, a member of class IWebBrowser.
	Set_Visible (see page 726)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 694)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 694)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
R	Application (see page 695)	This is Application, a member of class IWebBrowser.
R	Busy (see page 695)	This is Busy, a member of class IWebBrowser.
R	Container (see page 695)	This is Container, a member of class IWebBrowser.
R	Document (see page 695)	This is Document, a member of class IWebBrowser.
R	FullName (see page 727)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 727)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 696)	This is Height, a member of class IWebBrowser.
R	HWnd (see page 727)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 696)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 696)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 696)	This is LocationURL, a member of class IWebBrowser.
	MenuBar (see page 728)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 728)	This is Name, a member of class IWebBrowserApp.
R	Parent (see page 696)	This is Parent, a member of class IWebBrowser.
R	Path (see page 728)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 728)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 729)	This is StatusText, a member of class IWebBrowserApp.

	ToolBar (see page 729)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 697)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 697)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 697)	This is type_, a member of class IWebBrowser.
	Visible (see page 729)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 697)	This is Width, a member of class IWebBrowser.

7.2.29.2 IWebBrowserApp Methods

The methods of the IWebBrowserApp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 722)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_FullName (see page 722)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 723)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_HWnd (see page 723)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_MenuBar (see page 723)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 723)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Path (see page 724)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 724)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 724)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 724)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Visible (see page 724)	This is Get_Visible, a member of class IWebBrowserApp.
	GetProperty (see page 725)	This is GetProperty, a member of class IWebBrowserApp.
	PutProperty (see page 725)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 725)	This is Quit, a member of class IWebBrowserApp.
	Set_FullScreen (see page 725)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_MenuBar (see page 726)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 726)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 726)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 726)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Visible (see page 726)	This is Set_Visible, a member of class IWebBrowserApp.

7.2.29.2.1 IWebBrowserApp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT); safecall;
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserApp.

7.2.29.2.2 IWebBrowserApp.Get_FullName Method

Pascal

```
function Get_FullName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class IWebBrowserApp.

7.2.29.2.3 IWebBrowserApp.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class IWebBrowserApp.

7.2.29.2.4 IWebBrowserApp.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer; safecall;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class IWebBrowserApp.

7.2.29.2.5 IWebBrowserApp.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class IWebBrowserApp.

7.2.29.2.6 IWebBrowserApp.Get_Name Method

Pascal

```
function Get_Name: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class IWebBrowserApp.

7.2.29.2.7 IWebBrowserApp.Get_Path Method

Pascal

```
function Get_Path: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class IWebBrowserApp.

7.2.29.2.8 IWebBrowserApp.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class IWebBrowserApp.

7.2.29.2.9 IWebBrowserApp.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString; safecall;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class IWebBrowserApp.

7.2.29.2.10 IWebBrowserApp.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class IWebBrowserApp.

7.2.29.2.11 IWebBrowserApp.Get_Visible Method

Pascal

```
function Get_Visible: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class IWebBrowserApp.

7.2.29.2.12 IWebBrowserApp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserApp.

7.2.29.2.13 IWebBrowserApp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant); safecall;
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserApp.

7.2.29.2.14 IWebBrowserApp.Quit Method

Pascal

```
procedure Quit; safecall;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserApp.

7.2.29.2.15 IWebBrowserApp.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool); safecall;
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class IWebBrowserApp.

7.2.29.2.16 IWebBrowserApp.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class IWebBrowserApp.

7.2.29.2.17 IWebBrowserApp.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class IWebBrowserApp.

7.2.29.2.18 IWebBrowserApp.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString); safecall;
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class IWebBrowserApp.

7.2.29.2.19 IWebBrowserApp.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT); safecall;
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class IWebBrowserApp.

7.2.29.2.20 IWebBrowserApp.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class IWebBrowserApp.

7.2.29.3 IWebBrowserApp Properties

The properties of the IWebBrowserApp class are listed here.

Public Properties

	Name	Description
R	FullName (see page 727)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 727)	This is FullScreen, a member of class IWebBrowserApp.
R	HWnd (see page 727)	This is HWnd, a member of class IWebBrowserApp.
	MenuBar (see page 728)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 728)	This is Name, a member of class IWebBrowserApp.
R	Path (see page 728)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 728)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 729)	This is StatusText, a member of class IWebBrowserApp.
	ToolBar (see page 729)	This is ToolBar, a member of class IWebBrowserApp.
	Visible (see page 729)	This is Visible, a member of class IWebBrowserApp.

7.2.29.3.1 IWebBrowserApp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserApp.

7.2.29.3.2 IWebBrowserApp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserApp.

7.2.29.3.3 IWebBrowserApp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserApp.

7.2.29.3.4 IWebBrowserApp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserApp.

7.2.29.3.5 IWebBrowserApp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserApp.

7.2.29.3.6 IWebBrowserApp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserApp.

7.2.29.3.7 IWebBrowserApp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserApp.

7.2.29.3.8 IWebBrowserApp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserApp.

7.2.29.3.9 IWebBrowserApp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserApp.

7.2.29.3.10 IWebBrowserApp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

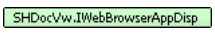
```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserApp.

7.2.30 IWebBrowserAppDisp Interface

Class Hierarchy



Pascal

```
IWebBrowserAppDisp = dispinterface;
```

C++

```
class IWebBrowserAppDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserAppDisp.

7.2.30.1 IWebBrowserAppDisp Members

The following tables list the members exposed by IWebBrowserAppDisp.

Public Methods

	Name	Description
	ClientToWindow (see page 731)	This is ClientToWindow, a member of class IWebBrowserAppDisp.
	GetProperty (see page 731)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 731)	This is GoBack, a member of class IWebBrowserAppDisp.
	GoForward (see page 731)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 732)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 732)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 732)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 732)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 733)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 733)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 733)	This is Refresh2, a member of class IWebBrowserAppDisp.
	Stop (see page 733)	This is Stop, a member of class IWebBrowserAppDisp.

Public Properties

	Name	Description
	Application (see page 734)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 734)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 734)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 735)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 735)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 735)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 735)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 736)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 736)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 736)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 736)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 736)	This is MenuBar, a member of class IWebBrowserAppDisp.
	Name (see page 737)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 737)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 737)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 737)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 738)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 738)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 738)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 738)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 738)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 739)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 739)	This is Width, a member of class IWebBrowserAppDisp.

7.2.30.2 IWebBrowserAppDisp Methods

The methods of the IWebBrowserAppDisp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 731)	This is ClientToWindow, a member of class IWebBrowserAppDisp.

	GetProperty (see page 731)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 731)	This is GoBack, a member of class IWebBrowserAppDisp.
	GoForward (see page 731)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 732)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 732)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 732)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 732)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 733)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 733)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 733)	This is Refresh2, a member of class IWebBrowserAppDisp.
	Stop (see page 733)	This is Stop, a member of class IWebBrowserAppDisp.

7.2.30.2.1 IWebBrowserAppDisp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserAppDisp.

7.2.30.2.2 IWebBrowserAppDisp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserAppDisp.

7.2.30.2.3 IWebBrowserAppDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserAppDisp.

7.2.30.2.4 IWebBrowserAppDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserAppDisp.

7.2.30.2.5 IWebBrowserAppDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserAppDisp.

7.2.30.2.6 IWebBrowserAppDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserAppDisp.

7.2.30.2.7 IWebBrowserAppDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserAppDisp.

7.2.30.2.8 IWebBrowserAppDisp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserAppDisp.

7.2.30.2.9 IWebBrowserAppDisp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserAppDisp.

7.2.30.2.10 IWebBrowserAppDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserAppDisp.

7.2.30.2.11 IWebBrowserAppDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserAppDisp.

7.2.30.2.12 IWebBrowserAppDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserAppDisp.

7.2.30.3 IWebBrowserAppDisp Properties

The properties of the IWebBrowserAppDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 734)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 734)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 734)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 735)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 735)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 735)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 735)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 736)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 736)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 736)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 736)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 736)	This is MenuBar, a member of class IWebBrowserAppDisp.
	Name (see page 737)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 737)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 737)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 737)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 738)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 738)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 738)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 738)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 738)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 739)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 739)	This is Width, a member of class IWebBrowserAppDisp.

7.2.30.3.1 IWebBrowserAppDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserAppDisp.

7.2.30.3.2 IWebBrowserAppDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserAppDisp.

7.2.30.3.3 IWebBrowserAppDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserAppDisp.

7.2.30.3.4 IWebBrowserAppDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserAppDisp.

7.2.30.3.5 IWebBrowserAppDisp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserAppDisp.

7.2.30.3.6 IWebBrowserAppDisp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserAppDisp.

7.2.30.3.7 IWebBrowserAppDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserAppDisp.

7.2.30.3.8 IWebBrowserAppDisp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserAppDisp.

7.2.30.3.9 IWebBrowserAppDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserAppDisp.

7.2.30.3.10 IWebBrowserAppDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserAppDisp.

7.2.30.3.11 IWebBrowserAppDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserAppDisp.

7.2.30.3.12 IWebBrowserAppDisp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserAppDisp.

7.2.30.3.13 IWebBrowserAppDisp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserAppDisp.

7.2.30.3.14 IWebBrowserAppDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserAppDisp.

7.2.30.3.15 IWebBrowserAppDisp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserAppDisp.

7.2.30.3.16 IWebBrowserAppDisp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserAppDisp.

7.2.30.3.17 IWebBrowserAppDisp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserAppDisp.

7.2.30.3.18 IWebBrowserAppDisp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserAppDisp.

7.2.30.3.19 IWebBrowserAppDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserAppDisp.

7.2.30.3.20 IWebBrowserAppDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserAppDisp.

7.2.30.3.21 IWebBrowserAppDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserAppDisp.

7.2.30.3.22 IWebBrowserAppDisp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserAppDisp.

7.2.30.3.23 IWebBrowserAppDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

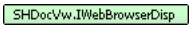
```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserAppDisp.

7.2.31 IWebBrowserDisp Interface

Class Hierarchy



Pascal

```
IWebBrowserDisp = dispinterface;
```

C++

```
class IWebBrowserDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserDisp.

7.2.31.1 IWebBrowserDisp Members

The following tables list the members exposed by IWebBrowserDisp.

Public Methods

	Name	Description
	GoBack (see page 740)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 741)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 741)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 741)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 741)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 741)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 742)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 742)	This is Stop, a member of class IWebBrowserDisp.

Public Properties

	Name	Description
	Application (see page 743)	This is Application, a member of class IWebBrowserDisp.
	Busy (see page 743)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 743)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 743)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 743)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 744)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 744)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 744)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 744)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 745)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 745)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 745)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 745)	This is Width, a member of class IWebBrowserDisp.

7.2.31.2 IWebBrowserDisp Methods

The methods of the IWebBrowserDisp class are listed here.

Public Methods

	Name	Description
	GoBack (see page 740)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 741)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 741)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 741)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 741)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 741)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 742)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 742)	This is Stop, a member of class IWebBrowserDisp.

7.2.31.2.1 IWebBrowserDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserDisp.

7.2.31.2.2 IWebBrowserDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserDisp.

7.2.31.2.3 IWebBrowserDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserDisp.

7.2.31.2.4 IWebBrowserDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserDisp.

7.2.31.2.5 IWebBrowserDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserDisp.

7.2.31.2.6 IWebBrowserDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserDisp.

7.2.31.2.7 IWebBrowserDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserDisp.

7.2.31.2.8 IWebBrowserDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserDisp.

7.2.31.3 IWebBrowserDisp Properties

The properties of the IWebBrowserDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 743)	This is Application, a member of class IWebBrowserDisp.
	Busy (see page 743)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 743)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 743)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 743)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 744)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 744)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 744)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 744)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 745)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 745)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 745)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 745)	This is Width, a member of class IWebBrowserDisp.

7.2.31.3.1 IWebBrowserDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserDisp.

7.2.31.3.2 IWebBrowserDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserDisp.

7.2.31.3.3 IWebBrowserDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserDisp.

7.2.31.3.4 IWebBrowserDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserDisp.

7.2.31.3.5 IWebBrowserDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserDisp.

7.2.31.3.6 IWebBrowserDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserDisp.

7.2.31.3.7 IWebBrowserDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserDisp.

7.2.31.3.8 IWebBrowserDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserDisp.

7.2.31.3.9 IWebBrowserDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserDisp.

7.2.31.3.10 IWebBrowserDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserDisp.

7.2.31.3.11 IWebBrowserDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserDisp.

7.2.31.3.12 IWebBrowserDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserDisp.

7.2.31.3.13 IWebBrowserDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserDisp.

7.3 Types

The following table lists types in this documentation.

Types

Name	Description
BrowserNavConstants (see page 748)	This is type SHDocVw.BrowserNavConstants.
CommandStateChangeConstants (see page 748)	This is type SHDocVw.CommandStateChangeConstants.
CScriptErrorList (see page 748)	This is type SHDocVw.CScriptErrorList.
InternetExplorer (see page 749)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 749)	OLECMDF (see page 750) influences the way in which an OLECMDID (see page 750) command is executed.
OLECMDF (see page 750)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 750)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 751)	This is type SHDocVw.POleVariant1.
RefreshConstants (see page 751)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 751)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 752)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 752)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 752)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 753)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 753)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 753)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 754)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 754)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 754)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 755)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 755)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 755)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 756)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 756)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 756)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 757)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 757)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 757)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 758)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 758)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 758)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 759)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 759)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 759)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 760)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 760)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 760)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 761)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 761)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 761)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 762)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 762)	This is type SHDocVw.TInternetExplorerWindowClosing.

TInternetExplorerWindowSetHeight (see page 762)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 763)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 764)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 764)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 764)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 765)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 765)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 765)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.
TWebBrowser_V1FrameNavigateComplete (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 766)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.
TWebBrowser_V1NavigateComplete (see page 767)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 767)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 768)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 768)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 768)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 769)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 769)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 769)	This is type SHDocVw.TWebBrowserBeforeNavigate2.
TWebBrowserClientToHostWindow (see page 770)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 770)	This is type SHDocVw.TWebBrowserCommandStateChange.
TWebBrowserDocumentComplete (see page 770)	This is type SHDocVw.TWebBrowserDocumentComplete.
TWebBrowserFileDownload (see page 771)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 771)	This is type SHDocVw.TWebBrowserNavigateComplete2.
TWebBrowserNavigateError (see page 771)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 772)	This is type SHDocVw.TWebBrowserNewWindow2.
TWebBrowserOnFullScreen (see page 772)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 772)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 773)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 773)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 773)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 774)	This is type SHDocVw.TWebBrowserOnVisible.
TWebBrowserPrintTemplateInstantiation (see page 774)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 774)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 775)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 775)	This is type SHDocVw.TWebBrowserProgressChange.
TWebBrowserPropertyChange (see page 775)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 776)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 776)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 776)	This is type SHDocVw.TWebBrowserTitleChange.
TWebBrowserUpdatePageStatus (see page 777)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 777)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 777)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 778)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 778)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 778)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 779)	This is type SHDocVw.TWebBrowserWindowSetWidth.
WebBrowser (see page 779)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 779)	This is type SHDocVw.WebBrowser_V1.

7.3.1 SHDocVw.BrowserNavConstants Type

Pascal

```
BrowserNavConstants = TOleEnum;
```

C++

```
TOleEnum BrowserNavConstants;
```

File

SHDocVw

Description

This is type SHDocVw.BrowserNavConstants.

7.3.2 SHDocVw.CommandStateChangeConstants Type

Pascal

```
CommandStateChangeConstants = TOleEnum;
```

C++

```
TOleEnum CommandStateChangeConstants;
```

File

SHDocVw

Description

This is type SHDocVw.CommandStateChangeConstants.

7.3.3 SHDocVw.CScriptErrorList Type

Pascal

```
CScriptErrorList = IScriptErrorList;
```

C++

```
IScriptErrorList CScriptErrorList;
```

File

SHDocVw

Description

This is type SHDocVw.CScriptErrorList.

7.3.4 SHDocVw.InternetExplorer Type

Pascal

```
InternetExplorer = IWebBrowser2;
```

C++

```
IWebBrowser2 InternetExplorer;
```

File

SHDocVw

Description

This is type SHDocVw.InternetExplorer.

7.3.5 SHDocVw.OLECMDEXECOPT Type

OLECMDF (see page 750) influences the way in which an OLECMDID (see page 750) command is executed.

Pascal

```
OLECMDEXECOPT = TOleEnum;
```

C++

```
TOleEnum OLECMDEXECOPT;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDEXECOPT indicates what an Automation object should do in response to an OLECMDID (see page 750) command. The following table lists the possible values:

Constant	Value	Meaning
OLECMDEXECOPT_DODEFAULT (see page 797)	0	Use the default behavior, whether prompting the user for input or not.
OLECMDEXECOPT_PROMPTUSER (see page 797)	1	Execute the command after obtaining user input.
OLECMDEXECOPT_DONTPROMPTUSER (see page 797)	2	Execute the command without prompting the user.
OLECMDEXECOPT_SHOWHELP (see page 798)	3	Show help for the command, but do not execute it.

See Also

OLECMDID (see page 750)

7.3.6 SHDocVw.OLECMDF Type

OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.

Pascal

```
OLECMDF = TOleEnum;
```

C++

```
TOleEnum OLECMDF;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDF is a set of flags that together indicate how a command can be used. These values can be or'd together:

Value	Meaning	
OLECMDF_SUPPORTED (see page 800)	The Automation object supports the command (but it may not be enabled).	
OLECMDF_ENABLED (see page 798)	The command is currently enabled.	
OLECMDF_LATCHED (see page 799)	The command is an on/off toggle that is currently on.	
OLECMDF_NINCHED (see page 799)	Not currently used.	
OLECMDF_INVISIBLE (see page 799)	Not currently used.	
OLECMDF_DEFHIDEONCTXTMENU (see page 798)	Not currently used.	

See Also

OLECMDID (see page 750)

7.3.7 SHDocVw.OLECMDID Type

OLECMDID represents a command that can be executed by an Automation object.

Pascal

```
OLECMDID = TOleEnum;
```

C++

```
TOleEnum OLECMDID;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDID represents a set of standard commands that can be executed by many Automation objects. Typically, each command represents a command in the user interface of the Automation object, such as a menu command. For specific details on how the Automation object interprets a particular command, see the documentation for the Automation object.

See Also

OLECMDEXECOPT (see page 749)

7.3.8 SHDocVw.POleVariant1 Type

Pascal

```
POleVariant1 = ^OleVariant;
```

C++

```
^OleVariant POleVariant1;
```

File

SHDocVw

Description

This is type SHDocVw.POleVariant1.

7.3.9 SHDocVw.RefreshConstants Type

Pascal

```
RefreshConstants = TOleEnum;
```

C++

```
TOleEnum RefreshConstants;
```

File

SHDocVw

Description

This is type SHDocVw.RefreshConstants.

7.3.10 SHDocVw.SearchAssistantOC Type

Pascal

```
SearchAssistantOC = ISearchAssistantOC3;
```

C++

```
ISearchAssistantOC3 SearchAssistantOC;
```

File

SHDocVw

Description

This is type SHDocVw.SearchAssistantOC.

7.3.11 SHDocVw.SecureLockIconConstants Type

Pascal

```
SecureLockIconConstants = TOleEnum;
```

C++

```
TOleEnum SecureLockIconConstants;
```

File

SHDocVw

Description

This is type SHDocVw.SecureLockIconConstants.

7.3.12 SHDocVw.ShellBrowserWindow Type

Pascal

```
ShellBrowserWindow = IWebBrowser2;
```

C++

```
IWebBrowser2 ShellBrowserWindow;
```

File

SHDocVw

Description

This is type SHDocVw.ShellBrowserWindow.

7.3.13 SHDocVw.ShellNameSpace Type

Pascal

```
ShellNameSpace = IShellNameSpace;
```

C++

```
IShellNameSpace ShellNameSpace;
```

File

SHDocVw

Description

This is type SHDocVw.ShellNameSpace.

7.3.14 SHDocVw.ShellUIHelper Type

Pascal

```
ShellUIHelper = IShellUIHelper;
```

C++

```
IShellUIHelper ShellUIHelper;
```

File

SHDocVw

Description

This is type SHDocVw.ShellUIHelper.

7.3.15 SHDocVw.ShellWindowFindWindowOptions Type

Pascal

```
ShellWindowFindWindowOptions = TOleEnum;
```

C++

```
TOleEnum ShellWindowFindWindowOptions;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowFindWindowOptions.

7.3.16 SHDocVw.ShellWindows Type

Pascal

```
ShellWindows = IShellWindows;
```

C++

```
IShellWindows ShellWindows;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindows.

7.3.17 SHDocVw.ShellWindowTypeConstants Type

Pascal

```
ShellWindowTypeConstants = TOleEnum;
```

C++

```
TOleEnum ShellWindowTypeConstants;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowTypeConstants.

7.3.18 SHDocVw.tagREADYSTATE Type

Pascal

```
tagREADYSTATE = TOleEnum;
```

C++

```
TOleEnum tagREADYSTATE;
```

File

SHDocVw

Description

This is type SHDocVw.tagREADYSTATE.

7.3.19 SHDocVw.TInternetExplorerBeforeNavigate2 Type

Pascal

```
TInternetExplorerBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData:
OleVariant; var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TInternetExplorerBeforeNavigate2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerBeforeNavigate2.

7.3.20 SHDocVw.TInternetExplorerClientToHostWindow Type

Pascal

```
TInternetExplorerClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY:
Integer) of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TInternetExplorerClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerClientToHostWindow.

7.3.21 SHDocVw.TInternetExplorerCommandStateChange Type

Pascal

```
TInternetExplorerCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) (
TInternetExplorerCommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerCommandStateChange.

7.3.22 SHDocVw.TInternetExplorerDocumentComplete Type

Pascal

```
TInternetExplorerDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerDocumentComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerDocumentComplete.

7.3.23 SHDocVw.TInternetExplorerFileDownload Type

Pascal

```
TInternetExplorerFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TInternetExplorerFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerFileDownload.

7.3.24 SHDocVw.TInternetExplorerNavigateComplete2 Type

Pascal

```
TInternetExplorerNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerNavigateComplete2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateComplete2.

7.3.25 SHDocVw.TInternetExplorerNavigateError Type

Pascal

```
TInternetExplorerNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
StatusCode: OleVariant; var Cancel: WordBool) ( TInternetExplorerNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateError.

7.3.26 SHDocVw.TInternetExplorerNewWindow2 Type

Pascal

```
TInternetExplorerNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) (
TInternetExplorerNewWindow2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNewWindow2.

7.3.27 SHDocVw.TInternetExplorerOnFullScreen Type

Pascal

```
TInternetExplorerOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TInternetExplorerOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnFullScreen.

7.3.28 SHDocVw.TInternetExplorerOnMenuBar Type

Pascal

```
TInternetExplorerOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TInternetExplorerOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnMenuBar.

7.3.29 SHDocVw.TInternetExplorerOnStatusBar Type

Pascal

```
TInternetExplorerOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TInternetExplorerOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnStatusBar.

7.3.30 SHDocVw.TInternetExplorerOnTheaterMode Type

Pascal

```
TInternetExplorerOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TInternetExplorerOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnTheaterMode.

7.3.31 SHDocVw.TInternetExplorerOnToolBar Type

Pascal

```
TInternetExplorerOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TInternetExplorerOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnToolBar.

7.3.32 SHDocVw.TInternetExplorerOnVisible Type

Pascal

```
TInternetExplorerOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TInternetExplorerOnVisible)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnVisible.

7.3.33 SHDocVw.TInternetExplorerPrintTemplateInstantiation Type

Pascal

```
TInternetExplorerPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp:
IDispatch) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.

7.3.34 SHDocVw.TInternetExplorerPrintTemplateTeardown Type

Pascal

```
TInternetExplorerPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.

7.3.35 SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type

Pascal

```
TInternetExplorerPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted:
WordBool) of object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TInternetExplorerPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.

7.3.36 SHDocVw.TInternetExplorerProgressChange Type

Pascal

```
TInternetExplorerProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) (
TInternetExplorerProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerProgressChange.

7.3.37 SHDocVw.TInternetExplorerPropertyChange Type

Pascal

```
TInternetExplorerPropertyChange = procedure (ASender: TObject; const szProperty: WideString)
of object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TInternetExplorerPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPropertyChange.

7.3.38 SHDocVw.TInternetExplorerSetSecureLockIcon Type

Pascal

```
TInternetExplorerSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TInternetExplorerSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerSetSecureLockIcon.

7.3.39 SHDocVw.TInternetExplorerStatusTextChange Type

Pascal

```
TInternetExplorerStatusTextChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerStatusTextChange.

7.3.40 SHDocVw.TInternetExplorerTitleChange Type

Pascal

```
TInternetExplorerTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerTitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerTitleChange.

7.3.41 SHDocVw.TInternetExplorerUpdatePageStatus Type

Pascal

```
TInternetExplorerUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var
nPage: OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TInternetExplorerUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerUpdatePageStatus.

7.3.42 SHDocVw.TInternetExplorerWindowClosing Type

Pascal

```
TInternetExplorerWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var
Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TInternetExplorerWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowClosing.

7.3.43 SHDocVw.TInternetExplorerWindowSetHeight Type

Pascal

```
TInternetExplorerWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TInternetExplorerWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetHeight.

7.3.44 SHDocVw.TInternetExplorerWindowSetLeft Type

Pascal

```
TInternetExplorerWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TInternetExplorerWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetLeft.

7.3.45 SHDocVw.TInternetExplorerWindowSetResizable Type

Pascal

```
TInternetExplorerWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of
object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TInternetExplorerWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetResizable.

7.3.46 SHDocVw.TInternetExplorerWindowSetTop Type

Pascal

```
TInternetExplorerWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TInternetExplorerWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetTop.

7.3.47 SHDocVw.TInternetExplorerWindowSetWidth Type

Pascal

```
TInternetExplorerWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TInternetExplorerWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetWidth.

7.3.48 SHDocVw.TShellNameSpaceFavoritesSelectionChange Type

Pascal

```
TShellNameSpaceFavoritesSelectionChange = procedure (ASender: TObject; cItems: Integer; hItem:
Integer; const strName: WideString; const strUrl: WideString; cVisits: Integer; const strDate:
WideString; fAvailableOffline: Integer) of object;
```

C++

```
(ASender: TObject; cItems: Integer; hItem: Integer; const strName: WideString; const strUrl:
WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline: Integer) (
TShellNameSpaceFavoritesSelectionChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.

7.3.49 SHDocVw.TShellWindowsWindowRegistered Type

Pascal

```
TShellWindowsWindowRegistered = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRegistered)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRegistered.

7.3.50 SHDocVw.TShellWindowsWindowRevoked Type

Pascal

```
TShellWindowsWindowRevoked = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRevoked)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRevoked.

7.3.51 SHDocVw.TWebBrowser_V1BeforeNavigate Type

Pascal

```
TWebBrowser_V1BeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1BeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1BeforeNavigate.

7.3.52 SHDocVw.TWebBrowser_V1CommandStateChange Type

Pascal

```
TWebBrowser_V1CommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowser_V1CommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1CommandStateChange.

7.3.53 SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type

Pascal

```
TWebBrowser_V1FrameBeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1FrameBeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.

7.3.54 SHDocVw.TWebBrowser_V1FrameNavigateComplete Type

Pascal

```
TWebBrowser_V1FrameNavigateComplete = procedure (ASender: TObject; const URL: WideString) of
object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1FrameNavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.

7.3.55 SHDocVw.TWebBrowser_V1FrameNewWindow Type

Pascal

```
TWebBrowser_V1FrameNewWindow = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) (
TWebBrowser_V1FrameNewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNewWindow.

7.3.56 SHDocVw.TWebBrowser_V1NavigateComplete Type

Pascal

```
TWebBrowser_V1NavigateComplete = procedure (ASender: TObject; const URL: WideString) of object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1NavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NavigateComplete.

7.3.57 SHDocVw.TWebBrowser_V1NewWindow Type

Pascal

```
TWebBrowser_V1NewWindow = procedure (ASender: TObject; const URL: WideString; Flags: Integer;
const TargetFrameName: WideString; var PostData: OleVariant; const Headers: WideString; var
Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) (
TWebBrowser_V1NewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NewWindow.

7.3.58 SHDocVw.TWebBrowser_V1ProgressChange Type

Pascal

```
TWebBrowser_V1ProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowser_V1ProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1ProgressChange.

7.3.59 SHDocVw.TWebBrowser_V1PropertyChange Type

Pascal

```
TWebBrowser_V1PropertyChange = procedure (ASender: TObject; const Property_: WideString) of
object;
```

C++

```
(ASender: TObject; const Property_: WideString) ( TWebBrowser_V1PropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1PropertyChange.

7.3.60 SHDocVw.TWebBrowser_V1Quit Type

Pascal

```
TWebBrowser_V1Quit = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowser_V1Quit)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1Quit.

7.3.61 SHDocVw.TWebBrowser_V1StatusTextChange Type

Pascal

```
TWebBrowser_V1StatusTextChange = procedure (ASender: TObject; const Text: WideString) of
object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1StatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1StatusTextChange.

7.3.62 SHDocVw.TWebBrowser_V1TitleChange Type

Pascal

```
TWebBrowser_V1TitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1TitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1TitleChange.

7.3.63 SHDocVw.TWebBrowserBeforeNavigate2 Type

Pascal

```
TWebBrowserBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData: OleVariant;
var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TWebBrowserBeforeNavigate2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserBeforeNavigate2.

7.3.64 SHDocVw.TWebBrowserClientToHostWindow Type

Pascal

```
TWebBrowserClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY: Integer)
of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TWebBrowserClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserClientToHostWindow.

7.3.65 SHDocVw.TWebBrowserCommandStateChange Type

Pascal

```
TWebBrowserCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowserCommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserCommandStateChange.

7.3.66 SHDocVw.TWebBrowserDocumentComplete Type

Pascal

```
TWebBrowserDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserDocumentComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserDocumentComplete.

7.3.67 SHDocVw.TWebBrowserFileDownload Type

Pascal

```
TWebBrowserFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowserFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserFileDownload.

7.3.68 SHDocVw.TWebBrowserNavigateComplete2 Type

Pascal

```
TWebBrowserNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserNavigateComplete2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNavigateComplete2.

7.3.69 SHDocVw.TWebBrowserNavigateError Type

Pascal

```
TWebBrowserNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
StatusCode: OleVariant; var Cancel: WordBool) ( TWebBrowserNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNavigateError.

7.3.70 SHDocVw.TWebBrowserNewWindow2 Type

Pascal

```
TWebBrowserNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) ( TWebBrowserNewWindow2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNewWindow2.

7.3.71 SHDocVw.TWebBrowserOnFullScreen Type

Pascal

```
TWebBrowserOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TWebBrowserOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnFullScreen.

7.3.72 SHDocVw.TWebBrowserOnMenuBar Type

Pascal

```
TWebBrowserOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TWebBrowserOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnMenuBar.

7.3.73 SHDocVw.TWebBrowserOnStatusBar Type

Pascal

```
TWebBrowserOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TWebBrowserOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnStatusBar.

7.3.74 SHDocVw.TWebBrowserOnTheaterMode Type

Pascal

```
TWebBrowserOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TWebBrowserOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnTheaterMode.

7.3.75 SHDocVw.TWebBrowserOnToolBar Type

Pascal

```
TWebBrowserOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TWebBrowserOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnToolBar.

7.3.76 SHDocVw.TWebBrowserOnVisible Type

Pascal

```
TWebBrowserOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TWebBrowserOnVisible)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnVisible.

7.3.77 SHDocVw.TWebBrowserPrintTemplateInstantiation Type

Pascal

```
TWebBrowserPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.

7.3.78 SHDocVw.TWebBrowserPrintTemplateTeardown Type

Pascal

```
TWebBrowserPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch) of
object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateTeardown.

7.3.79 SHDocVw.TWebBrowserPrivacyImpactedStateChange Type

Pascal

```
TWebBrowserPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted: WordBool) of object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TWebBrowserPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.

7.3.80 SHDocVw.TWebBrowserProgressChange Type

Pascal

```
TWebBrowserProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax: Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowserProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserProgressChange.

7.3.81 SHDocVw.TWebBrowserPropertyChange Type

Pascal

```
TWebBrowserPropertyChange = procedure (ASender: TObject; const szProperty: WideString) of object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TWebBrowserPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPropertyChange.

7.3.82 SHDocVw.TWebBrowserSetSecureLockIcon Type

Pascal

```
TWebBrowserSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TWebBrowserSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserSetSecureLockIcon.

7.3.83 SHDocVw.TWebBrowserStatusTextChange Type

Pascal

```
TWebBrowserStatusTextChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserStatusTextChange.

7.3.84 SHDocVw.TWebBrowserTitleChange Type

Pascal

```
TWebBrowserTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserTitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserTitleChange.

7.3.85 SHDocVw.TWebBrowserUpdatePageStatus Type

Pascal

```
TWebBrowserUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var nPage:
OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TWebBrowserUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserUpdatePageStatus.

7.3.86 SHDocVw.TWebBrowserWindowClosing Type

Pascal

```
TWebBrowserWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TWebBrowserWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowClosing.

7.3.87 SHDocVw.TWebBrowserWindowSetHeight Type

Pascal

```
TWebBrowserWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TWebBrowserWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetHeight.

7.3.88 SHDocVw.TWebBrowserWindowSetLeft Type

Pascal

```
TWebBrowserWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TWebBrowserWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetLeft.

7.3.89 SHDocVw.TWebBrowserWindowSetResizable Type

Pascal

```
TWebBrowserWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TWebBrowserWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetResizable.

7.3.90 SHDocVw.TWebBrowserWindowSetTop Type

Pascal

```
TWebBrowserWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TWebBrowserWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetTop.

7.3.91 SHDocVw.TWebBrowserWindowSetWidth Type

Pascal

```
TWebBrowserWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TWebBrowserWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetWidth.

7.3.92 SHDocVw.WebBrowser Type

Pascal

```
WebBrowser = IWebBrowser2;
```

C++

```
IWebBrowser2 WebBrowser;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser.

7.3.93 SHDocVw.WebBrowser_V1 Type

Pascal

```
WebBrowser_V1 = IWebBrowser;
```

C++

```
IWebBrowser WebBrowser_V1;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser_V1.

7.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CGID_IWebBrowser (see page 782)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 783)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 783)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 783)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 784)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 784)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 784)	This is constant SHDocVw.CLASS_ShellUIHelper.
CLASS_ShellWindows (see page 785)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 785)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 785)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 786)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 786)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 786)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 787)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 787)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 787)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 788)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 788)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 788)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 789)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 789)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 789)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 790)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 790)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 790)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 791)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 792)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 792)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 792)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 793)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 793)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 793)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 794)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 794)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 794)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 795)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 795)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 795)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 796)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 796)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 796)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

OLECMDEXECOPT_DONTPROMPTUSER (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 797)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 798)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 798)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 798)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 799)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 799)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 799)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 800)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 800)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 800)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 801)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 801)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 801)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 802)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 802)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 802)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.
OLECMDID_FIND (see page 803)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 803)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 803)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 804)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 804)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 804)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 805)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 805)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 805)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 806)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 806)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 806)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 807)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 807)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 807)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 808)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 808)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 808)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 809)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 809)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 809)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 810)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 810)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 810)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 811)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 811)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 811)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 812)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 812)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 812)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 813)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 813)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 813)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 814)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 814)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 814)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 815)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

OLECMDID_SPELL (see page 815)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 815)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 816)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 816)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 816)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 817)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 817)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 817)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 818)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 818)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 818)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 819)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 819)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 819)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 820)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 820)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 820)	This is constant SHDocVw.secureLockIconMixed.
secureLockIconSecure128Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 821)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 822)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 822)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 822)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 823)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 823)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 823)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 824)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 824)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 824)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 825)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 825)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 825)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

7.4.1 SHDocVw.CGID_IWebBrowser Constant

Pascal

```
CGID_IWebBrowser: TGUID = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

C++

```
TGUID CGID_IWebBrowser = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

File

SHDocVw

Description

This is constant SHDocVw.CGID_IWebBrowser.

7.4.2 SHDocVw.CLASS_CScriptErrorList Constant

Pascal

```
CLASS_CScriptErrorList: TGUID = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID CLASS_CScriptErrorList = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_CScriptErrorList.

7.4.3 SHDocVw.CLASS_InternetExplorer Constant

Pascal

```
CLASS_InternetExplorer: TGUID = '{0002DF01-0000-0000-C000-000000000046}';
```

C++

```
TGUID CLASS_InternetExplorer = '{0002DF01-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_InternetExplorer.

7.4.4 SHDocVw.CLASS_SearchAssistantOC Constant

Pascal

```
CLASS_SearchAssistantOC: TGUID = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID CLASS_SearchAssistantOC = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_SearchAssistantOC.

7.4.5 SHDocVw.CLASS_ShellBrowserWindow Constant

Pascal

```
CLASS_ShellBrowserWindow: TGUID = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

C++

```
TGUID CLASS_ShellBrowserWindow = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellBrowserWindow.

7.4.6 SHDocVw.CLASS_ShellNameSpace Constant

Pascal

```
CLASS_ShellNameSpace: TGUID = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID CLASS_ShellNameSpace = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellNameSpace.

7.4.7 SHDocVw.CLASS_ShellUIHelper Constant

Pascal

```
CLASS_ShellUIHelper: TGUID = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

C++

```
TGUID CLASS_ShellUIHelper = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellUIHelper.

7.4.8 SHDocVw.CLASS_ShellWindows Constant

Pascal

```
CLASS_ShellWindows: TGUID = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

C++

```
TGUID CLASS_ShellWindows = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellWindows.

7.4.9 SHDocVw.CLASS_WebBrowser Constant

Pascal

```
CLASS_WebBrowser: TGUID = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

C++

```
TGUID CLASS_WebBrowser = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser.

7.4.10 SHDocVw.CLASS_WebBrowser_V1 Constant

Pascal

```
CLASS_WebBrowser_V1: TGUID = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID CLASS_WebBrowser_V1 = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser_V1.

7.4.11 SHDocVw.CSC_NAVIGATEBACK Constant

Pascal

```
CSC_NAVIGATEBACK = $00000002;
```

C++

```
CSC_NAVIGATEBACK = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEBACK.

7.4.12 SHDocVw.CSC_NAVIGATEFORWARD Constant

Pascal

```
CSC_NAVIGATEFORWARD = $00000001;
```

C++

```
CSC_NAVIGATEFORWARD = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEFORWARD.

7.4.13 SHDocVw.CSC_UPDATECOMMANDS Constant

Pascal

```
CSC_UPDATECOMMANDS = $FFFFFFFF;
```

C++

```
CSC_UPDATECOMMANDS = 0xFFFFFFFF;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_UPDATECOMMANDS.

7.4.14 SHDocVw.DIID__SearchAssistantEvents Constant

Pascal

```
DIID__SearchAssistantEvents: TGUID = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID DIID__SearchAssistantEvents = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID__SearchAssistantEvents.

7.4.15 SHDocVw.DIID_DShellNameSpaceEvents Constant

Pascal

```
DIID_DShellNameSpaceEvents: TGUID = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID DIID_DShellNameSpaceEvents = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellNameSpaceEvents.

7.4.16 SHDocVw.DIID_DShellWindowsEvents Constant

Pascal

```
DIID_DShellWindowsEvents: TGUID = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

C++

```
TGUID DIID_DShellWindowsEvents = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellWindowsEvents.

7.4.17 SHDocVw.DIID_DWebBrowserEvents Constant

Pascal

```
DIID_DWebBrowserEvents: TGUID = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID DIID_DWebBrowserEvents = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents.

7.4.18 SHDocVw.DIID_DWebBrowserEvents2 Constant

Pascal

```
DIID_DWebBrowserEvents2: TGUID = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

C++

```
TGUID DIID_DWebBrowserEvents2 = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents2.

7.4.19 SHDocVw.dtlOcxPage Constant

Pascal

```
dtlOcxPage = 'ActiveX';
```

C++

```
dtlOcxPage = 'ActiveX';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlOcxPage.

7.4.20 SHDocVw.dtlServerPage Constant

Pascal

```
dtlServerPage = 'Servers';
```

C++

```
dtlServerPage = 'Servers';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlServerPage.

7.4.21 SHDocVw.HTMLID_FIND Constant

Pascal

```
HTMLID_FIND = 1;
```

C++

```
HTMLID_FIND = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_FIND.

7.4.22 SHDocVw.HTMLID_OPTIONS Constant

Pascal

```
HTMLID_OPTIONS = 3;
```

C++

```
HTMLID_OPTIONS = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_OPTIONS.

7.4.23 SHDocVw.HTMLID_VIEWSOURCE Constant

Pascal

```
HTMLID_VIEWSOURCE = 2;
```

C++

```
HTMLID_VIEWSOURCE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_VIEWSOURCE.

7.4.24 SHDocVw.IID_IScriptErrorList Constant

Pascal

```
IID_IScriptErrorList: TGUID = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID IID_IScriptErrorList = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IScriptErrorList.

7.4.25 SHDocVw.IID_ISearch Constant

Pascal

```
IID_ISearch: TGUID = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearch = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearch.

7.4.26 SHDocVw.IID_ISearchAssistantOC Constant

Pascal

```
IID_ISearchAssistantOC: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

C++

```
TGUID IID_ISearchAssistantOC = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC.

7.4.27 SHDocVw.IID_ISearchAssistantOC2 Constant

Pascal

```
IID_ISearchAssistantOC2: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

C++

```
TGUID IID_ISearchAssistantOC2 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC2.

7.4.28 SHDocVw.IID_ISearchAssistantOC3 Constant

Pascal

```
IID_ISearchAssistantOC3: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

C++

```
TGUID IID_ISearchAssistantOC3 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC3.

7.4.29 SHDocVw.IID_ISearches Constant

Pascal

```
IID_ISearches: TGUID = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearches = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearches.

7.4.30 SHDocVw.IID_IShellFavoritesNameSpace Constant

Pascal

```
IID_IShellFavoritesNameSpace: TGUID = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID IID_IShellFavoritesNameSpace = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellFavoritesNameSpace.

7.4.31 SHDocVw.IID_IShellNameSpace Constant

Pascal

```
IID_IShellNameSpace: TGUID = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

C++

```
TGUID IID_IShellNameSpace = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellNameSpace.

7.4.32 SHDocVw.IID_IShellUIHelper Constant

Pascal

```
IID_IShellUIHelper: TGUID = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

C++

```
TGUID IID_IShellUIHelper = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellUIHelper.

7.4.33 SHDocVw.IID_IShellWindows Constant

Pascal

```
IID_IShellWindows: TGUID = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

C++

```
TGUID IID_IShellWindows = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellWindows.

7.4.34 SHDocVw.IID_IWebBrowser Constant

Pascal

```
IID_IWebBrowser: TGUID = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID IID_IWebBrowser = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser.

7.4.35 SHDocVw.IID_IWebBrowser2 Constant

Pascal

```
IID_IWebBrowser2: TGUID = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

C++

```
TGUID IID_IWebBrowser2 = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser2.

7.4.36 SHDocVw.IID_IWebBrowserApp Constant

Pascal

```
IID_IWebBrowserApp: TGUID = '{0002DF05-0000-0000-C000-000000000046}';
```

C++

```
TGUID IID_IWebBrowserApp = '{0002DF05-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowserApp.

7.4.37 SHDocVw.LIBID_SHDocVw Constant

Pascal

```
LIBID_SHDocVw: TGUID = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID LIBID_SHDocVw = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.LIBID_SHDocVw.

7.4.38 SHDocVw.navAllowAutosearch Constant

Pascal

```
navAllowAutosearch = $00000010;
```

C++

```
navAllowAutosearch = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.navAllowAutosearch.

7.4.39 SHDocVw.navBrowserBar Constant

Pascal

```
navBrowserBar = $00000020;
```

C++

```
navBrowserBar = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.navBrowserBar.

7.4.40 SHDocVw.navNoHistory Constant

Pascal

```
navNoHistory = $00000002;
```

C++

```
navNoHistory = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoHistory.

7.4.41 SHDocVw.navNoReadFromCache Constant

Pascal

```
navNoReadFromCache = $00000004;
```

C++

```
navNoReadFromCache = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoReadFromCache.

7.4.42 SHDocVw.navNoWriteToCache Constant

Pascal

```
navNoWriteToCache = $00000008;
```

C++

```
navNoWriteToCache = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoWriteToCache.

7.4.43 SHDocVw.navOpenInNewWindow Constant

Pascal

```
navOpenInNewWindow = $00000001;
```

C++

```
navOpenInNewWindow = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.navOpenInNewWindow.

7.4.44 SHDocVw.OLECMDEXECOPT_DODEFAULT Constant

Pascal

```
OLECMDEXECOPT_DODEFAULT = $00000000;
```

C++

```
OLECMDEXECOPT_DODEFAULT = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

7.4.45 SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_DONTPROMPTUSER = $00000002;
```

C++

```
OLECMDEXECOPT_DONTPROMPTUSER = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.

7.4.46 SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_PROMPTUSER = $00000001;
```

C++

```
OLECMDEXECOPT_PROMPTUSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.

7.4.47 SHDocVw.OLECMDEXECOPT_SHOWHELP Constant

Pascal

```
OLECMDEXECOPT_SHOWHELP = $00000003;
```

C++

```
OLECMDEXECOPT_SHOWHELP = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.

7.4.48 SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant

Pascal

```
OLECMDF_DEFHIDEONCTXTMENU = $00000020;
```

C++

```
OLECMDF_DEFHIDEONCTXTMENU = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.

7.4.49 SHDocVw.OLECMDF_ENABLED Constant

Pascal

```
OLECMDF_ENABLED = $00000002;
```

C++

```
OLECMDF_ENABLED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_ENABLED.

7.4.50 SHDocVw.OLECMDF_INVISIBLE Constant

Pascal

```
OLECMDF_INVISIBLE = $00000010;
```

C++

```
OLECMDF_INVISIBLE = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_INVISIBLE.

7.4.51 SHDocVw.OLECMDF_LATCHED Constant

Pascal

```
OLECMDF_LATCHED = $00000004;
```

C++

```
OLECMDF_LATCHED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_LATCHED.

7.4.52 SHDocVw.OLECMDF_NINCHED Constant

Pascal

```
OLECMDF_NINCHED = $00000008;
```

C++

```
OLECMDF_NINCHED = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_NINCHED.

7.4.53 SHDocVw.OLECMDF_SUPPORTED Constant

Pascal

```
OLECMDF_SUPPORTED = $00000001;
```

C++

```
OLECMDF_SUPPORTED = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_SUPPORTED.

7.4.54 SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant

Pascal

```
OLECMDID_ALLOWUILESSSAVEAS = $0000002E;
```

C++

```
OLECMDID_ALLOWUILESSSAVEAS = 0x0000002E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.

7.4.55 SHDocVw.OLECMDID_CLEARSELECTION Constant

Pascal

```
OLECMDID_CLEARSELECTION = $00000012;
```

C++

```
OLECMDID_CLEARSELECTION = 0x00000012;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLEARSELECTION.

7.4.56 SHDocVw.OLECMDID_CLOSE Constant

Pascal

```
OLECMDID_CLOSE = $0000002D;
```

C++

```
OLECMDID_CLOSE = 0x0000002D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLOSE.

7.4.57 SHDocVw.OLECMDID_COPY Constant

Pascal

```
OLECMDID_COPY = $0000000C;
```

C++

```
OLECMDID_COPY = 0x0000000C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_COPY.

7.4.58 SHDocVw.OLECMDID_CUT Constant

Pascal

```
OLECMDID_CUT = $0000000B;
```

C++

```
OLECMDID_CUT = 0x0000000B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CUT.

7.4.59 SHDocVw.OLECMDID_DELETE Constant

Pascal

```
OLECMDID_DELETE = $00000021;
```

C++

```
OLECMDID_DELETE = 0x00000021;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DELETE.

7.4.60 SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant

Pascal

```
OLECMDID_DONTDOWNLOADCSS = $0000002F;
```

C++

```
OLECMDID_DONTDOWNLOADCSS = 0x0000002F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.

7.4.61 SHDocVw.OLECMDID_ENABLE_INTERACTION Constant

Pascal

```
OLECMDID_ENABLE_INTERACTION = $00000024;
```

C++

```
OLECMDID_ENABLE_INTERACTION = 0x00000024;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

7.4.62 SHDocVw.OLECMDID_FIND Constant

Pascal

```
OLECMDID_FIND = $00000020;
```

C++

```
OLECMDID_FIND = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_FIND.

7.4.63 SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_GETPRINTTEMPLATE = $00000034;
```

C++

```
OLECMDID_GETPRINTTEMPLATE = 0x00000034;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.

7.4.64 SHDocVw.OLECMDID_GETZOOMRANGE Constant

Pascal

```
OLECMDID_GETZOOMRANGE = $00000014;
```

C++

```
OLECMDID_GETZOOMRANGE = 0x00000014;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETZOOMRANGE.

7.4.65 SHDocVw.OLECMDID_HIDETOOLBARS Constant

Pascal

```
OLECMDID_HIDETOOLBARS = $00000018;
```

C++

```
OLECMDID_HIDETOOLBARS = 0x00000018;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HIDETOOLBARS.

7.4.66 SHDocVw.OLECMDID_HTTPEQUIV Constant

Pascal

```
OLECMDID_HTTPEQUIV = $00000022;
```

C++

```
OLECMDID_HTTPEQUIV = 0x00000022;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV.

7.4.67 SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant

Pascal

```
OLECMDID_HTTPEQUIV_DONE = $00000023;
```

C++

```
OLECMDID_HTTPEQUIV_DONE = 0x00000023;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.

7.4.68 SHDocVw.OLECMDID_NEW Constant

Pascal

```
OLECMDID_NEW = $00000002;
```

C++

```
OLECMDID_NEW = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_NEW.

7.4.69 SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant

Pascal

```
OLECMDID_ONTOOLBARACTIVATED = $0000001F;
```

C++

```
OLECMDID_ONTOOLBARACTIVATED = 0x0000001F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.

7.4.70 SHDocVw.OLECMDID_ONUNLOAD Constant

Pascal

```
OLECMDID_ONUNLOAD = $00000025;
```

C++

```
OLECMDID_ONUNLOAD = 0x00000025;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONUNLOAD.

7.4.71 SHDocVw.OLECMDID_OPEN Constant

Pascal

```
OLECMDID_OPEN = $00000001;
```

C++

```
OLECMDID_OPEN = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_OPEN.

7.4.72 SHDocVw.OLECMDID_PAGESETUP Constant

Pascal

```
OLECMDID_PAGESETUP = $00000008;
```

C++

```
OLECMDID_PAGESETUP = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PAGESETUP.

7.4.73 SHDocVw.OLECMDID_PASTE Constant

Pascal

```
OLECMDID_PASTE = $0000000D;
```

C++

```
OLECMDID_PASTE = 0x0000000D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTE.

7.4.74 SHDocVw.OLECMDID_PASTESPECIAL Constant

Pascal

```
OLECMDID_PASTESPECIAL = $0000000E;
```

C++

```
OLECMDID_PASTESPECIAL = 0x0000000E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTESPECIAL.

7.4.75 SHDocVw.OLECMDID_PREREFRESH Constant

Pascal

```
OLECMDID_PREREFRESH = $00000027;
```

C++

```
OLECMDID_PREREFRESH = 0x00000027;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PREREFRESH.

7.4.76 SHDocVw.OLECMDID_PRINT Constant

Pascal

```
OLECMDID_PRINT = $00000006;
```

C++

```
OLECMDID_PRINT = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT.

7.4.77 SHDocVw.OLECMDID_PRINT2 Constant

Pascal

```
OLECMDID_PRINT2 = $00000031;
```

C++

```
OLECMDID_PRINT2 = 0x00000031;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT2.

7.4.78 SHDocVw.OLECMDID_PRINTPREVIEW Constant

Pascal

```
OLECMDID_PRINTPREVIEW = $00000007;
```

C++

```
OLECMDID_PRINTPREVIEW = 0x00000007;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW.

7.4.79 SHDocVw.OLECMDID_PRINTPREVIEW2 Constant

Pascal

```
OLECMDID_PRINTPREVIEW2 = $00000032;
```

C++

```
OLECMDID_PRINTPREVIEW2 = 0x00000032;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.

7.4.80 SHDocVw.OLECMDID_PROPERTIES Constant

Pascal

```
OLECMDID_PROPERTIES = $0000000A;
```

C++

```
OLECMDID_PROPERTIES = 0x0000000A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTIES.

7.4.81 SHDocVw.OLECMDID_PROPERTYBAG2 Constant

Pascal

```
OLECMDID_PROPERTYBAG2 = $00000026;
```

C++

```
OLECMDID_PROPERTYBAG2 = 0x00000026;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTYBAG2.

7.4.82 SHDocVw.OLECMDID_REDO Constant

Pascal

```
OLECMDID_REDO = $00000010;
```

C++

```
OLECMDID_REDO = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REDO.

7.4.83 SHDocVw.OLECMDID_REFRESH Constant

Pascal

```
OLECMDID_REFRESH = $00000016;
```

C++

```
OLECMDID_REFRESH = 0x00000016;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REFRESH.

7.4.84 SHDocVw.OLECMDID_SAVE Constant

Pascal

```
OLECMDID_SAVE = $00000003;
```

C++

```
OLECMDID_SAVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVE.

7.4.85 SHDocVw.OLECMDID_SAVEAS Constant

Pascal

```
OLECMDID_SAVEAS = $00000004;
```

C++

```
OLECMDID_SAVEAS = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVEAS.

7.4.86 SHDocVw.OLECMDID_SAVECOPYAS Constant

Pascal

```
OLECMDID_SAVECOPYAS = $00000005;
```

C++

```
OLECMDID_SAVECOPYAS = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVECOPYAS.

7.4.87 SHDocVw.OLECMDID_SELECTALL Constant

Pascal

```
OLECMDID_SELECTALL = $00000011;
```

C++

```
OLECMDID_SELECTALL = 0x00000011;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SELECTALL.

7.4.88 SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant

Pascal

```
OLECMDID_SETDOWNLOADSTATE = $0000001D;
```

C++

```
OLECMDID_SETDOWNLOADSTATE = 0x0000001D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.

7.4.89 SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_SETPRINTTEMPLATE = $00000033;
```

C++

```
OLECMDID_SETPRINTTEMPLATE = 0x00000033;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.

7.4.90 SHDocVw.OLECMDID_SETPROGRESSMAX Constant

Pascal

```
OLECMDID_SETPROGRESSMAX = $00000019;
```

C++

```
OLECMDID_SETPROGRESSMAX = 0x00000019;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.

7.4.91 SHDocVw.OLECMDID_SETPROGRESSPOS Constant

Pascal

```
OLECMDID_SETPROGRESSPOS = $0000001A;
```

C++

```
OLECMDID_SETPROGRESSPOS = 0x0000001A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.

7.4.92 SHDocVw.OLECMDID_SETPROGRESSTEXT Constant

Pascal

```
OLECMDID_SETPROGRESSTEXT = $0000001B;
```

C++

```
OLECMDID_SETPROGRESSTEXT = 0x0000001B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.

7.4.93 SHDocVw.OLECMDID_SETTITLE Constant

Pascal

```
OLECMDID_SETTITLE = $0000001C;
```

C++

```
OLECMDID_SETTITLE = 0x0000001C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETTITLE.

7.4.94 SHDocVw.OLECMDID_SHOWFIND Constant

Pascal

```
OLECMDID_SHOWFIND = $0000002A;
```

C++

```
OLECMDID_SHOWFIND = 0x0000002A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWFIND.

7.4.95 SHDocVw.OLECMDID_SHOWMESSAGE Constant

Pascal

```
OLECMDID_SHOWMESSAGE = $00000029;
```

C++

```
OLECMDID_SHOWMESSAGE = 0x00000029;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWMESSAGE.

7.4.96 SHDocVw.OLECMDID_SHOWPAGESETUP Constant

Pascal

```
OLECMDID_SHOWPAGESETUP = $0000002B;
```

C++

```
OLECMDID_SHOWPAGESETUP = 0x0000002B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.

7.4.97 SHDocVw.OLECMDID_SHOWPRINT Constant

Pascal

```
OLECMDID_SHOWPRINT = $0000002C;
```

C++

```
OLECMDID_SHOWPRINT = 0x0000002C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPRINT.

7.4.98 SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant

Pascal

```
OLECMDID_SHOWSCRIPTERROR = $00000028;
```

C++

```
OLECMDID_SHOWSCRIPTERROR = 0x00000028;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

7.4.99 SHDocVw.OLECMDID_SPELL Constant

Pascal

```
OLECMDID_SPELL = $00000009;
```

C++

```
OLECMDID_SPELL = 0x00000009;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SPELL.

7.4.100 SHDocVw.OLECMDID_STOP Constant

Pascal

```
OLECMDID_STOP = $00000017;
```

C++

```
OLECMDID_STOP = 0x00000017;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOP.

7.4.101 SHDocVw.OLECMDID_STOPDOWNLOAD Constant

Pascal

```
OLECMDID_STOPDOWNLOAD = $0000001E;
```

C++

```
OLECMDID_STOPDOWNLOAD = 0x0000001E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.

7.4.102 SHDocVw.OLECMDID_UNDO Constant

Pascal

```
OLECMDID_UNDO = $0000000F;
```

C++

```
OLECMDID_UNDO = 0x0000000F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UNDO.

7.4.103 SHDocVw.OLECMDID_UPDATECOMMANDS Constant

Pascal

```
OLECMDID_UPDATECOMMANDS = $00000015;
```

C++

```
OLECMDID_UPDATECOMMANDS = 0x00000015;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.

7.4.104 SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant

Pascal

```
OLECMDID_UPDATEPAGESTATUS = $00000030;
```

C++

```
OLECMDID_UPDATEPAGESTATUS = 0x00000030;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.

7.4.105 SHDocVw.OLECMDID_ZOOM Constant

Pascal

```
OLECMDID_ZOOM = $00000013;
```

C++

```
OLECMDID_ZOOM = 0x00000013;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ZOOM.

7.4.106 SHDocVw.READYSTATE_COMPLETE Constant

Pascal

```
READYSTATE_COMPLETE = $00000004;
```

C++

```
READYSTATE_COMPLETE = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_COMPLETE.

7.4.107 SHDocVw.READYSTATE_INTERACTIVE Constant

Pascal

```
READYSTATE_INTERACTIVE = $00000003;
```

C++

```
READYSTATE_INTERACTIVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_INTERACTIVE.

7.4.108 SHDocVw.READYSTATE_LOADED Constant

Pascal

```
READYSTATE_LOADED = $00000002;
```

C++

```
READYSTATE_LOADED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADED.

7.4.109 SHDocVw.READYSTATE_LOADING Constant

Pascal

```
READYSTATE_LOADING = $00000001;
```

C++

```
READYSTATE_LOADING = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADING.

7.4.110 SHDocVw.READYSTATE_UNINITIALIZED Constant

Pascal

```
READYSTATE_UNINITIALIZED = $00000000;
```

C++

```
READYSTATE_UNINITIALIZED = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_UNINITIALIZED.

7.4.111 SHDocVw.REFRESH_COMPLETELY Constant

Pascal

```
REFRESH_COMPLETELY = 3;
```

C++

```
REFRESH_COMPLETELY = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_COMPLETELY.

7.4.112 SHDocVw.REFRESH_CONTINUE Constant

Pascal

```
REFRESH_CONTINUE = 2;
```

C++

```
REFRESH_CONTINUE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_CONTINUE.

7.4.113 SHDocVw.REFRESH_IFEXPIRED Constant

Pascal

```
REFRESH_IFEXPIRED = 1;
```

C++

```
REFRESH_IFEXPIRED = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_IFEXPIRED.

7.4.114 SHDocVw.REFRESH_NORMAL Constant

Pascal

```
REFRESH_NORMAL = 0;
```

C++

```
REFRESH_NORMAL = 0;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_NORMAL.

7.4.115 SHDocVw.secureLockIconMixed Constant

Pascal

```
secureLockIconMixed = $00000001;
```

C++

```
secureLockIconMixed = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconMixed.

7.4.116 SHDocVw.secureLockIconSecure128Bit Constant

Pascal

```
secureLockIconSecure128Bit = $00000006;
```

C++

```
secureLockIconSecure128Bit = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure128Bit.

7.4.117 SHDocVw.secureLockIconSecure40Bit Constant

Pascal

```
secureLockIconSecure40Bit = $00000003;
```

C++

```
secureLockIconSecure40Bit = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure40Bit.

7.4.118 SHDocVw.secureLockIconSecure56Bit Constant

Pascal

```
secureLockIconSecure56Bit = $00000004;
```

C++

```
secureLockIconSecure56Bit = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure56Bit.

7.4.119 SHDocVw.secureLockIconSecureFortezza Constant

Pascal

```
secureLockIconSecureFortezza = $00000005;
```

C++

```
secureLockIconSecureFortezza = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureFortezza.

7.4.120 SHDocVw.secureLockIconSecureUnknownBits Constant

Pascal

```
secureLockIconSecureUnknownBits = $00000002;
```

C++

```
secureLockIconSecureUnknownBits = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureUnknownBits.

7.4.121 SHDocVw.secureLockIconUnsecure Constant

Pascal

```
secureLockIconUnsecure = $00000000;
```

C++

```
secureLockIconUnsecure = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconUnsecure.

7.4.122 SHDocVw.SHDocVwMajorVersion Constant

Pascal

```
SHDocVwMajorVersion = 1;
```

C++

```
SHDocVwMajorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMajorVersion.

7.4.123 SHDocVw.SHDocVwMinorVersion Constant

Pascal

```
SHDocVwMinorVersion = 1;
```

C++

```
SHDocVwMinorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMinorVersion.

7.4.124 SHDocVw.SWC_3RDPARTY Constant

Pascal

```
SWC_3RDPARTY = $00000002;
```

C++

```
SWC_3RDPARTY = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_3RDPARTY.

7.4.125 SHDocVw.SWC_BROWSER Constant

Pascal

```
SWC_BROWSER = $00000001;
```

C++

```
SWC_BROWSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_BROWSER.

7.4.126 SHDocVw.SWC_CALLBACK Constant

Pascal

```
SWC_CALLBACK = $00000004;
```

C++

```
SWC_CALLBACK = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_CALLBACK.

7.4.127 SHDocVw.SWC_EXPLORER Constant

Pascal

```
SWC_EXPLORER = $00000000;
```

C++

```
SWC_EXPLORER = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_EXPLORER.

7.4.128 SHDocVw.SWFO_COOKIEPASSED Constant

Pascal

```
SWFO_COOKIEPASSED = $00000004;
```

C++

```
SWFO_COOKIEPASSED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_COOKIEPASSED.

7.4.129 SHDocVw.SWFO_INCLUDEPENDING Constant

Pascal

```
SWFO_INCLUDEPENDING = $00000002;
```

C++

```
SWFO_INCLUDEPENDING = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_INCLUDEPENDING.

7.4.130 SHDocVw.SWFO_NEEDDISPATCH Constant

Pascal

```
SWFO_NEEDDISPATCH = $00000001;
```

C++

```
SWFO_NEEDDISPATCH = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_NEEDDISPATCH.

8 ShellAnimations Namespace

This is namespace ShellAnimations.

Topics

Name	Description
Classes (see page 828)	The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 828)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

8.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 828)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

8.1.1 TShellResources Class

TShellResources provides forwards compability for programs that use system animations in Windows Vista.

Class Hierarchy

Pascal

```
TShellResources = class(TComponent);
```

C++

```
class TShellResources : public TComponent;
```

File

ShellAnimations

Description

Windows Vista shell animations are no longer accessible outside of the system; programs must now store local copies of these animations instead of linking to them as was done in the past.

Placing TShellResources onto a form copies the following system animation resources into the program:

- CopyFile.res
- CopyFiles.res
- DeleteFile.res
- EmptyRecycle.res
- FindComputer.res
- FindFiles.res
- FindFolder.res
- RecycleFile.res

TShellResources should only be used in programs that are run on Vista and use animations from the common library, otherwise this function adds unneeded overhead.

See Also

TAnimate

9 SimpleDS Namespace

This is namespace SimpleDS.

Topics

Name	Description
Classes (see page 830)	The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 830)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 838)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

9.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 830)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 838)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

9.1.1 TInternalSQLDataSet Class

Class Hierarchy

Pascal

```
TInternalSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TInternalSQLDataSet : public TCustomSQLDataSet;
```

File

SimpleDS

Description

This is class SimpleDS.TInternalSQLDataSet.

9.1.1.1 TInternalSQLDataSet Members

The following tables list the members exposed by TInternalSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.

	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.

	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.

Public Methods

	Name	Description
V	Create (see page 1425)	Creates an instance of a TCustomSQLDataSet (see page 1421) component.
V	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	Destroy (see page 1426)	Destroys the TCustomSQLDataSet (see page 1421) instance.
V	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
V	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
V	PrepareStatement (see page 1440)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1467)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.

	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	CommandText (see page 833)	Specifies the command the dataset executes.
	CommandType (see page 834)	Indicates the meaning of the CommandText property.
	DataSource (see page 835)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 836)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 836)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 837)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 837)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	SortFieldNames (see page 838)	Indicates the sort order when CommandType is ctTable.

9.1.1.2 TInternalSQLDataSet Properties

The properties of the TInternalSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 833)	Specifies the command the dataset executes.
	CommandType (see page 834)	Indicates the meaning of the CommandText property.
	DataSource (see page 835)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 836)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 836)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 837)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 837)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 838)	Indicates the sort order when CommandType is ctTable.

9.1.1.2.1 TInternalSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType property:

When CommandType is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL method is called.

When CommandType is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL method is called, depending on whether the stored procedure returns a set of records.

When CommandType is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params property to supply parameter values.

See Also

CommandType (see page 1456)

Params (see page 1463)

Open

ExecSQL (see page 1560)

SortFieldNames (see page 1466)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

9.1.1.2.2 TInternalSQLDataSet.CommandType Property

Indicates the meaning of the CommandText property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText property. TCustomSQLDataSet descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 1455)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

9.1.1.2.3 TInternalSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1455)

SQL (see page 1587)

Params (see page 1463)

MasterSource (see page 1615)

9.1.1.2.4 TInternalSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider

Fetching the Data using TSQLDataSet

9.1.1.2.5 TInternalSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1546)

SQLConnection (see page 1467)

9.1.1.2.6 TInternalSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1463)

CommandText (see page 1455)

SQL (see page 1587)

9.1.1.2.7 TInternalSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by

the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1462)

SQL (see page 1587)

StoredProcName (see page 1603)

CommandText (see page 1455)

TParam

Value

DataType

DataSource (see page 1457)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

9.1.1.2.8 TInternalSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1456)

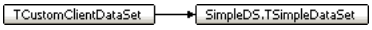
Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

9.1.2 TSimpleDataSet Class

TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

Class Hierarchy



Pascal

```
TSimpleDataSet = class(TCustomClientDataSet);
```

C++

```
class TSimpleDataSet : public TCustomClientDataSet;
```

File

SimpleDS

Description

TSimpleDataSet is a client dataset that uses an internal TSQLDataSet (see page 1556) and TDataSetProvider for fetching data and applying updates. It combines the fast access and easy deployment of a unidirectional dataset with a client dataset's ability to edit and navigate data.

TSimpleDataSet uses dbExpress for fast access to database information. As a client dataset, TSimpleDataSet caches that information in memory and saves any updates the application makes. With its internal dataset provider, TSimpleDataSet can apply those updates back to the database server. Thus, TSimpleDataSet works like TClientDataSet when it is connected to a local TSQLDataSet (see page 1556) via a provider, except that the source dataset and provider are internal.

TSimpleDataSet surfaces some of the properties and events of TSQLDataSet (see page 1556) so that you can specify the database server from which to fetch data, indicate what data to fetch from that server, influence what information is included in data packets, and provide input on the updating process.

In addition to using the source TSQLDataSet (see page 1556), TSimpleDataSet can read from and write to a dedicated file on disk. The internal provider and source dataset are not needed for working with file-based data, making TClientDataSet a more suitable choice if you are writing a purely file-based application. However, the ability to work with a file on disk makes TSimpleDataSet well-suited to briefcase model applications.

See Also

TDataSetProvider

Choosing the Type of Dataset for Caching Updates

Types of Unidirectional Datasets

9.1.2.1 TSimpleDataSet Members

The following tables list the members exposed by TSimpleDataSet.

Protected Methods

	Name	Description
	AllocConnection (see page 842)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 842)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocProvider (see page 843)	This is AllocProvider, a member of class TSimpleDataSet.
	Loaded (see page 843)	Initializes the component after the form file has been read into memory.
	Notification (see page 843)	Forwards notification messages to all owned components.
	OpenCursor (see page 844)	This is OpenCursor, a member of class TSimpleDataSet.
	PSGetCommandText (see page 844)	Gets the SQL command to be executed by PSExecute.
	SetConnection (see page 845)	This is SetConnection, a member of class TSimpleDataSet.

Public Methods

	Name	Description
	Create (see page 841)	Creates and initializes an instance of TSimpleDataSet (see page 838).
	Destroy (see page 842)	Destroys the instance of TSimpleDataSet (see page 838).

Published Events

	Name	Description
	AfterApplyUpdates (see page 873)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 873)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 875)	Occurs after an application closes a dataset.
	AfterDelete (see page 877)	Occurs after an application deletes a record.
	AfterEdit (see page 879)	Occurs after an application starts editing a record.
	AfterExecute (see page 880)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 881)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 882)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 882)	Occurs after an application inserts a new record.
	AfterOpen (see page 883)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 884)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 885)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 885)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 886)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 886)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 887)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 888)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 888)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 889)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 889)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 890)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 891)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 892)	Occurs before an application enters insert mode.
	BeforeOpen (see page 894)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 894)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 895)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 895)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 896)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 897)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 897)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 898)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 898)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 899)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 899)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 900)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Published Properties

	Name	Description
	Active (see page 846)	Specifies whether the client dataset contains data.
	Aggregates (see page 848)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 849)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 849)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 850)	Identifies the TSQLConnection (see page 1510) component that connects the dataset to a database server.
	Constraints (see page 850)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 851)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 851)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 852)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 853)	Points to the list of field definitions for the dataset.
	FileName (see page 856)	Specifies a file that stores the data of the client dataset.
	Filter (see page 857)	Specifies the text of the current filter.
	Filtered (see page 859)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 859)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 860)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 861)	Lists the fields to use as an index.
	IndexName (see page 862)	Identifies an index for the client dataset.
	MasterFields (see page 863)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 863)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 864)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 865)	Indicates the number or type of records in a single data packet.
	Params (see page 866)	Contains parameter values that are sent to the provider.
	ReadOnly (see page 871)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 871)	Indicates whether field and index definitions are saved with the client dataset.

9.1.2.2 TSimpleDataSet.Create Constructor

Creates and initializes an instance of TSimpleDataSet (see page 838).

Pascal

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

C++

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

Description

Call Create to create TSimpleDataSet (see page 838) at runtime. Datasets added to forms or data modules at design time are created automatically.

AOwner is the component that is responsible for freeing the TSimpleDataSet (see page 838) instance. If AOwner is nil (Delphi) or NULL (C++), the application must explicitly free the dataset.

Create creates the internal source dataset and provider that fetch data from the database server and apply updates back to the database server.

See Also

Owner

9.1.2.3 TSimpleDataSet.Destroy Destructor

Destroys the instance of TSimpleDataSet (see page 838).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

9.1.2.4 TSimpleDataSet Methods

The methods of the TSimpleDataSet class are listed here.

Protected Methods

	Name	Description
	AllocConnection (see page 842)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 842)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocProvider (see page 843)	This is AllocProvider, a member of class TSimpleDataSet.
	Loaded (see page 843)	Initializes the component after the form file has been read into memory.
	Notification (see page 843)	Forwards notification messages to all owned components.
	OpenCursor (see page 844)	This is OpenCursor, a member of class TSimpleDataSet.
	PSGetCommandText (see page 844)	Gets the SQL command to be executed by PSExecute.
	SetConnection (see page 845)	This is SetConnection, a member of class TSimpleDataSet.

9.1.2.4.1 TSimpleDataSet.AllocConnection Method

Pascal

```
procedure AllocConnection; virtual;
```

C++

```
virtual __fastcall AllocConnection();
```

Description

This is AllocConnection, a member of class TSimpleDataSet.

9.1.2.4.2 TSimpleDataSet.AllocDataSet Method

Pascal

```
procedure AllocDataSet; virtual;
```

C++

```
virtual __fastcall AllocDataSet();
```

Description

This is AllocDataSet, a member of class TSimpleDataSet.

9.1.2.4.3 TSimpleDataSet.AllocProvider Method

Pascal

```
procedure AllocProvider; virtual;
```

C++

```
virtual __fastcall AllocProvider();
```

Description

This is AllocProvider, a member of class TSimpleDataSet.

9.1.2.4.4 TSimpleDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

9.1.2.4.5 TSimpleDataSet.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

9.1.2.4.6 TSimpleDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSimpleDataSet.

9.1.2.4.7 TSimpleDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText

PSExecute

PSSetCommandText

GetRecords

Execute

Options

9.1.2.4.8 TSimpleDataSet.SetConnection Method

Pascal

```
procedure SetConnection(Value: TSQLConnection); virtual;
```

C++

```
virtual __fastcall SetConnection(TSQLConnection Value);
```

Description

This is SetConnection, a member of class TSimpleDataSet.

9.1.2.5 TSimpleDataSet Properties

The properties of the TSimpleDataSet class are listed here.

Published Properties

	Name	Description
	Active (see page 846)	Specifies whether the client dataset contains data.
	Aggregates (see page 848)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 849)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 849)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 850)	Identifies the TSQLConnection (see page 1510) component that connects the dataset to a database server.
	Constraints (see page 850)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 851)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 851)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 852)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 853)	Points to the list of field definitions for the dataset.
	FileName (see page 856)	Specifies a file that stores the data of the client dataset.
	Filter (see page 857)	Specifies the text of the current filter.
	Filtered (see page 859)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 859)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 860)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 861)	Lists the fields to use as an index.
	IndexName (see page 862)	Identifies an index for the client dataset.
	MasterFields (see page 863)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 863)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 864)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 865)	Indicates the number or type of records in a single data packet.
	Params (see page 866)	Contains parameter values that are sent to the provider.

	ReadOnly (see page 871)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 871)	Indicates whether field and index definitions are saved with the client dataset.

9.1.2.5.1 TSimpleDataSet.Active Property

Specifies whether the client dataset contains data.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine whether the client dataset makes its data available. When Active is false, the client dataset is closed; It can't manipulate the data it represents. When Active is true, data can be read or edited using the client dataset.

Setting Active to true:

1Fills the client dataset with data. Depending on the other properties of the client dataset, this data comes from:

The file specified by the FileName property.

The provider specified by the ProviderName property or SetProvider method. (For some TCustomClientDataSet descendants, this is an internal provider component)

The data that was active when the client dataset was deactivated (only if the application has run continuously since the dataset was deactivated).

2Triggers a BeforeOpen event.

3Sets the State property to dsBrowse.

4Opens a cursor into the dataset.

5Triggers an After Open event.

If the client dataset has never been active previously, setting Active to true defines the structure (metadata) of the client dataset. This structure is retrieved with the data packet from the provider or stored on disk, or, if there is no data packet, it is built using the current value of the FieldDefs property or from the persistent field components listed by the Fields property.

If an error occurs during the dataset open, the dataset state is set to dsInactive, and the cursor is closed.

Setting Active to false:

1Triggers a BeforeClose event.

2Sets the State property to dsInactive.

3Closes the cursor, saving the current data to disk if the FileName property is set, and saving the current data packet to a cache so that it can be restored when the client dataset is reopened.

4Triggers an AfterClose event.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
```

```
  information it needs for incremental data fetching. Before
  fetching the next data packet, the client dataset packages
  up the key value of the last record so that the provider
  knows where to begin the next data packet. It also sends
  some application-specific information, which is stored in
  Memo1.  This method is used to pass the sql statement and the
  value of the first field for the last record to the server so
  that the server can return the correct records. The complete
  project using this code example is in
  Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
  }
  procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
  var
    LastValue: OleVariant;
    CDSClone: TClientDataSet;
  begin
    if ClientDataSet1.Active then
    begin
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
  {
  The following code fragment illustrates how DataSets and
  DataSetCount can be used to ensure that an action is taken
  for every open dataset.  Assign CheckButtonActive as the
  AfterClose and AfterOpen event handlers for TClientDataSet1.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    if not RemoteServer1.Connected then
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
```

```
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

FileName

Open

Close

State

AfterOpen

BeforeOpen

FieldDefs

Fields

ProviderName

9.1.2.5.2 TSimpleDataSet.Aggregates Property

Lists all aggregates that apply to the client dataset.

Pascal

```
property Aggregates;
```

C++

```
__property Aggregates;
```

Description

Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

Do not use Aggregates to obtain summarized values maintained by the client dataset. Aggregates can contain TAggregate objects that are not currently active. Instead, use the ActiveAggs property, which represents only those TAggregate objects that are currently active.

Note: You do not need to add a TAggregate object to the Aggregates collection when you add an aggregate field using the Fields editor. The Fields editor automatically adds TAggregate objects to Aggregates for every aggregate field you define.

See Also

ActiveAggs

TAggregate

IndexName

TAggregateField

Obtaining Aggregate Values

Specifying Aggregates

Defining an Aggregate Field

9.1.2.5.3 TSimpleDataSet.AggregatesActive Property

Specifies whether the client dataset calculates and maintains aggregate values.

Pascal

```
property AggregatesActive;
```

C++

```
__property AggregatesActive;
```

Description

Use AggregatesActive to indicate whether the client dataset should maintain aggregate values. When AggregatesActive is false (the default), the client dataset does not support maintained aggregates. This allows the client dataset to avoid the overhead of calculating values that the application is not using. When AggregatesActive is true, the client dataset calculates and maintains all aggregate values specified by the Aggregates property that are compatible with the current index.

When aggregates are maintained, the Value method of every active aggregate object returns a value that reflects the current data in the client dataset. When users edit the data in the dataset, these values are recalculated to reflect the user's changes.

Note: To selectively enable and disable aggregates rather than turning them all on or off at once, use the Active property of individual TAggregate objects. These objects are available through the Aggregates property.

See Also

ActiveAggs

Aggregates

IndexName

Value

Active

9.1.2.5.4 TSimpleDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields;
```

C++

```
__property AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields

Calculating Fields

9.1.2.5.5 TSimpleDataSet.Connection Property

Identifies the TSQLConnection (see page 1510) component that connects the dataset to a database server.

Pascal

```
property Connection: TSQLConnection;
```

C++

```
__property TSQLConnection Connection;
```

Description

Use Connection to specify the component that connects this dataset to a database server. TSimpleDataSet (see page 838) can use either an external TSQLConnection (see page 1510) component or, if you set the ConnectionName property instead, an internally-created component. By using an external TSQLConnection (see page 1510) component, you can let multiple datasets share a single database connection.

See Also

Setting Up a Simple Dataset

Connecting to a Data Store Using TADOConnection

9.1.2.5.6 TSimpleDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint

CustomConstraint

ImportedConstraint

Constraining Data Values

9.1.2.5.7 TSimpleDataSet.DataSet Property

Provides access to the internal dataset that fetches data from the server.

Pascal

```
property DataSet: TInternalSQLDataSet;
```

C++

```
__property TInternalSQLDataSet DataSet;
```

Description

Read (see page 3727) DataSet to access the internal dataset that TSimpleDataSet (see page 838) uses to fetch data from the server. TSimpleDataSet (see page 838) populates its Data property by fetching data from this internal dataset using an internal provider component.

See Also

Data

9.1.2.5.8 TSimpleDataSet.DisableStringTrim Property

Specifies whether trailing spaces are trimmed from field values when records are posted.

Pascal

```
property DisableStringTrim;
```

C++

```
__property DisableStringTrim;
```

Description

When DisableStringTrim is false, trailing spaces are trimmed from field values when records are posted. When DisableStringTrim is true, field values are posted exactly as typed, including any trailing spaces.

9.1.2.5.9 TSimpleDataSet.FetchOnDemand Property

Indicates whether to fetch data packets from the provider on an as-needed basis.

Pascal

```
property FetchOnDemand;
```

C++

```
__property FetchOnDemand;
```

Description

By default FetchOnDemand is true, meaning that the client dataset fetches additional packets of data as needed (for example, as a user scrolls through data, or conducts a search). If the provider does not automatically include BLOB fields or nested detail sets in data packets, setting FetchOnDemand to true causes the client dataset to fetch these values when needed as well.

Set FetchOnDemand to false only if your application contains code that explicitly fetches data packets as needed using the GetNextPacket method. Similarly, if FetchOnDemand is false and the provider does not include BLOB data by default (Options includes poFetchBlobsOnDemand), the application must fetch BLOB fields explicitly using the FetchBlobs method. If the provider does not include nested detail sets (Options includes poFetchDetailsOnDemand) and FetchOnDemand is false, the application must fetch nested details explicitly using the FetchDetails method.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

GetNextPacket

Options

FetchBlobs

FetchDetails

Requesting Data from the Source Dataset or Document

9.1.2.5.10 TSimpleDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs;
```

C++

```
__property FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

```
/*
The following example shows how to create a table.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  if (Table1->Exists)
    MessageDlg("CustInfo table already exists.", mtWarning, TMsgDlgButtons() << mbOK, 0);
  else // Don't overwrite an existing table
  {
    // describe the fields in the table
    Table1->FieldDefs->Clear();
    TFieldDef *newDef = Table1->FieldDefs->AddFieldDef();
    newDef->Name = "Field1";
    newDef->DataType = ftInteger;
    newDef->Required = true;
    newDef = Table1->FieldDefs->AddFieldDef();
    newDef->Name = "Field2";
    newDef->DataType = ftString;
    newDef->Size = 30;
    // Next, describe any indexes
    Table1->IndexDefs->Clear();
    /* the 1st index has no name because it is a Paradox primary key */
    Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
    Table1->IndexDefs->Add("Fld2Index", "Field2", TIndexOptions() << ixCaseInsensitive);
    // Now that we have specified what we want, create the table
    Table1->CreateTable();
    Table1->Active = True;
    for (int i = 1; i <= 20; i++)
      Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  }
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->Active = True;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Close();
  Table1->DeleteTable();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
```

```
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
```

```
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.Active:= True;
end;
```

See Also

CreateTable

Fields

FieldValues

ClearFields

GetFieldList

GetFieldNames

FieldByName

FindField

SetFields

ObjectView

AggFields

CreateDataSet

Creating and Deleting Tables

9.1.2.5.11 TSimpleDataSet.FileName Property

Specifies a file that stores the data of the client dataset.

Pascal

```
property FileName;
```

C++

```
__property FileName;
```

Description

Set FileName if the client dataset always reads its data from and writes its data to a single file. When FileName represents a valid file name, the client dataset automatically writes its data to that file when it is closed. When FileName specifies the name of an existing file, the client dataset automatically reads its data from that file when it is opened.

Changes are not merged into the data when it is saved to the file. This allows an application that employs the "briefcase" model to set FileName without losing updates that are destined for a database server. Single-tiered applications should explicitly merge changes before closing the dataset and saving the data to disk.

Note: When client datasets write their data to a file, they overwrite the contents of the file. Any previous contents of the file are lost.

To read from or write to a different file, use the LoadFromFile and SaveToFile methods instead. These methods can also be used to load and save data at times other than when the Active property changes.

See Also

Open

Close

Active

LoadFromFile

SaveToFile

Loading Data from a File or Stream

Saving Data to a File or Stream

Using a Dedicated File on Disk

9.1.2.5.12 TSimpleDataSet.Filter Property

Specifies the text of the current filter.

Pascal

```
property Filter;
```

C++

```
__property Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a client dataset, only those records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. Client datasets support the following set of filter operations: See the Limiting what records appear link at the bottom of this topic for a complete list of available filter operations.

Operator or function ComparisonsExampleComment

Operator or function Comparisons	Example	Comment
=	State = 'CA'	
<>	State <> 'CA'	
>=	DateEntered >= '1/1/1998'	
<=	Total <= 100,000	
>	Percentile > 50	
<	Field1 < Field2	
BLANK	State <> 'CA' or State = BLANK	Blank records do not appear unless explicitly included in the filter.
IS NULL	Field1 IS NULL	
IS NOT NULL	Field1 IS NOT NULL	

Logical operators		
and	State = 'CA' and Country = 'US'	
or	State = 'CA' or State = 'MA'	
not	not (State = 'CA')	
Arithmetic operators		
+	Total + 5 > 100	Applies to numbers, strings, or date (time) + number
-	Field1 - 7 <> 10	Applies to numbers, dates, or date (time) - number
*	Discount * 100 > 20	Applies to numbers only
/	Discount > Total / 5	Applies to numbers only
String functions		
Upper	Upper(Field1) = 'ALWAYS'	
Lower	Lower(Field1 + Field2) = 'josp'	
Substring	Substring(DateString,8) = '1998' | Substring(DateString,1,3) = 'JAN'	Value goes from position of second argument to end or number of chars in third arg. First char has position 1.
Trim	Trim(Field1 + Field2) | Trim(Field1, '-')	removes third argument from front and back. If no third argument, trims spaces.
TrimLeft	TrimLeft(StringField) | TrimLeft(Field1, '$') <> ''	See Trim
TrimRight	TrimRight(StringField) | TrimRight(Field1, '.') <> ''	See Trim
DateTime functions		
Year	Year(DateField) = 2000	
Month	Month(DateField) <> 12	
Day	Day(DateField) = 1	
Hour	Hour(DateField) < 16	
Minute	Minute(DateField) = 0	
Second	Second(DateField) = 30	
GetDate	GetDate - DateField > 7	Represents current date and time
Date	DateField = Date(GetDate)	Returns the date portion of a datetime value
Time	TimeField > Time(GetDate)	Returns the time portion of a datetime value
Miscellaneous		
Like	Memo LIKE '%filters%'	Works like SQL-92 without the ESC clause. When applied to BLOB fields, FilterOptions determines whether case is considered.
In	Day(DateField) in (1,7)	Works like SQL-92. Second argument is a list of values all with the same type.
*	State = 'M*'	Wildcard for partial comparisons

See Also

Filtered

FilterOptions

Limiting Which Records Appear

FindFirst

FindLast

FindNext

FindPrior

OnFilterRecord

Filter

9.1.2.5.13 TSimpleDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter

FilterOptions

OnFilterRecord

Locate

Lookup

9.1.2.5.14 TSimpleDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions;
```

C++

```
__property FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter

Filtered

OnFilterRecord

Locate

Lookup

9.1.2.5.15 TSimpleDataSet.IndexDefs Property

Contains information about the indexes for a client dataset.

Pascal

```
property IndexDefs;
```

C++

```
__property IndexDefs;
```

Description

Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset. Set the value of IndexDefs before calling CreateDataSet to create a set of indexes with the client dataset.

Note: The index items in IndexDefs may not always reflect the current indexes available for a dataset. Before examining IndexDefs, call the IndexDefs.Update (Delphi) or IndexDefs->Update (C++) method to refresh the item list.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
```

```
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames

TIndexDef

CreateDataSet

Adding a New Index

9.1.2.5.16 TSimpleDataSet.IndexFieldNames Property

Lists the fields to use as an index.

Pascal

```
property IndexFieldNames;
```

C++

```
__property IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Note: Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Tip: Use IndexFieldNames to create sort orders on the fly at runtime.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs

IndexName

Creating an Index with IndexFieldNames

Adding a New Index

9.1.2.5.17 TSimpleDataSet.IndexName Property

Identifies an index for the client dataset.

Pascal

```
property IndexName;
```

C++

```
__property IndexName;
```

Description

Use IndexName to specify an alternative index for a client dataset.If IndexName is empty, the dataset's sort order is based on the IndexFieldNames property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
```

```
    CDS2.IndexDefs.Update;
    { Find a field named 'Field2' }
    for I := 0 to CDS2.IndexDefs.Count - 1 do
      if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
      begin
        { set that index as the current index for the dataset}
        CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
      end;
    CDS2.Active := True;
  end;
```

See Also

IndexFieldNames

Deleting and Switching Indexes

9.1.2.5.18 TSimpleDataSet.MasterFields Property

Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.

Pascal

```
property MasterFields;
```

C++

```
__property MasterFields;
```

Description

Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

See Also

MasterSource

Making the Table a Detail of Another Dataset

9.1.2.5.19 TSimpleDataSet.MasterSource Property

Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.

Pascal

```
property MasterSource;
```

C++

```
__property MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/detail relationships using nested datasets, where the detail dataset is stored with the master table's data. To assign a client dataset's data as the value of a nested detail, use the DataSetField property.

See Also

MasterFields

Making the Table a Detail of Another Dataset

9.1.2.5.20 TSimpleDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView;
```

C++

```
__property ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList

Fields

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

9.1.2.5.21 TSimpleDataSet.PacketRecords Property

Indicates the number or type of records in a single data packet.

Pascal

```
property PacketRecords;
```

C++

```
__property PacketRecords;
```

Description

Use PacketRecords to specify how many records should be contained in a single packet fetched from a provider, or to indicate that the packet returned should contain only metadata.

When a client dataset is instantiated, PacketRecords is automatically set to -1, meaning that a single packet should contain all records in the dataset.

If PacketRecords is greater than zero, then it specifies the number of records to return in a packet.

To retrieve the metadata for a dataset, set PacketRecords to 0. When PacketRecords is zero, the provider returns only information from its dataset that defines the database's structure, such as table, column, constraint, and domain definitions.

Warning: When PacketRecords is greater than 0, the client dataset must fetch its data incrementally. If the provider is on a stateless application server, the client dataset must inform the provider where to start fetching data. This communication can occur using a BeforeGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
```

```
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

FetchOnDemand

GetNextPacket

BeforeGetRecords

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

9.1.2.5.22 TSimpleDataSet.Params Property

Contains parameter values that are sent to the provider.

Pascal

```
property Params;
```

C++

```
__property Params;
```

Description

Use Params to specify parameter values that the provider should pass to a source dataset. If the client dataset uses an internal provider, Params represents the values for any parameters used by the SQL statement specified by the CommandText property. If the client dataset uses an external provider that is associated with a dataset that represents a query or stored procedure, these parameter values supply input parameter values for that query or stored procedure and return any output parameters of a stored procedure. If the provider is associated with a TTable or TSQLTable component, Params limits the records that are sent in data packets. After specifying the parameters (at design time or runtime), they are sent to the provider automatically when the client dataset fetches data or executes a query or stored procedure using its Execute method.

To send parameter values to a query or stored procedure, for every parameter on the query or stored procedure for which the client dataset should send a value, add a parameter object (TParam) to Params. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the provider when the client dataset fetches data.

To send parameter values to a TTable or TSQLTable, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the provider only contain records that match the assigned value on the corresponding fields. You can achieve the same affect by applying a filter to the field, but using Params is more efficient because the data need not be included in data packets.

Note: To initialize Params so that it reflects the parameters of a provider's associated query or stored procedure, call the FetchParams method.

Warning: The parameters are only applied to the provider when a query or stored procedure is run the first time the client dataset fetches records. To change parameter values and force a query or stored procedure to be rerun with new values, use the Execute method.

C++ Examples:

```
/*
  This example saves params from a ClientDataSet, alters them
  and restores them.
*/
TParams *SavedParams;

void __fastcall TForm1::SaveParamsClick(TObject *Sender)
{
  // Save the parameters for the TDataSet
  SavedParams->Assign(CDS->Params);
}

void __fastcall TForm1::AlterParamsClick(TObject *Sender)
{
  CDS->Params->CreateParam(ftString, "stringParam", ptInput);  // you can also edit the params
in the CDS dynamically
  CDS->Params->CreateParam(ftSmallint, "smallIntParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "intParam", ptInput);
  CDS->Params->CreateParam(ftWord, "wordParam", ptInput);
  CDS->Params->CreateParam(ftBoolean, "boolParam", ptInput);
  CDS->Params->CreateParam(ftFloat, "floatParam", ptInput);
  CDS->Params->CreateParam(ftCurrency, "currencyParam", ptInput);
  CDS->Params->CreateParam(ftBCD, "bcdParam", ptInput);
  CDS->Params->CreateParam(ftDate, "dateParam", ptInput);
  CDS->Params->CreateParam(ftTime, "timeParam", ptInput);
  CDS->Params->CreateParam(ftDateTime, "dateTimeParam", ptInput);
  // Call FetchParams to ensure parameters reflect server metadata
  CDS->FetchParams();
//  with CDS->Params do
  for (int I = 0; I < CDS->Params->Count; I++)
  {
    AnsiString ListItem = ListBox1->Items->Strings[I];
    switch (CDS->Params->Items[I]->DataType)
    {
      case ftString:
        CDS->Params->Items[I]->AsString = ListItem; break;
      case ftSmallint:
        CDS->Params->Items[I]->AsSmallInt = StrToIntDef(ListItem, 0); break;
      case ftInteger:
        CDS->Params->Items[I]->AsInteger = StrToIntDef(ListItem, 0); break;
      case ftWord:
        CDS->Params->Items[I]->AsWord = StrToIntDef(ListItem, 0); break;
      case ftBoolean:
        CDS->Params->Items[I]->AsBoolean = (ListItem == "True"); break;
      case ftFloat:
        CDS->Params->Items[I]->AsFloat = StrToFloat(ListItem); break;
      case ftCurrency:
        CDS->Params->Items[I]->AsCurrency = double(StrToFloat(ListItem)); break;
      case ftBCD:
        CDS->Params->Items[I]->AsBCD = StrToCurr(ListItem); break;
      case ftDate:
        CDS->Params->Items[I]->AsDate = StrToDate(ListItem); break;
      case ftTime:
        CDS->Params->Items[I]->AsTime = StrToTime(ListItem); break;
      case ftDateTime:
        CDS->Params->Items[I]->AsDateTime = StrToDateTime(ListItem); break;
```

```
    }
  }
}

void __fastcall TForm1::RestoreParamsClick(TObject *Sender)
{
  // Restore the parameters to TDataSet
  CDS->Params->AssignValues(SavedParams);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Initialize SavedParams
  SavedParams = new TParams;
  ListBox1->Items->Add("Here is a string");
  ListBox1->Items->Add("4");
  ListBox1->Items->Add("3456789");
  ListBox1->Items->Add("34");
  ListBox1->Items->Add("0");
  ListBox1->Items->Add("12.56");
  ListBox1->Items->Add("4.23");
  ListBox1->Items->Add("99.5");
  ListBox1->Items->Add("4/12/53");
  ListBox1->Items->Add("7:45:00");
  ListBox1->Items->Add("4/12/53 7:45:00");
}
```

Delphi Examples:

```
{
  This example uses the FetchParams method to sync the params
  with the server.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
```

```
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
```

```
    ClientDataSet1.FetchParams;
    with ClientDataSet1.Params do
    begin
      for I := 0 to Count - 1 do
      begin
        ListItem := ListBox1.Items[I];
        case Items[I].DataType of
          ftString:
            Items[I].AsString := ListItem;
          ftSmallInt:
            Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
          ftInteger:
            Items[I].AsInteger := StrToIntDef(ListItem, 0);
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

Params

Params

Active

FetchParams

Params (see page 1463)

Parameters

Getting Parameters from the Application Server

Limiting Records with Parameters

Overview of Using Cached Updates

Passing Parameters to the Source Dataset

Sending Query or Stored Procedure Parameters

Setting Up a Simple Dataset

9.1.2.5.23 TSimpleDataSet.ReadOnly Property

Specifies whether the client dataset is read-only for this application.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset's CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider's Options property can limit whether data can be edited, and if so, what types of edits are allowed.

See Also

Active

CanModify

Options

9.1.2.5.24 TSimpleDataSet.StoreDefs Property

Indicates whether field and index definitions are saved with the client dataset.

Pascal

```
property StoreDefs;
```

C++

```
__property StoreDefs;
```

Description

If StoreDefs is true, the client dataset's index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet method into a one-step procedure that creates fields and indexes at runtime.

See Also

FieldDefs

IndexDefs

CreateDataSet

9.1.2.6 TSimpleDataSet Events

The events of the TSimpleDataSet class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 873)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 873)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 875)	Occurs after an application closes a dataset.
	AfterDelete (see page 877)	Occurs after an application deletes a record.
	AfterEdit (see page 879)	Occurs after an application starts editing a record.
	AfterExecute (see page 880)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 881)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 882)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 882)	Occurs after an application inserts a new record.
	AfterOpen (see page 883)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 884)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 885)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 885)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 886)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 886)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 887)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 888)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 888)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 889)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 889)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 890)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 891)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 892)	Occurs before an application enters insert mode.
	BeforeOpen (see page 894)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 894)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 895)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 895)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 896)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 897)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 897)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 898)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 898)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 899)	Occurs when an application inserts or appends a new dataset record.

	OnPostError (see page 899)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 900)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

9.1.2.6.1 TSimpleDataSet.AfterApplyUpdates Event

Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.

Pascal

```
property AfterApplyUpdates;
```

C++

```
__property AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to respond to custom information returned to the client dataset from a provider's AfterApplyUpdates event handler. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

6.The client dataset generates an OnReconcileError for every record that could not be applied by the provider.

See Also

ApplyUpdates

AS_ApplyUpdates

applyupdates

AfterApplyUpdates

BeforeApplyUpdates

OnReconcileError

Applying Updates

Communicating with the Client Dataset

9.1.2.6.2 TSimpleDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel;
```

C++

```
__property AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
```

```
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterCancel:= Table1AfterCancel;
  Table1.Active:= True;
end;
```

See Also

BeforeCancel

Cancel

State

9.1.2.6.3 TSimpleDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active

State

BeforeClose

Close

Params (see page 1546)

Opening and Closing Datasets

9.1.2.6.4 TSimpleDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete;
```

C++

```
__property AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
```

```
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
```

```
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterDelete:= Table1AfterDelete;
  Table1.Active:= True;
end;
```

See Also

State

BeforeDelete

Delete

9.1.2.6.5 TSimpleDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit;
```

C++

```
__property AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit

Edit

9.1.2.6.6 TSimpleDataSet.AfterExecute Event

Occurs after the client dataset receives the results from an execute command.

Pascal

```
property AfterExecute;
```

C++

```
__property AfterExecute;
```

Description

Write an AfterExecute event handler to respond to custom information returned to the client dataset from a provider's AfterExecute event handler. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing an SQL command from the client. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute

AS_Execute

Execute

AfterExecute

BeforeExecute

CommandText

Communicating with the Client Dataset

9.1.2.6.7 TSimpleDataSet.AfterGetParams Event

Occurs after the client dataset receives parameter values from the provider.

Pascal

```
property AfterGetParams;
```

C++

```
__property AfterGetParams;
```

Description

Write an AfterGetParams event handler to respond to custom information returned to the client dataset from a provider's AfterGetParams event handler. AfterGetParams is part of the mechanism by which a client dataset and a provider communicate information when the client application calls FetchParams. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams

AS_GetParams

GetParams

aftergetparams

BeforeGetParams

Communicating with the Client Dataset

9.1.2.6.8 TSimpleDataSet.AfterGetRecords Event

Occurs after the client dataset fetches a data packet from the provider.

Pascal

```
property AfterGetRecords;
```

C++

```
__property AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to respond to custom information returned to the client dataset from a provider's AfterGetRecords event handler. AfterGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1. The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.
2. The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.
3. The provider generates an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.
4. The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.
5. The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

See Also

AS_GetRecords

GetRecords

AfterGetRecords

BeforeGetRecords

Communicating with the Client Dataset

9.1.2.6.9 TSimpleDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert;
```

C++

```
__property AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert

Append

Insert

9.1.2.6.10 TSimpleDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    I: Integer;
  begin
    if not RemoteServer1.Connected then
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

Active

State

BeforeOpen

Open

Opening and Closing Datasets

9.1.2.6.11 TSimpleDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost;
```

C++

```
__property AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
```

```
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost

Post

9.1.2.6.12 TSimpleDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh;
```

C++

```
__property AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh

Refresh

9.1.2.6.13 TSimpleDataSet.AfterRowRequest Event

Occurs after the client dataset fetches new information about the current record.

Pascal

```
property AfterRowRequest;
```

C++

```
__property AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to respond to custom information returned to the client dataset from a provider's AfterRowRequest event handler. AfterRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or

respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

9.1.2.6.14 TSimpleDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll

First

Last

MoveBy

Next

Prior

Navigating Datasets

9.1.2.6.15 TSimpleDataSet.BeforeApplyUpdates Event

Occurs before the client dataset applies updates to the remote server.

Pascal

```
property BeforeApplyUpdates;
```

C++

```
__property BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to send custom information to the client dataset's provider. BeforeApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider's AfterApplyUpdates event handler.

See Also

ApplyUpdates

AS_ApplyUpdates

applyupdates

BeforeApplyUpdates

AfterApplyUpdates

Applying Updates

Communicating with the Client Dataset

9.1.2.6.16 TSimpleDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel;
```

C++

```
__property BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel

Delete

Edit

Insert

9.1.2.6.17 TSimpleDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active

AfterClose

Close

Opening and Closing Datasets

Posting Data

9.1.2.6.18 TSimpleDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete;
```

C++

```
__property BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete

Delete

Abort (see page 4111)

9.1.2.6.19 TSimpleDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit;
```

C++

```
__property BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State

AfterEdit

Edit

9.1.2.6.20 TSimpleDataSet.BeforeExecute Event

Occurs before the client dataset executes the provider's query or stored procedure.

Pascal

```
property BeforeExecute;
```

C++

```
__property BeforeExecute;
```

Description

Write a BeforeExecute event handler to send custom information to the client dataset's provider. BeforeExecute is part of the

mechanism by which a client dataset and a provider communicate information when explicitly executing an SQL command by the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute

AS_Execute

Execute

beforeexecute

AfterExecute

CommandText

Communicating with the Client Dataset

9.1.2.6.21 TSimpleDataSet.BeforeGetParams Event

Occurs before the client dataset fetches parameter values from the provider.

Pascal

```
property BeforeGetParams;
```

C++

```
__property BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to send custom information to the client dataset's provider. BeforeGetparams is part of the mechanism by which a client dataset and a provider communicate information when passing parameter values from the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams

AS_GetParams

GetParams

BeforeGetParams

AfterGetParams

Communicating with the Client Dataset

9.1.2.6.22 TSimpleDataSet.BeforeGetRecords Event

Occurs before the client dataset fetches a data packet from the provider.

Pascal

```
property BeforeGetRecords;
```

C++

```
__property BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to send custom information to the client dataset's provider. BeforeGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
```

```
  knows where to begin the next data packet. It also sends
  some application-specific information, which is stored in
  Memo1.  This method is used to pass the sql statement and the
  value of the first field for the last record to the server so
  that the server can return the correct records. The complete
  project using this code example is in
  Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
  }
  procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
  var
    LastValue: OleVariant;
    CDSClone: TClientDataSet;
  begin
    if ClientDataSet1.Active then
    begin
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

AS_GetRecords

GetRecords

BeforeGetRecords

AfterGetRecords

Communicating with the Client Dataset

9.1.2.6.23 TSimpleDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert;
```

C++

```
__property BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
```

```
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State

AfterInsert

Append

Insert

9.1.2.6.24 TSimpleDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active

AfterOpen

Open

Opening and Closing Datasets

9.1.2.6.25 TSimpleDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost;
```

C++

```
__property BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
```

```
  and calls the Abort procedure if the control is empty; Abort
  cancels the post before it happens.
  }
  procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
  begin
    if DBEdit1.Text = '' then
      Abort;
    DataSet.Fields[2].AsString := DBEdit1.Text;
  end;
```

See Also

AfterPost

Post

Abort (see page 4111)

Working with Field Component Methods at Runtime

9.1.2.6.26 TSimpleDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh;
```

C++

```
__property BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh

Refresh

9.1.2.6.27 TSimpleDataSet.BeforeRowRequest Event

Occurs before the client dataset fetches new information about the current record.

Pascal

```
property BeforeRowRequest;
```

C++

```
__property BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to send custom information to the client dataset's provider. BeforeRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is

passed to the provider as the OwnerData parameter.

2.The provider on the application server receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

9.1.2.6.28 TSimpleDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll

First

Last

MoveBy

Next

Prior

Navigating Datasets

9.1.2.6.29 TSimpleDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active

AutoCalcFields

Open

Post

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

9.1.2.6.30 TSimpleDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError;
```

C++

```
__property OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError

OnPostError

Delete

9.1.2.6.31 TSimpleDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError;
```

C++

```
__property OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError

OnPostError

Edit

Insert

9.1.2.6.32 TSimpleDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter

Filtered

FilterOptions

State

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

9.1.2.6.33 TSimpleDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord;
```

C++

```
__property OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert

BeforeInsert

9.1.2.6.34 TSimpleDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError;
```

C++

```
__property OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError

OnEditError

Post

9.1.2.6.35 TSimpleDataSet.OnReconcileError Event

Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Pascal

```
property OnReconcileError;
```

C++

```
__property OnReconcileError;
```

Description

Write an OnReconcileError event handler to respond to error conditions that arise when the Reconcile method cycles through records that could not be applied to the database. This event handles only those update errors that were not handled in an OnUpdateError event handler.

The OnReconcileError event handler is called once for each record that could not be applied to the dataset or handled by the provider.

Warning: A provider can't detect when an update conflicts with another user's changes to a memo field. This situation does not generate an OnReconcileError event.

See Also

Reconcile

OldValue

NewValue

CurValue

ApplyUpdates

OnUpdateError

OnUpdateError

Applying Updates

Setting Up a Simple Dataset

Reconciling Update Errors

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

10 SiteComp Namespace

This is namespace SiteComp.

Topics

Name	Description
Classes (see page 905)	The following table lists classes in this documentation.
Interfaces (see page 919)	The following table lists interfaces in this documentation.
Functions (see page 1061)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1062)	The following table lists structs, records, enums in this documentation.
Types (see page 1067)	The following table lists types in this documentation.
Variables (see page 1069)	The following table lists variables in this documentation.
Constants (see page 1072)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 905)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 907)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 910)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 915)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 918)	This is class SiteComp.TScriptObject.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 1072)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 1072)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 1073)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 1073)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 1061)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 1061)	This is function SiteComp.FullyQualifiedName.

Interfaces

	Name	Description
	IActionImageProducer (see page 920)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 923)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 924)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 925)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 927)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 929)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 934)	This is class SiteComp.IAdapterError.
	IAdapterErrors (see page 937)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 940)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 941)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 943)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 945)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 947)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 951)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 954)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 955)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 956)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 957)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 958)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 959)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 960)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 961)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 962)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 963)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 964)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 965)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 966)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 968)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 969)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 971)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 972)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 973)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 976)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 977)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 978)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 979)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 982)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 983)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 984)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 987)	This is class SiteComp.IInternetExecuteScript.
	IInternetFileTypes (see page 988)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 989)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 990)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 992)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 993)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 994)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

	IIteratorSupport (see page 996)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 998)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 999)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 1001)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 1006)	This is class SiteComp.IPageResult.
	ISessionsService (see page 1007)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 1010)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 1011)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 1013)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 1014)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 1017)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 1019)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 1022)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 1023)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 1025)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 1026)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 1028)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 1029)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 1030)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 1031)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 1032)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 1034)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 1037)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 1038)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 1039)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 1040)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 1041)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 1042)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 1043)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 1045)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 1049)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 1051)	IWebSessionIDs lists the session IDs (see page 1053) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 1053)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 1055)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 1056)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 1057)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 1059)	This is class SiteComp.IWebValueIsEqual.

Structs, Records, Enums

	Name	Description
	TAdapterActionHTMLElementType (see page 1062)	This is record SiteComp.TAdapterActionHTMLElementType.
	TAdapterDisplayCharacteristic (see page 1062)	This is record SiteComp.TAdapterDisplayCharacteristic.
	TAdapterDisplayHTMLElementType (see page 1063)	This is record SiteComp.TAdapterDisplayHTMLElementType.
	TAdapterDisplayViewModeType (see page 1063)	This is record SiteComp.TAdapterDisplayViewModeType.
	TAdapterInputHTMLElementType (see page 1064)	This is record SiteComp.TAdapterInputHTMLElementType.
	TAdapterItemRequestParamOption (see page 1065)	This is record SiteComp.TAdapterItemRequestParamOption.
	TDispatchPageFlag (see page 1065)	This is record SiteComp.TDispatchPageFlag.
	TQualifyOption (see page 1065)	This is record SiteComp.TQualifyOption.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 1067)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 1067)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 1068)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 1068)	This is type SiteComp.TSessionID.

Variables

Name	Description
ExecuteScript (see page 1069)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 1069)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 1070)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 1070)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 1070)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 1071)	This is variable SiteComp.ScriptDebugger.

10.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 905)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 907)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 910)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 915)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 918)	This is class SiteComp.TScriptObject.

10.1.1 TAbstractDesigntimeWarnings Class

TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.

Class Hierarchy

Pascal

```
TAbstractDesigntimeWarnings = class;
```

C++

```
class TAbstractDesigntimeWarnings;
```

File

SiteComp

Description

TAbstractDesigntimeWarnings is the base class for objects that collect the warning messages for an adapter page producer. At design time, as you add Web items to an adapter page producer, the page producer and the Web items you add to it write warnings to a TAbstractDesigntimeWarnings object describing any problems you can rectify (such as properties that need to be set or components that need to be added.) These warning messages appear in the preview of the emerging HTML form on the HTML view page of the editor.

See Also

TAdapterPageProducer (see page 1076)

GetDesigntimeWarnings

10.1.1.1 TAbstractDesigntimeWarnings Members

The following tables list the members exposed by TAbstractDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 906)	Requests any design time warnings from a specified object.
	AddString (see page 906)	Adds a warning string to the set of warnings that appear at design time.

10.1.1.2 TAbstractDesigntimeWarnings Methods

The methods of the TAbstractDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	AddObject (see page 906)	Requests any design time warnings from a specified object.
	AddString (see page 906)	Adds a warning string to the set of warnings that appear at design time.

10.1.1.2.1 TAbstractDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); virtual; abstract;
```

C++

```
virtual __fastcall AddObject(TObject * AObject) = 0;
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 6353)

AddString (see page 906)

10.1.1.2.2 TAbstractDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); virtual; abstract;
```

C++

```
virtual __fastcall AddString(UnicodeString AWarning) = 0;
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 1076)

AddObject (see page 906)

10.1.2 TAbstractInetFileType Class

Class Hierarchy

Pascal

```
TAbstractInetFileType = class(TObject);
```

C++

```
class TAbstractInetFileType : public TObject;
```

File

SiteComp

Description

This is class SiteComp.TAbstractInetFileType.

10.1.2.1 TAbstractInetFileType Members

The following tables list the members exposed by TAbstractInetFileType.

Protected Methods

	Name	Description
	GetExtCount (see page 908)	This is GetExtCount, a member of class TAbstractInetFileType.
	GetFileExt (see page 908)	This is GetFileExt, a member of class TAbstractInetFileType.
	GetTypeID (see page 909)	This is GetTypeID, a member of class TAbstractInetFileType.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.

	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	FileExt (see page 909)	This is FileExt, a member of class TAbstractInetFileType.
R	FileExtCount (see page 909)	This is FileExtCount, a member of class TAbstractInetFileType.
R	TypeID (see page 910)	This is TypeID, a member of class TAbstractInetFileType.

10.1.2.2 TAbstractInetFileType Methods

The methods of the TAbstractInetFileType class are listed here.

Protected Methods

	Name	Description
A	GetExtCount (see page 908)	This is GetExtCount, a member of class TAbstractInetFileType.
A	GetFileExt (see page 908)	This is GetFileExt, a member of class TAbstractInetFileType.
A	GetTypeID (see page 909)	This is GetTypeID, a member of class TAbstractInetFileType.

10.1.2.2.1 TAbstractInetFileType.GetExtCount Method

Pascal

```
function GetExtCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetExtCount() = 0;
```

Description

This is GetExtCount, a member of class TAbstractInetFileType.

10.1.2.2.2 TAbstractInetFileType.GetFileExt Method

Pascal

```
function GetFileExt(I: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetFileExt(int I) = 0;
```

Description

This is GetFileExt, a member of class TAbstractInetFileType.

10.1.2.2.3 TAbstractInetFileType.GetTypeID Method

Pascal

```
function GetTypeID: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetTypeID() = 0;
```

Description

This is GetTypeID, a member of class TAbstractInetFileType.

10.1.2.3 TAbstractInetFileType Properties

The properties of the TAbstractInetFileType class are listed here.

Public Properties

	Name	Description
	FileExt (see page 909)	This is FileExt, a member of class TAbstractInetFileType.
	FileExtCount (see page 909)	This is FileExtCount, a member of class TAbstractInetFileType.
	TypeID (see page 910)	This is TypeID, a member of class TAbstractInetFileType.

10.1.2.3.1 TAbstractInetFileType.FileExt Property

Pascal

```
property FileExt [I: Integer]: string;
```

C++

```
__property UnicodeString FileExt[int I];
```

Description

This is FileExt, a member of class TAbstractInetFileType.

10.1.2.3.2 TAbstractInetFileType.FileExtCount Property

Pascal

```
property FileExtCount: Integer;
```

C++

```
__property int FileExtCount;
```

Description

This is FileExtCount, a member of class TAbstractInetFileType.

10.1.2.3.3 TAbstractInetFileType.TypeID Property

Pascal

```
property TypeID: string;
```

C++

```
__property UnicodeString TypeID;
```

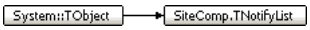
Description

This is TypeID, a member of class TAbstractInetFileType.

10.1.3 TNotifyList Class

TNotifyList is a collection of objects that should be notified of changes.

Class Hierarchy



Pascal

```
TNotifyList = class(TObject);
```

C++

```
class TNotifyList : public TObject;
```

File

SiteComp

Description

TNotifyList maintains a list of objects that have registered interest in change notifications. This class is used for a helper object on components that support the INotifyList (see page 999) interface. When objects use the INotifyList (see page 999) interface to register interest in change notifications, the component adds them to its TNotifyList instance. When those objects use the INotifyList (see page 999) interface to unregister their interest, they are removed from the TNotifyList instance.

When the object that uses TNotifyList changes, it sends a notification to every object in the list. For example, when an adapter changes, it obtains an INotifyAdapterChange (see page 998) interface from every object in the list. It uses this interface to call the object's NotifyAdapterChange method.

See Also

TCustomAdapter

INotifyAdapterChange (see page 998)

INotifyList (see page 999)

10.1.3.1 TNotifyList Members

The following tables list the members exposed by TNotifyList.

Public Methods

	Name	Description
	AddNotify (see page 912)	Adds a new object to the list.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 911)	Creates an initializes an instance of TNotifyList (see page 910).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 912)	Frees an instance of TNotifyList (see page 910).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetCount (see page 913)	Returns the value of the Count (see page 914) property.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 913)	Returns the value of the Items (see page 914) property.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RemoveNotify (see page 913)	Removes an object from the list.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Count (see page 914)	Indicates the number of objects in the list.
R	Items (see page 914)	Lists the objects in this collection.

10.1.3.2 TNotifyList.Create Constructor

Creates an initializes an instance of TNotifyList (see page 910).

Pascal

```
constructor Create;
```

C++

```
__fastcall TNotifyList();
```

Description

WebSnap applications should not need to create instances of TNotifyList (see page 910). These objects are created by the component that uses them to keep track of objects that have registered an interest in change notifications.

See Also

NotifyList (see page 5560)

INotifyList (see page 999)

10.1.3.3 TNotifyList.Destroy Destructor

Frees an instance of TNotifyList (see page 910).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNotifyList();
```

Description

WebSnap applications should not need to free instances of TNotifyList (see page 910), as this is handled by the components that use them. If you must free a TNotifyList (see page 910) instance, use the Free method.

See Also

System::TObject::Free (see page 3382)

10.1.3.4 TNotifyList Methods

The methods of the TNotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 912)	Adds a new object to the list.
	GetCount (see page 913)	Returns the value of the Count (see page 914) property.
	GetItem (see page 913)	Returns the value of the Items (see page 914) property.
	RemoveNotify (see page 913)	Removes an object from the list.

10.1.3.4.1 TNotifyList.AddNotify Method

Adds a new object to the list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the end of the Items (see page 914) property array. This method is called to add objects that

want to be informed of changes to the component that uses this list.

ANotify is the object to add. This object should implement an interface that allows the associated component to inform it of changes. For example, objects in the TNotifyList (see page 910) collection maintained by an adapter support the INotifyAdapterChange (see page 998) interface.

See Also

Items (see page 914)

RemoveNotify (see page 913)

INotifyAdapterChange (see page 998)

10.1.3.4.2 TNotifyList.GetCount Method

Returns the value of the Count (see page 914) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 914) property. It returns the number of objects in the list.

See Also

Count (see page 914)

10.1.3.4.3 TNotifyList.GetItem Method

Returns the value of the Items (see page 914) property.

Pascal

```
function GetItem(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetItem(int Index);
```

Description

GetItem is the read implementation of the Items (see page 914) property.

Index is the zero-offset index of the item to return.

GetItem returns the specified item from the list.

See Also

Items (see page 914)

10.1.3.4.4 TNotifyList.RemoveNotify Method

Removes an object from the list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list. This method is called to remove objects that are no longer interested in changes to the associated component.

ANotify is the object to remove. This object must have been previously added by a call to the AddNotify (see page 912) method.

See Also

Items (see page 914)

AddNotify (see page 912)

10.1.3.5 TNotifyList Properties

The properties of the TNotifyList class are listed here.

Public Properties

	Name	Description
	Count (see page 914)	Indicates the number of objects in the list.
	Items (see page 914)	Lists the objects in this collection.

10.1.3.5.1 TNotifyList.Count Property

Indicates the number of objects in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of objects in the Items (see page 914) property array. Use this value as an upper bound for the index when iterating over all the objects in the list.

See Also

Items (see page 914)

10.1.3.5.2 TNotifyList.Items Property

Lists the objects in this collection.

Pascal

```
property Items [Index: Integer]: TObject;
```

C++

```
__property TObject * Items[int Index];
```

Description

Use Items to reference an object in the list. Each item is an object that supports an interface that allows the associated

component to inform it of changes. For example, objects in the TNotifyList (see page 910) collection maintained by an adapter support the INotifyAdapterChange (see page 998) interface.

Index is the index of the object, where 0 specifies the first item, 1 specifies the second item, and so on. The Count (see page 914) property gives an upper bound for this index.

Note: In Delphi, Items is the default property of TNotifyList (see page 910). That means the property name may be omitted in code.

See Also

Count (see page 914)

AddNotify (see page 912)

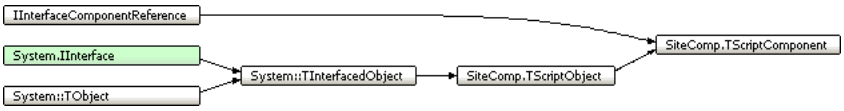
RemoveNotify (see page 913)

INotifyAdapterChange (see page 998)

operator_sb

10.1.4 TScriptComponent Class

Class Hierarchy



Pascal

```
TScriptComponent = class(TScriptObject, IInterfaceComponentReference);
```

C++

```
class TScriptComponent : public TScriptObject, public IInterfaceComponentReference;
```

File

SiteComp

Description

This is class SiteComp.TScriptComponent.

10.1.4.1 TScriptComponent Members

The following tables list the members exposed by TScriptComponent.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.

	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetComponent (see page 917)	Returns a reference to the component associated by the implementing class.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 917)	This is Create, a member of class TScriptComponent.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Obj (see page 917)	This is Obj, a member of class TScriptComponent.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

10.1.4.2 TScriptComponent.Create Constructor

Pascal

```
constructor Create(AObject: TComponent); virtual;
```

C++

```
virtual __fastcall TScriptComponent(TComponent * AObject);
```

Description

This is Create, a member of class TScriptComponent.

10.1.4.3 TScriptComponent Methods

The methods of the TScriptComponent class are listed here.

Protected Methods

	Name	Description
	GetComponent (see page 917)	Returns a reference to the component associated by the implementing class.

10.1.4.3.1 TScriptComponent.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent value that is associated with the implementing class. For example, the TComponent class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent

10.1.4.4 TScriptComponent Properties

The properties of the TScriptComponent class are listed here.

Public Properties

	Name	Description
R	Obj (see page 917)	This is Obj, a member of class TScriptComponent.

10.1.4.4.1 TScriptComponent.Obj Property

Pascal

```
property Obj: TComponent;
```

C++

```
__property TComponent * Obj;
```

Description

This is Obj, a member of class TScriptComponent.

10.1.5 TScriptObject Class

Class Hierarchy

Pascal

```
TScriptObject = class(TInterfacedObject);
```

C++

```
class TScriptObject : public TInterfacedObject;
```

File

SiteComp

Description

This is class SiteComp.TScriptObject.

10.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

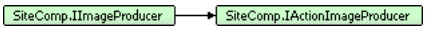
	Name	Description
	IActionImageProducer (see page 920)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 923)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 924)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 925)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 927)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 929)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 934)	This is class SiteComp.IAdapterError.
	IAdapterErrors (see page 937)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 940)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 941)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 943)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 945)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 947)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 951)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 954)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 955)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 956)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 957)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 958)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 959)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 960)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 961)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 962)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 963)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 964)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 965)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 966)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 968)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 969)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 971)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 972)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 973)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 976)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 977)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 978)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 979)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 982)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 983)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 984)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 987)	This is class SiteComp.IInternetExecuteScript.

	IInternetFileTypes (see page 988)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 989)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 990)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 992)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 993)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 994)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.
	IIteratorSupport (see page 996)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 998)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 999)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 1001)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 1006)	This is class SiteComp.IPageResult.
	ISessionsService (see page 1007)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 1010)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 1011)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 1013)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 1014)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 1017)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 1019)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 1022)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 1023)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 1025)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 1026)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 1028)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 1029)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 1030)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 1031)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 1032)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 1034)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 1037)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 1038)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 1039)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 1040)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 1041)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 1042)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 1043)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 1045)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 1049)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 1051)	IWebSessionIDs lists the session IDs (see page 1053) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 1053)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 1055)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 1056)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 1057)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 1059)	This is class SiteComp.IWebValueIsEqual.

10.2.1 IActionImageProducer Interface

IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.

Class Hierarchy



Pascal

```
IActionImageProducer = interface(IImageProducer);
```

C++

```
__interface IActionImageProducer : public IImageProducer;
```

File

SiteComp

Description

IActionImageProducer is the interface used to access the object that supplies images for an adapter action. This interface can supply a single image for the action, or a set of related images that can be used when an HTML control changes its appearance in response to user actions.

IActionImageProducer does not directly supply the HREF for the image or images it represents. Instead, it supplies the interface for another object. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image. This extra level of indirection allows the object that implements IActionImageProducer to work with multiple adapter fields that produce a variety of images.

Note: Method declarations that use IActionImageProducer use the _di_IActionImageProducer type instead. This type is a DelphiInterface (see page 3263) wrapper around the IActionImageProducer interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IActionImageProducer >
_di_IActionImageProducer;
```

See Also

TAdapterActionButton (see page 6286)

TCustomAdapterActionButton (see page 6397)

10.2.1.1 IActionImageProducer Members

The following tables list the members exposed by IActionImageProducer.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 922)	Returns the object that is responsible for providing a specified image from a set of related images.
	GetAdapterImage (see page 984)	This is GetAdapterImage, a member of class IImageProducer.
	GetDisplayStyle (see page 922)	Indicates what type of images this interface can be used to access.

10.2.1.2 IActionImageProducer Methods

The methods of the IActionImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 922)	Returns the object that is responsible for providing a specified image from a set of related images.

	GetDisplayStyle (see page 922)	Indicates what type of images this interface can be used to access.

10.2.1.2.1 IActionImageProducer.GetAdapterEventImage Method

Returns the object that is responsible for providing a specified image from a set of related images.

Pascal

```
function GetAdapterEventImage(const AComponentName: WideString; const AEvent: WideString;
const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterEventImage(const BSTR AComponentName, const BSTR AEvent, const
BSTR ACaption);
```

Description

GetAdapterEventImage returns the object that supplies an HREF for the specified image. That object can supply the HREF in one of two ways:

It can implement the IWebImageHREF (see page 1039) interface.

It can be the object that produces the image in response to request messages. In this case, the adapter dispatcher's EncodeAdapterItemRequestAsHREF method generates the HREF to address this object.

AComponentName is the name of the component that is requesting the image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

AEvent identifies the user event that the image should reflect. This parameter is used when the GetDisplayStyle (see page 922) method returns 'EventImages'. If AEvent is an empty string, GetAdapterEventImage returns the object that supplies the base image. If AEvent is not empty, GetAdapterEventImage returns the object that supplies the version of the image to use in response to the associated user action. Possible values are 'onmouseover', 'onmouseout', and 'onmousedown'.

ACaption is a text label that describes the desired image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

See Also

IWebImageHREF (see page 1039)

EncodeAdapterItemRequestAsHREF (see page 6147)

getadapterimage (see page 984)

GetDisplayStyle (see page 922)

10.2.1.2.2 IActionImageProducer.GetDisplayStyle Method

Indicates what type of images this interface can be used to access.

Pascal

```
function GetDisplayStyle(Sender: TComponent): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(TComponent * Sender);
```

Description

GetDisplayStyle returns either 'Image' or 'EventImages', depending on whether it can supply a single image for an adapter action and caption, or multiple images that reflect various user events.

If GetDisplayStyle returns 'Image', use the GetAdapterImage (see page 984) method to obtain an interface for the image

object. If GetDisplayStyle returns 'EventImages', use the GetAdapterEventImage (see page 922) method to obtain an interface for the image object.

When a TAdapterActionButton component uses this interface for its ImageProducer property, its DisplayType property should be compatible with the display style this method returns. That is, if this method returns 'Image', the adapter action button should have a DisplayType of ctImage or ctDefault. If this method returns 'EventImages', the adapter action button can have a DisplayType of ctImage, ctEventImages, or ctDefault.

Sender is the component that is requesting the image.

See Also

DisplayType (see page 6406)

GetAdapterEventImage (see page 922)

getadapterimage (see page 984)

10.2.2 IAdapterAccess Interface

Class Hierarchy

SiteComp.IAdapterAccess

Pascal

```
IAdapterAccess = interface;
```

C++

```
__interface IAdapterAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterAccess.

10.2.2.1 IAdapterAccess Members

The following tables list the members exposed by IAdapterAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.

10.2.2.2 IAdapterAccess Methods

The methods of the IAdapterAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.

10.2.2.2.1 IAdapterAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterAccess.

10.2.2.2.2 IAdapterAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterAccess.

10.2.2.2.3 IAdapterAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

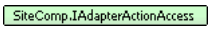
```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterAccess.

10.2.3 IAdapterActionAccess Interface

Class Hierarchy



Pascal

```
IAdapterActionAccess = interface;
```

C++

```
__interface IAdapterActionAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAccess.

10.2.3.1 IAdapterActionAccess Members

The following tables list the members exposed by IAdapterActionAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

10.2.3.2 IAdapterActionAccess Methods

The methods of the IAdapterActionAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

10.2.3.2.1 IAdapterActionAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterActionAccess.

10.2.4 IAdapterActionAttributes Interface

Class Hierarchy

SiteComp.IAdapterActionAttributes

Pascal

```
IAdapterActionAttributes = interface;
```

C++

```
__interface IAdapterActionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAttributes.

10.2.4.1 IAdapterActionAttributes Members

The following tables list the members exposed by IAdapterActionAttributes.

Public Methods

	Name	Description
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.

Public Properties

	Name	Description
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

10.2.4.2 IAdapterActionAttributes Methods

The methods of the IAdapterActionAttributes class are listed here.

Public Methods

	Name	Description
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.

10.2.4.2.1 IAdapterActionAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterActionAttributes.

10.2.4.3 IAdapterActionAttributes Properties

The properties of the IAdapterActionAttributes class are listed here.

Public Properties

	Name	Description
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

10.2.4.3.1 IAdapterActionAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

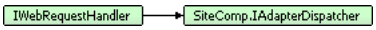
Description

This is Visible, a member of class IAdapterActionAttributes.

10.2.5 IAdapterDispatcher Interface

IAdapterDispatcher is the interface for interacting with the adapter dispatcher.

Class Hierarchy



Pascal

```
IAdapterDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IAdapterDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TAdapterDispatcher component implements IAdapterDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages that represent HTML form submissions and requests for dynamic images that can be resolved using adapter actions and fields. It also encodes the hidden fields and HREF values that are used to make those calls to adapter fields and actions.

Note: In C++, method declarations that use IAdapterDispatcher use the _di_IAdapterDispatcher type instead. This type is a DelphiInterface (see page 3263) wrapper around the IAdapterDispatcher interface:

Note: typedef System::DelphiInterface (see page 3263)< IAdapterDispatcher > _di_IAdapterDispatcher;

See Also

TAdapterDispatcher (see page 6122)

TCustomWebAppComponents (see page 6175)

TWebAPpComponents

10.2.5.1 IAdapterDispatcher Members

The following tables list the members exposed by IAdapterDispatcher.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 928)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 928)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 929)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

10.2.5.2 IAdapterDispatcher Methods

The methods of the IAdapterDispatcher class are listed here.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 928)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 928)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 929)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

10.2.5.2.1 IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsFieldValue to encode the adapter action or image field parameters as they should appear as field values in form submissions. That is, the resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 929)

EncodeAdapterItemRequestAsHREF (see page 928)

10.2.5.2.2 IAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsHREF to encode the adapter action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 929)

EncodeAdapterItemRequestAsFieldValue (see page 928)

10.2.5.2.3 IAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; AParams: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams AParams);
```

Description

Call GetAdapterItemRequestParams to retrieve the adapter action and image field parameters.

AHandler is the interface of the adapter whose action or field requires the parameters.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

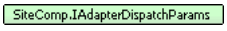
EncodeAdapterItemRequestAsHREF (see page 928)

EncodeAdapterItemRequestAsFieldValue (see page 928)

10.2.6 IAdapterDispatchParams Interface

IAdapterDispatchParams describes adapter action parameters and adapter field values.

Class Hierarchy



Pascal

```
IAdapterDispatchParams = interface;
```

C++

```
__interface IAdapterDispatchParams;
```

File

SiteComp

Description

Use IAdapterDispatchParams to obtain context information that is needed to execute an adapter action or to obtain the value of an image field from an adapter. When server-side script in a WebSnap application executes an adapter action or requests the value of an image field, an object that implements this interface is created to store information about the adapter that can perform the specified task and the parameters that must be passed to the adapter for it to do so. This interface is then passed to the adapter action's HandleRequest method so that it can look up the adapter and the parameters.

Note: In C++, method declarations that use IAdapterDispatchParams use the _di_IAdapterDispatchParams type instead. This type is a DelphiInterface (see page 3263) wrapper around the IAdapterDispatchParams interface:

Note: typedef System::DelphiInterface (see page 3263)< IAdapterDispatchParams > _di_IAdapterDispatchParams;

See Also

IAdapterRequestHandler (see page 954)

TAdapterDispatchParams (see page 6128)

TBaseAdapterAction (see page 5435)

10.2.6.1 IAdapterDispatchParams Members

The following tables list the members exposed by IAdapterDispatchParams.

Public Methods

	Name	Description
	GetHandler (see page 931)	Returns the value of the Handler (see page 932) property.
	GetObjectIdentifier (see page 931)	Returns the value of the ObjectIdentifier (see page 932) property.
	GetParams (see page 931)	Returns the value of the Params (see page 933) property.
	GetRequestIdentifier (see page 931)	Returns the value of the RequestIdentifier (see page 933) property.

Public Properties

	Name	Description
R	Handler (see page 932)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 932)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 933)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 933)	Identifies the type of the request identified by the ObjectIdentifier (see page 932) property.

10.2.6.2 IAdapterDispatchParams Methods

The methods of the IAdapterDispatchParams class are listed here.

Public Methods

	Name	Description
	GetHandler (see page 931)	Returns the value of the Handler (see page 932) property.
	GetObjectIdentifier (see page 931)	Returns the value of the ObjectIdentifier (see page 932) property.
	GetParams (see page 931)	Returns the value of the Params (see page 933) property.

	GetRequestIdentifier (see page 931)	Returns the value of the RequestIdentifier (see page 933) property.

10.2.6.2.1 IAdapterDispatchParams.GetHandler Method

Returns the value of the Handler (see page 932) property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler (see page 932) property.

See Also

Handler (see page 932)

10.2.6.2.2 IAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier (see page 932) property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall UnicodeString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier (see page 932) property.

See Also

ObjectIdentifier (see page 932)

10.2.6.2.3 IAdapterDispatchParams.GetParams Method

Returns the value of the Params (see page 933) property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params (see page 933) property.

See Also

Params (see page 933)

10.2.6.2.4 IAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier (see page 933) property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall UnicodeString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier (see page 933) property.

See Also

RequestIdentifier (see page 933)

10.2.6.3 IAdapterDispatchParams Properties

The properties of the IAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
R	Handler (see page 932)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 932)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 933)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 933)	Identifies the type of the request identified by the ObjectIdentifier (see page 932) property.

10.2.6.3.1 IAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read (see page 3727) Handler to access the adapter that can execute the action or supply the image that is requested.

See Also

ObjectIdentifier (see page 932)

RequestIdentifier (see page 6134)

10.2.6.3.2 IAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property UnicodeString ObjectIdentifier;
```

Description

Read (see page 3727) ObjectIdentifier to determine the name of the requested action or field. The interpretation of this value is the responsibility of the object specified by the Handler (see page 932) parameter.

See Also

Handler (see page 932)

RequestIdentifier (see page 6134)

10.2.6.3.3 IAdapterDispatchParams.Params Property

Indicates the parameter values to use when executing the action or looking up the image.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Read (see page 3727) Params to determine the parameters that the adapter action can use to perform its task. Each string in this string list has the form Name=Value, where Name is the name of a parameter and Value is its value.

The interpretation of the parameter values is up to the adapter specified by the Handler (see page 932) object or to the adapter action object that handles the request.

See Also

Handler (see page 932)

ObjectIdentifier (see page 6133)

10.2.6.3.4 IAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 932) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property UnicodeString RequestIdentifier;
```

Description

Read (see page 3727) RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 932) parameter.

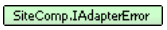
See Also

Handler (see page 932)

ObjectIdentifier (see page 6133)

10.2.7 IAdapterError Interface

Class Hierarchy



Pascal

```
IAdapterError = interface;
```

C++

```
__interface IAdapterError;
```

File

SiteComp

Description

This is class SiteComp.IAdapterError.

10.2.7.1 IAdapterError Members

The following tables list the members exposed by IAdapterError.

Public Methods

	Name	Description
	GetErrorText (see page 934)	Returns the value of the ErrorText (see page 936) property.
	GetField (see page 935)	Returns the value of the Field (see page 936) property.
	GetID (see page 935)	Returns the value of the ID (see page 936) property.

Public Properties

	Name	Description
R	ErrorText (see page 936)	Returns text describing the error.
R	Field (see page 936)	Returns text describing the error.
R	ID (see page 936)	Returns error ID.

10.2.7.2 IAdapterError Methods

The methods of the IAdapterError class are listed here.

Public Methods

	Name	Description
	GetErrorText (see page 934)	Returns the value of the ErrorText (see page 936) property.
	GetField (see page 935)	Returns the value of the Field (see page 936) property.
	GetID (see page 935)	Returns the value of the ID (see page 936) property.

10.2.7.2.1 IAdapterError.GetErrorText Method

Returns the value of the ErrorText (see page 936) property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall UnicodeString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText (see page 936) property.

See Also

ErrorText (see page 936)

10.2.7.2.2 IAdapterError.GetField Method

Returns the value of the Field (see page 936) property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field (see page 936) property.

See Also

Field (see page 936)

10.2.7.2.3 IAdapterError.GetID Method

Returns the value of the ID (see page 936) property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID (see page 936) property.

See Also

ID (see page 936)

10.2.7.3 IAdapterError Properties

The properties of the IAdapterError class are listed here.

Public Properties

	Name	Description
R	ErrorText (see page 936)	Returns text describing the error.
R	Field (see page 936)	Returns text describing the error.

	ID (see page 936)	Returns error ID.

10.2.7.3.1 IAdapterError.ErrorText Property

Returns text describing the error.

Pascal

```
property ErrorText: string;
```

C++

```
__property UnicodeString ErrorText;
```

Description

Examine ErrorText to obtain text describing the error.

See Also

GetErrorText (see page 934)

10.2.7.3.2 IAdapterError.Field Property

Returns text describing the error.

Pascal

```
property Field: TComponent;
```

C++

```
__property TComponent * Field;
```

Description

Examine Field to obtain a reference to the field component involved in the error.

See Also

GetField (see page 935)

10.2.7.3.3 IAdapterError.ID Property

Returns error ID.

Pascal

```
property ID: integer;
```

C++

```
__property int ID;
```

Description

Examine ID to obtain the error ID value.

See Also

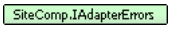
GetID (see page 935)

ID (see page 5167)

10.2.8 IAdapterErrors Interface

IAdapterErrors is implemented by adapter errors objects.

Class Hierarchy



Pascal

```
IAdapterErrors = interface;
```

C++

```
__interface IAdapterErrors;
```

File

SiteComp

Description

Adapter errors objects keep a list of errors detected while executing an adapter action or generating a page using an adapter. Adapter errors objects may be accessed with server-side script using the adapter's Errors (see page 940) property.

See Also

IAdapterError (see page 934)

TCustomAdapter

10.2.8.1 IAdapterErrors Members

The following tables list the members exposed by IAdapterErrors.

Public Methods

	Name	Description
	ClearErrors (see page 938)	Clears the errors list.
	DefineLabel (see page 938)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 938)	Returns a particular error.
	GetErrorCount (see page 939)	Return the number of errors in the errors list.
	GetObjectErrors (see page 939)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 939)	Indicates whether the errors list contains objects associated with a particular adapter field.

Public Properties

	Name	Description
R	ErrorCount (see page 940)	Returns the number of errors in the errors list.
R	Errors (see page 940)	Returns an error from the list.

10.2.8.2 IAdapterErrors Methods

The methods of the IAdapterErrors class are listed here.

Public Methods

	Name	Description
	ClearErrors (see page 938)	Clears the errors list.
	DefineLabel (see page 938)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 938)	Returns a particular error.
	GetErrorCount (see page 939)	Return the number of errors in the errors list.
	GetObjectErrors (see page 939)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 939)	Indicates whether the errors list contains objects associated with a particular adapter field.

10.2.8.2.1 IAdapterErrors.ClearErrors Method

Clears the errors list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clears the errors list.

10.2.8.2.2 IAdapterErrors.DefineLabel Method

Associates a label with all errors associated with a particular adapter field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

DefineLabel is responsible for replacing an label text in error messages associated with a particular adapter field with the given label. DefineLabel is used when the label used on the Web page is not the same as the label generated as part of the error message.

10.2.8.2.3 IAdapterErrors.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 934)

10.2.8.2.4 IAdapterErrors.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

10.2.8.2.5 IAdapterErrors.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const UnicodeString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 939)

10.2.8.2.6 IAdapterErrors.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const UnicodeString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 939)

10.2.8.3 IAdapterErrors Properties

The properties of the IAdapterErrors class are listed here.

Public Properties

	Name	Description
	ErrorCount (see page 940)	Returns the number of errors in the errors list.

	Errors (see page 940)	Returns an error from the list.

10.2.8.3.1 IAdapterErrors.ErrorCount Property

Returns the number of errors in the errors list.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the result of GetErrorCount (see page 939).

See Also

GetErrorCount (see page 939)

10.2.8.3.2 IAdapterErrors.Errors Property

Returns an error from the list.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

I is a zero-based index. Errors returns the result of GetError (see page 938).

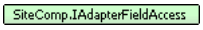
See Also

IAdapterError (see page 934)

GetError (see page 938)

10.2.9 IAdapterFieldAccess Interface

Class Hierarchy



Pascal

```
IAdapterFieldAccess = interface;
```

C++

```
__interface IAdapterFieldAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAccess.

10.2.9.1 IAdapterFieldAccess Members

The following tables list the members exposed by IAdapterFieldAccess.

Public Methods

	Name	Description
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.

10.2.9.2 IAdapterFieldAccess Methods

The methods of the IAdapterFieldAccess class are listed here.

Public Methods

	Name	Description
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.

10.2.9.2.1 IAdapterFieldAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterFieldAccess.

10.2.9.2.2 IAdapterFieldAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterFieldAccess.

10.2.10 IAdapterFieldAttributes Interface

Class Hierarchy

SiteComp.IAdapterFieldAttributes

Pascal

```
IAdapterFieldAttributes = interface;
```

C++

```
__interface IAdapterFieldAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAttributes.

10.2.10.1 IAdapterFieldAttributes Members

The following tables list the members exposed by IAdapterFieldAttributes.

Public Methods

	Name	Description
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.

Public Properties

	Name	Description
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

10.2.10.2 IAdapterFieldAttributes Methods

The methods of the IAdapterFieldAttributes class are listed here.

Public Methods

	Name	Description
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.

10.2.10.2.1 IAdapterFieldAttributes.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IAdapterFieldAttributes.

10.2.10.2.2 IAdapterFieldAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterFieldAttributes.

10.2.10.3 IAdapterFieldAttributes Properties

The properties of the IAdapterFieldAttributes class are listed here.

Public Properties

	Name	Description
	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

10.2.10.3.1 IAdapterFieldAttributes.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IAdapterFieldAttributes.

10.2.10.3.2 IAdapterFieldAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IAdapterFieldAttributes.

10.2.11 IAdapterHiddenField Interface

Class Hierarchy

SiteComp.IAdapterHiddenField

Pascal

```
IAdapterHiddenField = interface;
```

C++

```
__interface IAdapterHiddenField;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenField.

10.2.11.1 IAdapterHiddenField Members

The following tables list the members exposed by IAdapterHiddenField.

Public Methods

	Name	Description
	GetName (see page 944)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 944)	This is GetValue, a member of class IAdapterHiddenField.

Public Properties

	Name	Description
R	Name (see page 945)	This is Name, a member of class IAdapterHiddenField.
R	Value (see page 945)	This is Value, a member of class IAdapterHiddenField.

10.2.11.2 IAdapterHiddenField Methods

The methods of the IAdapterHiddenField class are listed here.

Public Methods

	Name	Description
	GetName (see page 944)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 944)	This is GetValue, a member of class IAdapterHiddenField.

10.2.11.2.1 IAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall UnicodeString GetName();
```

Description

This is GetName, a member of class IAdapterHiddenField.

10.2.11.2.2 IAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall UnicodeString GetValue();
```

Description

This is GetValue, a member of class IAdapterHiddenField.

10.2.11.3 IAdapterHiddenField Properties

The properties of the IAdapterHiddenField class are listed here.

Public Properties

	Name	Description
	Name (see page 945)	This is Name, a member of class IAdapterHiddenField.
	Value (see page 945)	This is Value, a member of class IAdapterHiddenField.

10.2.11.3.1 IAdapterHiddenField.Name Property

Pascal

```
property Name: string;
```

C++

```
__property UnicodeString Name;
```

Description

This is Name, a member of class IAdapterHiddenField.

10.2.11.3.2 IAdapterHiddenField.Value Property

Pascal

```
property Value: string;
```

C++

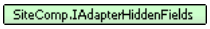
```
__property UnicodeString Value;
```

Description

This is Value, a member of class IAdapterHiddenField.

10.2.12 IAdapterHiddenFields Interface

Class Hierarchy



Pascal

```
IAdapterHiddenFields = interface;
```

C++

```
__interface IAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenFields.

10.2.12.1 IAdapterHiddenFields Members

The following tables list the members exposed by IAdapterHiddenFields.

Public Methods

	Name	Description
	GetField (see page 946)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 946)	This is GetFieldCount, a member of class IAdapterHiddenFields.

Public Properties

	Name	Description
R	FieldCount (see page 947)	This is FieldCount, a member of class IAdapterHiddenFields.
R	Fields (see page 947)	This is Fields, a member of class IAdapterHiddenFields.

10.2.12.2 IAdapterHiddenFields Methods

The methods of the IAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetField (see page 946)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 946)	This is GetFieldCount, a member of class IAdapterHiddenFields.

10.2.12.2.1 IAdapterHiddenFields.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class IAdapterHiddenFields.

10.2.12.2.2 IAdapterHiddenFields.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class IAdapterHiddenFields.

10.2.12.3 IAdapterHiddenFields Properties

The properties of the IAdapterHiddenFields class are listed here.

Public Properties

	Name	Description
	FieldCount (see page 947)	This is FieldCount, a member of class IAdapterHiddenFields.
	Fields (see page 947)	This is Fields, a member of class IAdapterHiddenFields.

10.2.12.3.1 IAdapterHiddenFields.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class IAdapterHiddenFields.

10.2.12.3.2 IAdapterHiddenFields.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

Description

This is Fields, a member of class IAdapterHiddenFields.

10.2.13 IAdapterItemRequestParams Interface

IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.

Class Hierarchy

SiteComp.IAdapterItemRequestParams

Pascal

```
IAdapterItemRequestParams = interface;
```

C++

```
__interface IAdapterItemRequestParams;
```

File

SiteComp

Description

Use IAdapterItemRequestParams to read the parameters that accompany requests to an adapter field or action. The parameters, which can be accessed through the ParamValues (see page 951) property, are a set of 'Name=Value' strings. In addition to providing access to the parameters, IAdapterItemRequestParams has two properties, SuccessPage (see page 951) and FailurePage (see page 950), which identify the Web pages that should be returned to the caller when the adapter request succeeds or fails.

Note: In C++, method declarations that use IAdapterItemRequestParams use the _di_IAdapterItemRequestParams type instead. This type is a DelphiInterface (see page 3263) wrapper around the IAdapterItemRequestParams interface:

Note: typedef System::DelphiInterface (see page 3263)< IAdapterItemRequestParams > _di_IAdapterItemRequestParams;

See Also

TPageDispatcher (see page 6210)

TCustomWebAppComponents (see page 6175)

TWebAPpComponents

10.2.13.1 IAdapterItemRequestParams Members

The following tables list the members exposed by IAdapterItemRequestParams.

Public Methods

	Name	Description
	GetFailurePage (see page 948)	Returns the value of the FailurePage (see page 950) property.
	GetParamValues (see page 949)	Returns the value of the ParamValues (see page 951) property.
	GetSuccessPage (see page 949)	Returns the value of the SuccessPage (see page 951) property.
	SetFailurePage (see page 949)	Sets the value of the FailurePage (see page 950) property.
	SetSuccessPage (see page 950)	Sets the value of the SuccessPage (see page 951) property.

Public Properties

	Name	Description
	FailurePage (see page 950)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 951)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 951)	Indicates the Web page module that the application should return if the adapter request succeeds.

10.2.13.2 IAdapterItemRequestParams Methods

The methods of the IAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 948)	Returns the value of the FailurePage (see page 950) property.
	GetParamValues (see page 949)	Returns the value of the ParamValues (see page 951) property.
	GetSuccessPage (see page 949)	Returns the value of the SuccessPage (see page 951) property.
	SetFailurePage (see page 949)	Sets the value of the FailurePage (see page 950) property.
	SetSuccessPage (see page 950)	Sets the value of the SuccessPage (see page 951) property.

10.2.13.2.1 IAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage (see page 950) property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage (see page 950) property.

See Also

FailurePage (see page 950)

10.2.13.2.2 IAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues (see page 951) property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues (see page 951) property.

See Also

ParamValues (see page 951)

10.2.13.2.3 IAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage (see page 951) property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage (see page 951) property.

See Also

SuccessPage (see page 951)

10.2.13.2.4 IAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage (see page 950) property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const UnicodeString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage (see page 950) property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 950)

10.2.13.2.5 IAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage (see page 951) property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const UnicodeString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage (see page 951) property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 951)

10.2.13.3 IAdapterItemRequestParams Properties

The properties of the IAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 950)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 951)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 951)	Indicates the Web page module that the application should return if the adapter request succeeds.

10.2.13.3.1 IAdapterItemRequestParams.FailurePage Property

Indicates the Web page module that the application should return if the adapter request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 951)

10.2.13.3.2 IAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names

Values

10.2.13.3.3 IAdapterItemRequestParams.SuccessPage Property

Indicates the Web page module that the application should return if the adapter request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

```
__property UnicodeString SuccessPage;
```

Description

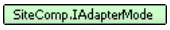
SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

See Also

SuccessPage

10.2.14 IAdapterMode Interface

Class Hierarchy



Pascal

```
IAdapterMode = interface;
```

C++

```
__interface IAdapterMode;
```

File

SiteComp

Description

This is class SiteComp.IAdapterMode.

10.2.14.1 IAdapterMode Members

The following tables list the members exposed by IAdapterMode.

Public Methods

	Name	Description
	GetAdapterMode (see page 952)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 952)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 953)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 953)	This is SetAdapterMode, a member of class IAdapterMode.

Public Properties

	Name	Description
	AdapterMode (see page 953)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 953)	This is NeedAdapterMode, a member of class IAdapterMode.

10.2.14.2 IAdapterMode Methods

The methods of the IAdapterMode class are listed here.

Public Methods

	Name	Description
	GetAdapterMode (see page 952)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 952)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 953)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 953)	This is SetAdapterMode, a member of class IAdapterMode.

10.2.14.2.1 IAdapterMode.GetAdapterMode Method

Pascal

```
function GetAdapterMode: string;
```

C++

```
__fastcall UnicodeString GetAdapterMode();
```

Description

This is GetAdapterMode, a member of class IAdapterMode.

10.2.14.2.2 IAdapterMode.GetNeedAdapterMode Method

Pascal

```
function GetNeedAdapterMode: Boolean;
```

C++

```
__fastcall Boolean GetNeedAdapterMode();
```

Description

This is GetNeedAdapterMode, a member of class IAdapterMode.

10.2.14.2.3 IAdapterMode.RestoreDefaultAdapterMode Method

Pascal

```
procedure RestoreDefaultAdapterMode;
```

C++

```
__fastcall RestoreDefaultAdapterMode();
```

Description

This is RestoreDefaultAdapterMode, a member of class IAdapterMode.

10.2.14.2.4 IAdapterMode.SetAdapterMode Method

Pascal

```
procedure SetAdapterMode(const AValue: string);
```

C++

```
__fastcall SetAdapterMode(const UnicodeString AValue);
```

Description

This is SetAdapterMode, a member of class IAdapterMode.

10.2.14.3 IAdapterMode Properties

The properties of the IAdapterMode class are listed here.

Public Properties

	Name	Description
	AdapterMode (see page 953)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 953)	This is NeedAdapterMode, a member of class IAdapterMode.

10.2.14.3.1 IAdapterMode.AdapterMode Property

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

This is AdapterMode, a member of class IAdapterMode.

10.2.14.3.2 IAdapterMode.NeedAdapterMode Property

Pascal

```
property NeedAdapterMode: Boolean;
```

C++

```
__property Boolean NeedAdapterMode;
```

Description

This is NeedAdapterMode, a member of class IAdapterMode.

10.2.15 IAdapterRequestHandler Interface

IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.

Class Hierarchy

SiteComp.IAdapterRequestHandler

Pascal

```
IAdapterRequestHandler = interface;
```

C++

```
__interface IAdapterRequestHandler;
```

File

SiteComp

Description

IAdapterRequestHandler is implemented by classes that execute an adapter action or look up the value of an image field.

See Also

TBaseAdapterAction (see page 5435)

TCustomAdapterImageField (see page 5609)

10.2.15.1 IAdapterRequestHandler Members

The following tables list the members exposed by IAdapterRequestHandler.

Public Methods

	Name	Description
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.

10.2.15.2 IAdapterRequestHandler Methods

The methods of the IAdapterRequestHandler class are listed here.

Public Methods

	Name	Description
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.

10.2.15.2.1 IAdapterRequestHandler.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext creates a new request context based on the specified parameters.

See Also

AdapterRequest (see page 5946)

TAdapterDispatcher (see page 6122)

CreateRequestContext (see page 5440)

CreateRequestContext (see page 5614)

10.2.15.2.2 IAdapterRequestHandler.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response.

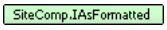
See Also

HandleRequest (see page 5451)

HandleRequest (see page 5616)

10.2.16 IAsFormatted Interface

Class Hierarchy



Pascal

```
IAsFormatted = interface;
```

C++

```
__interface IAsFormatted;
```

File

SiteComp

Description

This is class SiteComp.IAsFormatted.

10.2.16.1 IAsFormatted Members

The following tables list the members exposed by IAsFormatted.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.

10.2.16.2 IAsFormatted Methods

The methods of the IAsFormatted class are listed here.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.

10.2.16.2.1 IAsFormatted.AsFormatted Method

Pascal

```
function AsFormatted: string;
```

C++

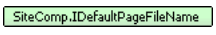
```
__fastcall UnicodeString AsFormatted();
```

Description

This is AsFormatted, a member of class IAsFormatted.

10.2.17 IDefaultPageFileName Interface

Class Hierarchy



Pascal

```
IDefaultPageFileName = interface;
```

C++

```
__interface IDefaultPageFileName;
```

File

SiteComp

Description

This is class SiteComp.IDefaultPageFileName.

10.2.17.1 IDefaultPageFileName Members

The following tables list the members exposed by IDefaultPageFileName.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

10.2.17.2 IDefaultPageFileName Methods

The methods of the IDefaultPageFileName class are listed here.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

10.2.17.2.1 IDefaultPageFileName.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const APageFile: string);
```

C++

```
__fastcall SetDefaultPageFileName(const UnicodeString APageFile);
```

Description

This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

10.2.18 IFieldValuesAdapter Interface

Class Hierarchy

Pascal

```
IFieldValuesAdapter = interface;
```

C++

```
__interface IFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IFieldValuesAdapter.

10.2.18.1 IFieldValuesAdapter Members

The following tables list the members exposed by IFieldValuesAdapter.

Public Methods

	Name	Description
	GetField (see page 958)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 958)	This is GetValueIndex, a member of class IFieldValuesAdapter.

10.2.18.2 IFieldValuesAdapter Methods

The methods of the IFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetField (see page 958)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 958)	This is GetValueIndex, a member of class IFieldValuesAdapter.

10.2.18.2.1 IFieldValuesAdapter.GetField Method

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

This is GetField, a member of class IFieldValuesAdapter.

10.2.18.2.2 IFieldValuesAdapter.GetValueIndex Method

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

This is GetValueIndex, a member of class IFieldValuesAdapter.

10.2.19 IFindIncludeFile Interface

Class Hierarchy

SiteComp.IFindIncludeFile

Pascal

```
IFindIncludeFile = interface;
```

C++

```
__interface IFindIncludeFile;
```

File

SiteComp

Description

This is class SiteComp.IFindIncludeFile.

10.2.19.1 IFindIncludeFile Members

The following tables list the members exposed by IFindIncludeFile.

Public Methods

	Name	Description
	FindIncludeFile (see page 959)	This is FindIncludeFile, a member of class IFindIncludeFile.

10.2.19.2 IFindIncludeFile Methods

The methods of the IFindIncludeFile class are listed here.

Public Methods

	Name	Description
	FindIncludeFile (see page 959)	This is FindIncludeFile, a member of class IFindIncludeFile.

10.2.19.2.1 IFindIncludeFile.FindIncludeFile Method

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const UnicodeString AFileName,
UnicodeString AResult);
```

Description

This is FindIncludeFile, a member of class IFindIncludeFile.

10.2.20 IGetAdapterActions Interface

Class Hierarchy

SiteComp.IGetAdapterActions

Pascal

```
IGetAdapterActions = interface;
```

C++

```
__interface IGetAdapterActions;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterActions.

10.2.20.1 IGetAdapterActions Members

The following tables list the members exposed by IGetAdapterActions.

Public Methods

	Name	Description
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.

10.2.20.2 IGetAdapterActions Methods

The methods of the IGetAdapterActions class are listed here.

Public Methods

	Name	Description
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.

10.2.20.2.1 IGetAdapterActions.GetAdapterActions Method

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

This is GetAdapterActions, a member of class IGetAdapterActions.

10.2.21 IGetAdapterDisplayCharacteristics Interface

Class Hierarchy

SiteComp.IGetAdapterDisplayCharacteristics

Pascal

```
IGetAdapterDisplayCharacteristics = interface;
```

C++

```
__interface IGetAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterDisplayCharacteristics.

10.2.21.1 IGetAdapterDisplayCharacteristics Members

The following tables list the members exposed by IGetAdapterDisplayCharacteristics.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

10.2.21.2 IGetAdapterDisplayCharacteristics Methods

The methods of the IGetAdapterDisplayCharacteristics class are listed here.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

10.2.21.2.1 IGetAdapterDisplayCharacteristics.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

10.2.22 IGetAdapterErrors Interface

Class Hierarchy

SiteComp.IGetAdapterErrors

Pascal

```
IGetAdapterErrors = interface;
```

C++

```
__interface IGetAdapterErrors;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterErrors.

10.2.22.1 IGetAdapterErrors Members

The following tables list the members exposed by IGetAdapterErrors.

Public Methods

	Name	Description
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

10.2.22.2 IGetAdapterErrors Methods

The methods of the IGetAdapterErrors class are listed here.

Public Methods

	Name	Description
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

10.2.22.2.1 IGetAdapterErrors.GetAdapterErrors Method

Pascal

```
function GetAdapterErrors: TObject;
```

C++

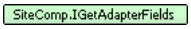
```
__fastcall TObject * GetAdapterErrors();
```

Description

This is GetAdapterErrors, a member of class IGetAdapterErrors.

10.2.23 IGetAdapterFields Interface

Class Hierarchy



Pascal

```
IGetAdapterFields = interface;
```

C++

```
__interface IGetAdapterFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterFields.

10.2.23.1 IGetAdapterFields Members

The following tables list the members exposed by IGetAdapterFields.

Public Methods

	Name	Description
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.

10.2.23.2 IGetAdapterFields Methods

The methods of the IGetAdapterFields class are listed here.

Public Methods

	Name	Description
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.

10.2.23.2.1 IGetAdapterFields.GetAdapterFields Method

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

This is GetAdapterFields, a member of class IGetAdapterFields.

10.2.24 IGetAdapterHiddenFields Interface

Class Hierarchy

SiteComp.IGetAdapterHiddenFields

Pascal

```
IGetAdapterHiddenFields = interface;
```

C++

```
__interface IGetAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterHiddenFields.

10.2.24.1 IGetAdapterHiddenFields Members

The following tables list the members exposed by IGetAdapterHiddenFields.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

10.2.24.2 IGetAdapterHiddenFields Methods

The methods of the IGetAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

10.2.24.2.1 IGetAdapterHiddenFields.GetAdapterHiddenFields Method

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.

10.2.24.2.2 IGetAdapterHiddenFields.GetAdapterHiddenRecordFields Method

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

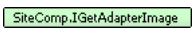
```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

10.2.25 IGetAdapterImage Interface

Class Hierarchy



Pascal

```
IGetAdapterImage = interface;
```

C++

```
__interface IGetAdapterImage;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterImage.

10.2.25.1 IGetAdapterImage Members

The following tables list the members exposed by IGetAdapterImage.

Public Methods

	Name	Description
	GetAdapterImage (see page 965)	This is GetAdapterImage, a member of class IGetAdapterImage.

10.2.25.2 IGetAdapterImage Methods

The methods of the IGetAdapterImage class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 965)	This is GetAdapterImage, a member of class IGetAdapterImage.

10.2.25.2.1 IGetAdapterImage.GetAdapterImage Method

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

This is GetAdapterImage, a member of class IGetAdapterImage.

10.2.26 IGetAdapterItemRequestParams Interface

Class Hierarchy

SiteComp.IGetAdapterItemRequestParams

Pascal

```
IGetAdapterItemRequestParams = interface;
```

C++

```
__interface IGetAdapterItemRequestParams;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterItemRequestParams.

10.2.26.1 IGetAdapterItemRequestParams Members

The following tables list the members exposed by IGetAdapterItemRequestParams.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

10.2.26.2 IGetAdapterItemRequestParams Methods

The methods of the IGetAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

10.2.26.2.1 IGetAdapterItemRequestParams.GetAdapterItemRequestParams Method

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

10.2.27 IGetAdapterRequestDefaultResponse Interface

Class Hierarchy

SiteComp.IGetAdapterRequestDefaultResponse

Pascal

```
IGetAdapterRequestDefaultResponse = interface;
```

C++

```
__interface IGetAdapterRequestDefaultResponse;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterRequestDefaultResponse.

10.2.27.1 IGetAdapterRequestDefaultResponse Members

The following tables list the members exposed by IGetAdapterRequestDefaultResponse.

Public Methods

	Name	Description
	GetFailurePage (see page 967)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 967)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

Public Properties

	Name	Description
	FailurePage (see page 968)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
	SuccessPage (see page 968)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.2 IGetAdapterRequestDefaultResponse Methods

The methods of the IGetAdapterRequestDefaultResponse class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 967)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 967)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.2.1 IGetAdapterRequestDefaultResponse.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.2.2 IGetAdapterRequestDefaultResponse.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.3 IGetAdapterRequestDefaultResponse Properties

The properties of the IGetAdapterRequestDefaultResponse class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 968)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
	SuccessPage (see page 968)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.3.1 IGetAdapterRequestDefaultResponse.FailurePage Property

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.

10.2.27.3.2 IGetAdapterRequestDefaultResponse.SuccessPage Property

Pascal

```
property SuccessPage: string;
```

C++

```
__property UnicodeString SuccessPage;
```

Description

This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.2.28 IGetAdapterValuesList Interface

Class Hierarchy

SiteComp.IGetAdapterValuesList

Pascal

```
IGetAdapterValuesList = interface;
```

C++

```
__interface IGetAdapterValuesList;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterValuesList.

10.2.28.1 IGetAdapterValuesList Members

The following tables list the members exposed by IGetAdapterValuesList.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

10.2.28.2 IGetAdapterValuesList Methods

The methods of the IGetAdapterValuesList class are listed here.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

10.2.28.2.1 IGetAdapterValuesList.GetAdapterValuesList Method

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

10.2.29 IGetDefaultAction Interface

Class Hierarchy

SiteComp.IGetDefaultAction

Pascal

```
IGetDefaultAction = interface;
```

C++

```
__interface IGetDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IGetDefaultAction.

10.2.29.1 IGetDefaultAction Members

The following tables list the members exposed by IGetDefaultAction.

Public Methods

	Name	Description
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.

Public Properties

	Name	Description
	DefaultAction (see page 970)	This is DefaultAction, a member of class IGetDefaultAction.

10.2.29.2 IGetDefaultAction Methods

The methods of the IGetDefaultAction class are listed here.

Public Methods

	Name	Description
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.

10.2.29.2.1 IGetDefaultAction.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class IGetDefaultAction.

10.2.29.3 IGetDefaultAction Properties

The properties of the IGetDefaultAction class are listed here.

Public Properties

	Name	Description
	DefaultAction (see page 970)	This is DefaultAction, a member of class IGetDefaultAction.

10.2.29.3.1 IGetDefaultAction.DefaultAction Property

Pascal

```
property DefaultAction: TComponent;
```

C++

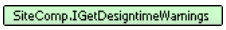
```
__property TComponent * DefaultAction;
```

Description

This is DefaultAction, a member of class IGetDefaultAction.

10.2.30 IGetDesigntimeWarnings Interface

Class Hierarchy



Pascal

```
IGetDesigntimeWarnings = interface;
```

C++

```
__interface IGetDesigntimeWarnings;
```

File

SiteComp

Description

This is class SiteComp.IGetDesigntimeWarnings.

10.2.30.1 IGetDesigntimeWarnings Members

The following tables list the members exposed by IGetDesigntimeWarnings.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

10.2.30.2 IGetDesigntimeWarnings Methods

The methods of the IGetDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

10.2.30.2.1 IGetDesigntimeWarnings.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

10.2.31 IGetFieldValuesAdapter Interface

Class Hierarchy

Pascal

```
IGetFieldValuesAdapter = interface;
```

C++

```
__interface IGetFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IGetFieldValuesAdapter.

10.2.31.1 IGetFieldValuesAdapter Members

The following tables list the members exposed by IGetFieldValuesAdapter.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

10.2.31.2 IGetFieldValuesAdapter Methods

The methods of the IGetFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

10.2.31.2.1 IGetFieldValuesAdapter.GetFieldValuesAdapter Method

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

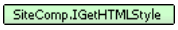
Description

This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

10.2.32 IGetHTMLStyle Interface

IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.

Class Hierarchy



Pascal

```
IGetHTMLStyle = interface;
```

C++

```
__interface IGetHTMLStyle;
```

File

SiteComp

Description

IGetHTMLStyle methods are typically called while generating HTML to display adapter fields and adapter actions. An adapter field implements this interface to indicate, for example, whether it's value should be displayed with an HTML input field or a select list. An adapter action implements this interface to indicate, for example, whether it's value should be displayed with an HTML button or hyperlink.

TAdapterPageProducer child components may call this interface when generating HTML. For example, TCustomAdapterDisplayField calls GetViewMode (see page 975) when the TCustomAdapterDisplayField's ViewMode property is set to vmDefault. TCustomAdapterActionButton calls GetDisplayStyle (see page 974) when its DisplayType property is ctDefault.

See Also

TAdapterNamedDisplayField (see page 5331)

TCustomAdapterDisplayColumn (see page 6428)

TCustomAdapterDisplayField (see page 6434)

10.2.32.1 IGetHTMLStyle Members

The following tables list the members exposed by IGetHTMLStyle.

Public Methods

	Name	Description
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.

10.2.32.2 IGetHTMLStyle Methods

The methods of the IGetHTMLStyle class are listed here.

Public Methods

	Name	Description
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.

10.2.32.2.1 IGetHTMLStyle.GetDisplayStyle Method

Returns the preferred display style for an adapter field or adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Edit).

Adapter fields typically return one of the following values:

Value	Meaning
Text	Display the field as read-only text
Image	Display the field as an image
List	Display the field as a list of read only values
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

Adapter actions typically return one of the following values:

Value	Meaning
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

User the server-side script DisplayStyle property to get these values from adapter fields and adapter actions.

See Also

DisplayType (see page 6439)

DisplayType (see page 6406)

10.2.32.2.2 IGetHTMLStyle.GetInputStyle Method

Returns the preferred editing style for an adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Browse).

Adapter fields typically return one of the following values:

Value	Meaning
TextInput	Edit the adapter field using a single line input field.
PasswordInput	Edit the adapter field as a password field.
Select	Edit the adapter field using a single selection list.
SelectMultiple	Edit the adapter field using a multiple selection list.
Radio	Edit the adapter field using radio buttons.
CheckBox	Edit the adapter field using check boxes.
TextArea	Edit the adapter field using a multi-line input field.
File	Edit the adapter field using a file upload input field.

Use the server-side script InputStyle property to get these values from adapter fields.

See Also

InputType (see page 6441)

10.2.32.2.3 IGetHTMLStyle.GetViewMode Method

Returns the preferred mode of an adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Insert (see page 3698)).

Adapter fields typically return one of the following values:

Value	Meaning
Input (see page 3895)	Display the adapter field as an input field, by default use the style returned by the GetInputStyle (see page 974) method.
Display	Display the adapter field as a read-only field, by default use the style returned by GetDisplayStyle (see page 974).
ToggleOnAccess	Display as an input field in the end-user has modify rights, otherwise displays as a read-only field.

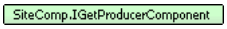
Use the server-side script ViewMode property to get these values from adapter fields.

See Also

ViewMode (see page 6442)

10.2.33 IGetProducerComponent Interface

Class Hierarchy



Pascal

```
IGetProducerComponent = interface;
```

C++

```
__interface IGetProducerComponent;
```

File

SiteComp

Description

This is class SiteComp.IGetProducerComponent.

10.2.33.1 IGetProducerComponent Members

The following tables list the members exposed by IGetProducerComponent.

Public Methods

	Name	Description
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.

10.2.33.2 IGetProducerComponent Methods

The methods of the IGetProducerComponent class are listed here.

Public Methods

	Name	Description
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.

10.2.33.2.1 IGetProducerComponent.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class IGetProducerComponent.

10.2.34 IGetScriptObject Interface

Class Hierarchy

SiteComp.IGetScriptObject

Pascal

```
IGetScriptObject = interface;
```

C++

```
__interface IGetScriptObject;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObject.

10.2.34.1 IGetScriptObject Members

The following tables list the members exposed by IGetScriptObject.

Public Methods

	Name	Description
	GetScriptObject (see page 977)	This is GetScriptObject, a member of class IGetScriptObject.

10.2.34.2 IGetScriptObject Methods

The methods of the IGetScriptObject class are listed here.

Public Methods

	Name	Description
	GetScriptObject (see page 977)	This is GetScriptObject, a member of class IGetScriptObject.

10.2.34.2.1 IGetScriptObject.GetScriptObject Method

Pascal

```
function GetScriptObject: IInterface; deprecated;
```

C++

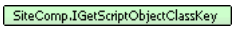
```
__fastcall IInterface GetScriptObject();
```

Description

This is GetScriptObject, a member of class IGetScriptObject.

10.2.35 IGetScriptObjectClassKey Interface

Class Hierarchy



Pascal

```
IGetScriptObjectClassKey = interface;
```

C++

```
__interface IGetScriptObjectClassKey;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObjectClassKey.

10.2.35.1 IGetScriptObjectClassKey Members

The following tables list the members exposed by IGetScriptObjectClassKey.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 978)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

10.2.35.2 IGetScriptObjectClassKey Methods

The methods of the IGetScriptObjectClassKey class are listed here.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 978)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

10.2.35.2.1 IGetScriptObjectClassKey.GetScriptObjectClassKey Method

Pascal

```
function GetScriptObjectClassKey: TClass; deprecated;
```

C++

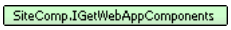
```
__fastcall TClass GetScriptObjectClassKey();
```

Description

This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

10.2.36 IGetWebAppComponents Interface

Class Hierarchy



Pascal

```
IGetWebAppComponents = interface;
```

C++

```
__interface IGetWebAppComponents;
```

File

SiteComp

Description

This is class SiteComp.IGetWebAppComponents.

10.2.36.1 IGetWebAppComponents Members

The following tables list the members exposed by IGetWebAppComponents.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 980)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 980)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 980)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 980)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 980)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 981)	This is GetWebUserList, a member of class IGetWebAppComponents.

Public Properties

	Name	Description
R	WebApplicationInfo (see page 981)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 981)	This is WebEndUser, a member of class IGetWebAppComponents.

10.2.36.2 IGetWebAppComponents Methods

The methods of the IGetWebAppComponents class are listed here.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 980)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 980)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 980)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 980)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 980)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 981)	This is GetWebUserList, a member of class IGetWebAppComponents.

10.2.36.2.1 IGetWebAppComponents.GetAdapterDispatcher Method

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

This is GetAdapterDispatcher, a member of class IGetWebAppComponents.

10.2.36.2.2 IGetWebAppComponents.GetLocateFileService Method

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

This is GetLocateFileService, a member of class IGetWebAppComponents.

10.2.36.2.3 IGetWebAppComponents.GetPageDispatcher Method

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

This is GetPageDispatcher, a member of class IGetWebAppComponents.

10.2.36.2.4 IGetWebAppComponents.GetWebApplicationInfo Method

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

This is GetWebApplicationInfo, a member of class IGetWebAppComponents.

10.2.36.2.5 IGetWebAppComponents.GetWebEndUser Method

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

This is GetWebEndUser, a member of class IGetWebAppComponents.

10.2.36.2.6 IGetWebAppComponents.GetWebUserList Method

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

This is GetWebUserList, a member of class IGetWebAppComponents.

10.2.36.3 IGetWebAppComponents Properties

The properties of the IGetWebAppComponents class are listed here.

Public Properties

	Name	Description
	WebApplicationInfo (see page 981)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
	WebEndUser (see page 981)	This is WebEndUser, a member of class IGetWebAppComponents.

10.2.36.3.1 IGetWebAppComponents.WebApplicationInfo Property

Pascal

```
property WebApplicationInfo: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo WebApplicationInfo;
```

Description

This is WebApplicationInfo, a member of class IGetWebAppComponents.

10.2.36.3.2 IGetWebAppComponents.WebEndUser Property

Pascal

```
property WebEndUser: IWebEndUser;
```

C++

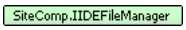
```
__property IWebEndUser WebEndUser;
```

Description

This is WebEndUser, a member of class IGetWebAppComponents.

10.2.37 IIDEFileManager Interface

Class Hierarchy



Pascal

```
IIDEFileManager = interface;
```

C++

```
__interface IIDEFileManager;
```

File

SiteComp

Description

This is class SiteComp.IIDEFileManager.

10.2.37.1 IIDEFileManager Members

The following tables list the members exposed by IIDEFileManager.

Public Methods

	Name	Description
	SearchFile (see page 982)	This is SearchFile, a member of class IIDEFileManager.

10.2.37.2 IIDEFileManager Methods

The methods of the IIDEFileManager class are listed here.

Public Methods

	Name	Description
	SearchFile (see page 982)	This is SearchFile, a member of class IIDEFileManager.

10.2.37.2.1 IIDEFileManager.SearchFile Method

Pascal

```
function SearchFile(const AFileName: string): string;
```

C++

```
__fastcall UnicodeString SearchFile(const UnicodeString AFileName);
```

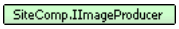
Description

This is SearchFile, a member of class IIDEFileManager.

10.2.38 IImageProducer Interface

IImageProducer is the interface for accessing the image that an adapter component supplies.

Class Hierarchy



Pascal

```
IImageProducer = interface;
```

C++

```
__interface IImageProducer;
```

File

SiteComp

Description

IImageProducer is the interface used to access the object that supplies an image to server-side script in a WebSnap application. It defines a single method, GetAdapterImage (see page 984), which returns the interface of the object that is responsible for providing the image. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image.

Note that IImageProducer introduces an extra level of indirection when fetching images. IImageProducer is not necessarily the interface of the object that provides the image, it merely retrieves that interface. The reason for this indirection is because the object that implements IImageProducer may work with multiple adapter fields that produce a variety of images, depending on the supplied caption.

Note: In C++, method declarations that use IImageProducer use the _di_IImageProducer type instead. This type is a DelphiInterface (see page 3263) wrapper around the IImageProducer interface:

Note: typedef System::DelphiInterface (see page 3263)< IImageProducer > _di_IImageProducer;

See Also

IActionImageProducer (see page 920)

TDataSetAdapterImageField

TAdapterImageField (see page 5297)

10.2.38.1 IImageProducer Members

The following tables list the members exposed by IImageProducer.

Public Methods

	Name	Description
	GetAdapterImage (see page 984)	This is GetAdapterImage, a member of class IImageProducer.

10.2.38.2 IImageProducer Methods

The methods of the IImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 984)	This is GetAdapterImage, a member of class IImageProducer.

10.2.38.2.1 IImageProducer.GetAdapterImage Method

Pascal

```
function GetAdapterImage(const AComponentName: string; const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterImage(const UnicodeString AComponentName, const BSTR ACaption);
```

Description

This is GetAdapterImage, a member of class IImageProducer.

10.2.39 IInternetEnvOptions Interface

Class Hierarchy

SiteComp.IInternetEnvOptions

Pascal

```
IInternetEnvOptions = interface;
```

C++

```
__interface IInternetEnvOptions;
```

File

SiteComp

Description

This is class SiteComp.IInternetEnvOptions.

10.2.39.1 IInternetEnvOptions Members

The following tables list the members exposed by IInternetEnvOptions.

Public Methods

	Name	Description
	GetDisableDebugging (see page 985)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 985)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 985)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 986)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

Public Properties

	Name	Description
	DisableDebugger (see page 986)	This is DisableDebugger, a member of class IInternetEnvOptions.
	HTMLExt (see page 986)	This is HTMLExt, a member of class IInternetEnvOptions.
	SampleImageFile (see page 986)	This is SampleImageFile, a member of class IInternetEnvOptions.
	ScriptTimeoutSecs (see page 987)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

10.2.39.2 IInternetEnvOptions Methods

The methods of the IInternetEnvOptions class are listed here.

Public Methods

	Name	Description
	GetDisableDebugging (see page 985)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 985)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 985)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 986)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

10.2.39.2.1 IInternetEnvOptions.GetDisableDebugging Method

Pascal

```
function GetDisableDebugging: Boolean;
```

C++

```
__fastcall Boolean GetDisableDebugging();
```

Description

This is GetDisableDebugging, a member of class IInternetEnvOptions.

10.2.39.2.2 IInternetEnvOptions.GetHTMLExt Method

Pascal

```
function GetHTMLExt: string;
```

C++

```
__fastcall UnicodeString GetHTMLExt();
```

Description

This is GetHTMLExt, a member of class IInternetEnvOptions.

10.2.39.2.3 IInternetEnvOptions.GetSampleImageFile Method

Pascal

```
function GetSampleImageFile: string;
```

C++

```
__fastcall UnicodeString GetSampleImageFile();
```

Description

This is GetSampleImageFile, a member of class IInternetEnvOptions.

10.2.39.2.4 IInternetEnvOptions.GetScriptTimeoutSecs Method

Pascal

```
function GetScriptTimeoutSecs: Integer;
```

C++

```
__fastcall int GetScriptTimeoutSecs();
```

Description

This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

10.2.39.3 IInternetEnvOptions Properties

The properties of the IInternetEnvOptions class are listed here.

Public Properties

	Name	Description
	DisableDebugger (see page 986)	This is DisableDebugger, a member of class IInternetEnvOptions.
	HTMLExt (see page 986)	This is HTMLExt, a member of class IInternetEnvOptions.
	SampleImageFile (see page 986)	This is SampleImageFile, a member of class IInternetEnvOptions.
	ScriptTimeoutSecs (see page 987)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

10.2.39.3.1 IInternetEnvOptions.DisableDebugger Property

Pascal

```
property DisableDebugger: Boolean;
```

C++

```
__property Boolean DisableDebugger;
```

Description

This is DisableDebugger, a member of class IInternetEnvOptions.

10.2.39.3.2 IInternetEnvOptions.HTMLExt Property

Pascal

```
property HTMLExt: string;
```

C++

```
__property UnicodeString HTMLExt;
```

Description

This is HTMLExt, a member of class IInternetEnvOptions.

10.2.39.3.3 IInternetEnvOptions.SampleImageFile Property

Pascal

```
property SampleImageFile: string;
```

C++

```
__property UnicodeString SampleImageFile;
```

Description

This is SampleImageFile, a member of class IInternetEnvOptions.

10.2.39.3.4 IInternetEnvOptions.ScriptTimeoutSecs Property

Pascal

```
property ScriptTimeoutSecs: Integer;
```

C++

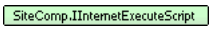
```
__property int ScriptTimeoutSecs;
```

Description

This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

10.2.40 IInternetExecuteScript Interface

Class Hierarchy



Pascal

```
IInternetExecuteScript = interface;
```

C++

```
__interface IInternetExecuteScript;
```

File

SiteComp

Description

This is class SiteComp.IInternetExecuteScript.

10.2.40.1 IInternetExecuteScript Members

The following tables list the members exposed by IInternetExecuteScript.

Public Methods

	Name	Description
	CanExecuteScript (see page 988)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 988)	This is ExecuteScript, a member of class IInternetExecuteScript.

10.2.40.2 IInternetExecuteScript Methods

The methods of the IInternetExecuteScript class are listed here.

Public Methods

	Name	Description
	CanExecuteScript (see page 988)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 988)	This is ExecuteScript, a member of class IInternetExecuteScript.

10.2.40.2.1 IInternetExecuteScript.CanExecuteScript Method

Pascal

```
function CanExecuteScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanExecuteScript(IUnknown AContext);
```

Description

This is CanExecuteScript, a member of class IInternetExecuteScript.

10.2.40.2.2 IInternetExecuteScript.ExecuteScript Method

Pascal

```
procedure ExecuteScript(AContext: IUnknown);
```

C++

```
__fastcall ExecuteScript(IUnknown AContext);
```

Description

This is ExecuteScript, a member of class IInternetExecuteScript.

10.2.41 IInternetFileTypes Interface

Class Hierarchy

SiteComp.IInternetFileTypes

Pascal

```
IInternetFileTypes = interface;
```

C++

```
__interface IInternetFileTypes;
```

File

SiteComp

Description

This is class SiteComp.IInternetFileTypes.

10.2.41.1 IInternetFileTypes Members

The following tables list the members exposed by IInternetFileTypes.

Public Methods

	Name	Description
	FindFileExt (see page 989)	This is FindFileExt, a member of class IInternetFileTypes.

10.2.41.2 IInternetFileTypes Methods

The methods of the IInternetFileTypes class are listed here.

Public Methods

	Name	Description
	FindFileExt (see page 989)	This is FindFileExt, a member of class IInternetFileTypes.

10.2.41.2.1 IInternetFileTypes.FindFileExt Method

Pascal

```
function FindFileExt(const AFileExt: string): TAbstractInetFileType;
```

C++

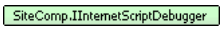
```
__fastcall TAbstractInetFileType FindFileExt(const UnicodeString AFileExt);
```

Description

This is FindFileExt, a member of class IInternetFileTypes.

10.2.42 IInternetScriptDebugger Interface

Class Hierarchy



Pascal

```
IInternetScriptDebugger = interface;
```

C++

```
__interface IInternetScriptDebugger;
```

File

SiteComp

Description

This is class SiteComp.IInternetScriptDebugger.

10.2.42.1 IInternetScriptDebugger Members

The following tables list the members exposed by IInternetScriptDebugger.

Public Methods

	Name	Description
	CanDebugScript (see page 990)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 990)	This is DebugScript, a member of class IInternetScriptDebugger.

10.2.42.2 IInternetScriptDebugger Methods

The methods of the IInternetScriptDebugger class are listed here.

Public Methods

	Name	Description
	CanDebugScript (see page 990)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 990)	This is DebugScript, a member of class IInternetScriptDebugger.

10.2.42.2.1 IInternetScriptDebugger.CanDebugScript Method

Pascal

```
function CanDebugScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanDebugScript(IUnknown AContext);
```

Description

This is CanDebugScript, a member of class IInternetScriptDebugger.

10.2.42.2.2 IInternetScriptDebugger.DebugScript Method

Pascal

```
procedure DebugScript(AContext: IUnknown);
```

C++

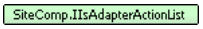
```
__fastcall DebugScript(IUnknown AContext);
```

Description

This is DebugScript, a member of class IInternetScriptDebugger.

10.2.43 IIsAdapterActionList Interface

Class Hierarchy



Pascal

```
IIsAdapterActionList = interface;
```

C++

```
__interface IIsAdapterActionList;
```

File

SiteComp

Description

This is class SiteComp.IIsAdapterActionList.

10.2.43.1 IIsAdapterActionList Members

The following tables list the members exposed by IIsAdapterActionList.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

Public Properties

	Name	Description
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

10.2.43.2 IIsAdapterActionList Methods

The methods of the IIsAdapterActionList class are listed here.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

10.2.43.2.1 IIsAdapterActionList.GetIsAdapterActionList Method

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

10.2.43.3 IIsAdapterActionList Properties

The properties of the IIsAdapterActionList class are listed here.

Public Properties

	Name	Description
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

10.2.43.3.1 IIsAdapterActionList.IsAdapterActionList Property

Pascal

```
property IsAdapterActionList: Boolean;
```

C++

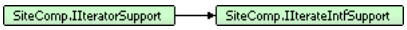
```
__property Boolean IsAdapterActionList;
```

Description

This is IsAdapterActionList, a member of class IIsAdapterActionList.

10.2.44 IIterateIntfSupport Interface

Class Hierarchy



Pascal

```
IIterateIntfSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateIntfSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateIntfSupport.

10.2.44.1 IIterateIntfSupport Members

The following tables list the members exposed by IIterateIntfSupport.

Public Methods

	Name	Description
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentIntf (see page 992)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

10.2.44.2 IIterateIntfSupport Methods

The methods of the IIterateIntfSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentIntf (see page 992)	This is GetCurrentIntf, a member of class IIterateIntfSupport.

10.2.44.2.1 IIterateIntfSupport.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

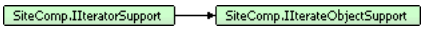
```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class IIterateIntfSupport.

10.2.45 IIterateObjectSupport Interface

Class Hierarchy



Pascal

```
IIterateObjectSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateObjectSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateObjectSupport.

10.2.45.1 IIterateObjectSupport Members

The following tables list the members exposed by IIterateObjectSupport.

Public Methods

	Name	Description
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

10.2.45.2 IIterateObjectSupport Methods

The methods of the IIterateObjectSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.

10.2.45.2.1 IIterateObjectSupport.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

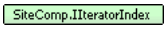
Description

This is GetCurrentObject, a member of class IIterateObjectSupport.

10.2.46 IIteratorIndex Interface

IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

Class Hierarchy



Pascal

```
IIteratorIndex = interface;
```

C++

```
__interface IIteratorIndex;
```

File

SiteComp

Description

Server-side script uses the IIteratorIndex interface to detect when an object is in the middle of an enumeration, and if it is, to determine the item on which the object is positioned. This interface is used in conjunction with the IIteratorSupport (see page 996) interface to enable server-side script to iterate over the items of an adapter or one of its sub-components.

Note: In C++, method declarations that use IIteratorIndex use the _di_IIteratorIndex type instead. This type is a DelphiInterface (see page 3263) wrapper around the IIteratorIndex interface:

Note: typedef System::DelphiInterface (see page 3263)< IIteratorIndex > _di_IIteratorIndex;

See Also

IIteratorSupport (see page 996)

10.2.46.1 IIteratorIndex Members

The following tables list the members exposed by IIteratorIndex.

Public Methods

	Name	Description
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.

Public Properties

	Name	Description
R	IteratorIndex (see page 995)	Indicates the index of the current item.

10.2.46.2 IIteratorIndex Methods

The methods of the IIteratorIndex class are listed here.

Public Methods

	Name	Description
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.

10.2.46.2.1 IIteratorIndex.GetIteratorIndex Method

Returns the value of the IteratorIndex (see page 995) property.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

GetIteratorIndex is the read implementation for the IteratorIndex (see page 995) property.

See Also

IteratorIndex (see page 995)

10.2.46.2.2 IIteratorIndex.InIterator Method

Indicates whether the object is iterating over its items.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

InIterator returns true if the object is iterating over its items. It returns false if an iteration never started or if an iteration process finished. When InIterator returns false, the current position of the object is indeterminate. In this case, the IteratorIndex (see page 995) property's value can't be used.

See Also

InIterator (see page 5549)

IIteratorSupport (see page 996)

10.2.46.3 IIteratorIndex Properties

The properties of the IIteratorIndex class are listed here.

Public Properties

	Name	Description
R	IteratorIndex (see page 995)	Indicates the index of the current item.

10.2.46.3.1 IIteratorIndex.IteratorIndex Property

Indicates the index of the current item.

Pascal

```
property IteratorIndex: Integer;
```

C++

```
__property int IteratorIndex;
```

Description

IteratorIndex indicates the index of the current item when the object is iterating over its items. When IteratorIndex is 0, the object is positioned on its first item; When IteratorIndex is 1, the object is positioned on its second item, and so on.

When the object is not iterating over its items (when the InIterator (see page 995) method returns false), the value of IteratorIndex is undefined.

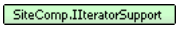
See Also

InIterator (see page 995)

10.2.47 IIteratorSupport Interface

IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.

Class Hierarchy



Pascal

```
IIteratorSupport = interface;
```

C++

```
__interface IIteratorSupport;
```

File

SiteComp

Description

Server-side script uses the IIteratorSupport interface to enumerate the items of an adapter or one of its sub-components. When the server-side script includes an enumerator, the WebSnap application automatically generates calls to the IIteratorSupport methods of the object that contains the items to enumerate.

The WebSnap application first calls the StartIterator (see page 998) method to position the object on its first item. To advance to the next item, the application calls the NextIteration (see page 997) method. Finally, the application calls the EndIterator (see page 997) method to signal the end of the iteration.

Each object that implements IIteratorSupport determines the items over which it iterates. For example, if the object is an adapter, these can be records or pages that the adapter uses to group information, or name/value pairs that the adapter represents. If the object is a list of fields or actions, these can be the fields or actions in the list. If the object is an adapter field, these can be the possible values of the field. If the object is an adapter action, these can be the commands the action can execute.

Note: In C++, method declarations that use IIteratorSupport use the _di_IIteratorSupport type instead. This type is a DelphiInterface (see page 3263) wrapper around the IIteratorSupport interface:

Note: typedef System::DelphiInterface (see page 3263)< IIteratorSupport > _di_IIteratorSupport;

See Also

IIteratorIndex (see page 994)

10.2.47.1 IIteratorSupport Members

The following tables list the members exposed by IIteratorSupport.

Public Methods

	Name	Description
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

10.2.47.2 IIteratorSupport Methods

The methods of the IIteratorSupport class are listed here.

Public Methods

	Name	Description
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

10.2.47.2.1 IIteratorSupport.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class IIteratorSupport.

10.2.47.2.2 IIteratorSupport.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator (see page 998) method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator (see page 997) method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 998)

enditerator (see page 997)

10.2.47.2.3 IIteratorSupport.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

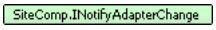
Description

This is StartIterator, a member of class IIteratorSupport.

10.2.48 INotifyAdapterChange Interface

INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.

Class Hierarchy



Pascal

```
INotifyAdapterChange = interface;
```

C++

```
__interface INotifyAdapterChange;
```

File

SiteComp

Description

Objects that depend on the fields, actions, or properties of an adapter implement the INotifyAdapterChange interface so that the adapter can notify them of changes. They register an interest in such notifications using the adapter's INotifyList (see page 999) interface. Then, when the adapter changes, it uses this interface to inform them of the change. They can remove themselves from the adapter's notification list using the INotifyList (see page 999) interface.

See Also

TCustomAdapter

TCustomAdapter

INotifyList (see page 999)

10.2.48.1 INotifyAdapterChange Members

The following tables list the members exposed by INotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.

10.2.48.2 INotifyAdapterChange Methods

The methods of the INotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.

10.2.48.2.1 INotifyAdapterChange.NotifyAdapterChange Method

Responds when the adapter changes.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

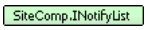
NotifyAdapterChange is the method objects implement to respond when the associated adapter changes.

Sender is the adapter component that changed.

10.2.49 INotifyList Interface

INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.

Class Hierarchy



Pascal

```
INotifyList = interface;
```

C++

```
__interface INotifyList;
```

File

SiteComp

Description

WebSnap components that let other objects register interest in their changes implement INotifyList. When another object wants to be informed of changes, it calls the AddNotify (see page 1000) method. Then, when the component that implements INotifyList changes, it informs all of the objects that have called AddNotify (see page 1000) of the change. When an object wants to unregister its interest, it calls the RemoveNotify (see page 1000) method.

See Also

TNotifyList (see page 910)

10.2.49.1 INotifyList Members

The following tables list the members exposed by INotifyList.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

10.2.49.2 INotifyList Methods

The methods of the INotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

10.2.49.2.1 INotifyList.AddNotify Method

Registers an interest in change notifications.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to register an interest in change notifications. After a call to AddNotify, the object is notified every time the component changes, until it unregisters its interest by a call to the RemoveNotify (see page 1000) method.

ANotify is the object to notify of changes. It should implement an interface that allows the notifying component to call it when changes occur. What interface this is depends on the component that implements INotifyList (see page 999).

See Also

RemoveNotify (see page 1000)

10.2.49.2.2 INotifyList.RemoveNotify Method

Unregisters interest in change notifications.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to unregister interest in change notifications. After a call to RemoveNotify, the object is no longer notified when the component changes.

ANotify is the object to unregister. It must be an object that previously registered interest by calling the AddNotify (see page 1000) method.

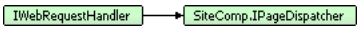
See Also

AddNotify (see page 1000)

10.2.50 IPageDispatcher Interface

IPageDispatcher is the interface for interacting with the page dispatcher.

Class Hierarchy



Pascal

```
IPageDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IPageDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TPageDispatcher component implements IPageDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages. The request messages identify a target Web page using the pathinfo portion of their target URL. If the request is not addressed to an URL with a pathinfo portion, the IPageDispatcher forwards the request to the Web page module identified by the DefaultPageName (see page 1005) property.

In addition to dispatching requests for Web pages, IPageDispatcher includes methods that check whether the current user needs to log in, check whether the current user has rights to view the requested page, and to redirect requests.

Note: In C++, method declarations that use IPageDispatcher use the _di_IPageDispatcher type instead. This type is a DelphiInterface (see page 3263) wrapper around the IPageDispatcher interface:

Note: typedef System::DelphiInterface (see page 3263)< IPageDispatcher > _di_IPageDispatcher;

See Also

TPageDispatcher (see page 6210)

TCustomWebAppComponents (see page 6175)

TWebAPpComponents

10.2.50.1 IPageDispatcher Members

The following tables list the members exposed by IPageDispatcher.

Public Methods

	Name	Description
	DispatchPageName (see page 1002)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 1003)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1003)	Returns the value of the DefaultPageName (see page 1005) property.
	GetLoginRequired (see page 1003)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1004)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1004)	Fills out a response message so that it redirects the caller to another page in the application.

Public Properties

	Name	Description
R	DefaultPageName (see page 1005)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

10.2.50.2 IPageDispatcher Methods

The methods of the IPageDispatcher class are listed here.

Public Methods

	Name	Description
	DispatchPageName (see page 1002)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 1003)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1003)	Returns the value of the DefaultPageName (see page 1005) property.
	GetLoginRequired (see page 1003)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1004)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1004)	Fills out a response message so that it redirects the caller to another page in the application.

10.2.50.2.1 IPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 1004)

HandleRequest

Content

WebContext (see page 5977)

10.2.50.2.2 IPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const UnicodeString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 1003)

HasViewRights (see page 5966)

OnCanViewPage (see page 6172)

10.2.50.2.3 IPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName (see page 1005) property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall UnicodeString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName (see page 1005) property.

See Also

DefaultPageName (see page 1005)

10.2.50.2.4 IPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const UnicodeString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 1003)

IsLoginRequired

OnIsLoginRequired (see page 6173)

10.2.50.2.5 IPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
__fastcall UnicodeString PageNameOfRequest(TWebRequest ARequest);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 1005)

10.2.50.2.6 IPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 1002)

HandleRequest

Content

WebContext (see page 5977)

10.2.50.3 IPageDispatcher Properties

The properties of the IPageDispatcher class are listed here.

Public Properties

	Name	Description
R	DefaultPageName (see page 1005)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

10.2.50.3.1 IPageDispatcher.DefaultPageName Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPageName: string;
```

C++

```
__property UnicodeString DefaultPageName;
```

Description

DefaultPageName is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPageName is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a TWebAppPageModule object.

See Also

DispatchPageName (see page 1002)

TWebAppPageModule (see page 6554)

10.2.51 IPageResult Interface

Class Hierarchy

SiteComp.IPageResult

Pascal

```
IPageResult = interface;
```

C++

```
__interface IPageResult;
```

File

SiteComp

Description

This is class SiteComp.IPageResult.

10.2.51.1 IPageResult Members

The following tables list the members exposed by IPageResult.

Public Methods

	Name	Description
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.

10.2.51.2 IPageResult Methods

The methods of the IPageResult class are listed here.

Public Methods

	Name	Description
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.

10.2.51.2.1 IPageResult.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const UnicodeString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class IPageResult.

10.2.51.2.2 IPageResult.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class IPageResult.

10.2.51.2.3 IPageResult.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AWebResponse:
TWebResponse): Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AWebResponse);
```

Description

This is RedirectToPage, a member of class IPageResult.

10.2.52 ISessionsService Interface

ISessionsService is the interface for controlling a TSessionsService object.

Class Hierarchy

SiteComp.ISessionsService

Pascal

```
ISessionsService = interface;
```

C++

```
__interface ISessionsService;
```

File

SiteComp

Description

The TSessionsService component implements ISessionsService. Most applications do not need to use this interface directly. Instead, the TWebAppComponents component uses this interface to drive the orderly management of sessions. In addition, Web session objects (TWebSession) use this interface to update the sessions service when they are terminated.

Note: In C++, method declarations that use ISessionsService use the _di_ISessionsService type instead. This type is a DelphiInterface (see page 3263) wrapper around the ISessionsService interface:

Note: typedef System::DelphiInterface (see page 3263)< ISessionsService > _di_ISessionsService;

See Also

TSessionsService (see page 6722)

TWebAppComponents (see page 6217)

TWebSession (see page 6225)

10.2.52.1 ISessionsService Members

The following tables list the members exposed by ISessionsService.

Public Methods

	Name	Description
	CheckStatusChange (see page 1008)	Checks whether any sessions have started or terminated.
	Clear (see page 1008)	Deletes all sessions from memory.
	NewSession (see page 1009)	Starts a new session.
	NotifyStatusChange (see page 1009)	Generates an OnEndSession or OnStartSession event, as approrpriate.

10.2.52.2 ISessionsService Methods

The methods of the ISessionsService class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 1008)	Checks whether any sessions have started or terminated.
	Clear (see page 1008)	Deletes all sessions from memory.
	NewSession (see page 1009)	Starts a new session.
	NotifyStatusChange (see page 1009)	Generates an OnEndSession or OnStartSession event, as approrpriate.

10.2.52.2.1 ISessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange examines the sessions currently in memory to see if any sessions have been started or terminated since the last call to NotifyStatusChange (see page 1009). If a session has started or terminated since the last call, CheckStatusChange returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 1009)

10.2.52.2.2 ISessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory. Clear does not terminate active sessions in an orderly fashion (for example, it omits the call to NotifyStatusChange (see page 1009)), it merely frees all memory used to store sessions.

See Also

NotifyStatusChange (see page 1009)

10.2.52.2.3 ISessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

See Also

NotifyStatusChange (see page 1009)

10.2.52.2.4 ISessionsService.NotifyStatusChange Method

Generates an OnEndSession or OnStartSession event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession event for any new sessions that have been created since the last call and to generate the OnEndSession event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession event handler exits.

See Also

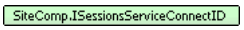
OnEndSession (see page 6721)

OnStartSession (see page 6721)

CheckStatusChange (see page 1008)

10.2.53 ISessionsServiceConnectID Interface

Class Hierarchy



Pascal

```
ISessionsServiceConnectID = interface;
```

C++

```
__interface ISessionsServiceConnectID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceConnectID.

10.2.53.1 ISessionsServiceConnectID Members

The following tables list the members exposed by ISessionsServiceConnectID.

Public Methods

	Name	Description
	Connect (see page 1010)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1011)	This is ConnectActive, a member of class ISessionsServiceConnectID.

10.2.53.2 ISessionsServiceConnectID Methods

The methods of the ISessionsServiceConnectID class are listed here.

Public Methods

	Name	Description
	Connect (see page 1010)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1011)	This is ConnectActive, a member of class ISessionsServiceConnectID.

10.2.53.2.1 ISessionsServiceConnectID.Connect Method

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

This is Connect, a member of class ISessionsServiceConnectID.

10.2.53.2.2 ISessionsServiceConnectID.ConnectActive Method

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

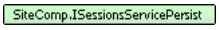
```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

This is ConnectActive, a member of class ISessionsServiceConnectID.

10.2.54 ISessionsServicePersist Interface

Class Hierarchy



Pascal

```
ISessionsServicePersist = interface;
```

C++

```
__interface ISessionsServicePersist;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServicePersist.

10.2.54.1 ISessionsServicePersist Members

The following tables list the members exposed by ISessionsServicePersist.

Public Methods

	Name	Description
	LoadFromFile (see page 1012)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1012)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 1012)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 1012)	This is SaveToStream, a member of class ISessionsServicePersist.

10.2.54.2 ISessionsServicePersist Methods

The methods of the ISessionsServicePersist class are listed here.

Public Methods

	Name	Description
	LoadFromFile (see page 1012)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1012)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 1012)	This is SaveToFile, a member of class ISessionsServicePersist.

	SaveToStream (see page 1012)	This is SaveToStream, a member of class ISessionsServicePersist.

10.2.54.2.1 ISessionsServicePersist.LoadFromFile Method

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

This is LoadFromFile, a member of class ISessionsServicePersist.

10.2.54.2.2 ISessionsServicePersist.LoadFromStream Method

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

This is LoadFromStream, a member of class ISessionsServicePersist.

10.2.54.2.3 ISessionsServicePersist.SaveToFile Method

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

This is SaveToFile, a member of class ISessionsServicePersist.

10.2.54.2.4 ISessionsServicePersist.SaveToStream Method

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

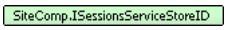
```
__fastcall SaveToStream(TStream S);
```

Description

This is SaveToStream, a member of class ISessionsServicePersist.

10.2.55 ISessionsServiceStoreID Interface

Class Hierarchy



Pascal

```
ISessionsServiceStoreID = interface;
```

C++

```
__interface ISessionsServiceStoreID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceStoreID.

10.2.55.1 ISessionsServiceStoreID Members

The following tables list the members exposed by ISessionsServiceStoreID.

Public Methods

	Name	Description
	ReadSessionID (see page 1013)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 1014)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

10.2.55.2 ISessionsServiceStoreID Methods

The methods of the ISessionsServiceStoreID class are listed here.

Public Methods

	Name	Description
	ReadSessionID (see page 1013)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 1014)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

10.2.55.2.1 ISessionsServiceStoreID.ReadSessionID Method

Pascal

```
function ReadSessionID(ARequest: TWebRequest; AResponse: TWebResponse; out AID: TSessionID):
Boolean;
```

C++

```
__fastcall Boolean ReadSessionID(TWebRequest ARequest, TWebResponse AResponse, TSessionID AID);
```

Description

This is ReadSessionID, a member of class ISessionsServiceStoreID.

10.2.55.2.2 ISessionsServiceStoreID.WriteSessionID Method

Pascal

```
procedure WriteSessionID(AResponse: TWebResponse; AID: TSessionID);
```

C++

```
__fastcall WriteSessionID(TWebResponse AResponse, TSessionID AID);
```

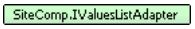
Description

This is WriteSessionID, a member of class ISessionsServiceStoreID.

10.2.56 IValuesListAdapter Interface

IValuesListAdapter is the interface for supplying name/value pairs.

Class Hierarchy



Pascal

```
IValuesListAdapter = interface;
```

C++

```
__interface IValuesListAdapter;
```

File

SiteComp

Description

Values list adapters such as TStringsValuesList and TDataSetValuesList implement the IValuesListAdapter interface. This allows adapter fields to use the values list adapter for supplying an associated list of name/value pairs. Typically, this list is used to enumerate the possible values for the adapter field so that they can be added to a selection control such as a list box in an HTML form.

The IValuesListAdapter interface introduces methods to read the current name or value, and to look up the name (or image) associated with a specified value. If the adapter that implements IValuesListAdapter does not also support iteration, server-side script can't change the current name/value pair. However, it can still look up the name or image for a specified value.

Note: In C++, method declarations that use IValuesListAdapter use the _di_IValuesListAdapter type instead. This type is a DelphiInterface (see page 3263) wrapper around the IValuesListAdapter interface:

Note: typedef System::DelphiInterface (see page 3263)< IValuesListAdapter > _di_IValuesListAdapter;

See Also

TAdapterUpdateField (see page 5407)

TCustomDataSetAdapterField

TStringsValuesList (see page 5843)

TDataSetValuesList

IIteratorSupport (see page 996)

10.2.56.1 IValuesListAdapter Members

The following tables list the members exposed by IValuesListAdapter.

Public Methods

	Name	Description
	GetListImage (see page 1015)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1016)	Returns the image object associated with a specified value.
	GetListName (see page 1016)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1016)	Returns the name associated with a specified value.
	GetListValue (see page 1017)	Returns the value of the current name/value pair.

10.2.56.2 IValuesListAdapter Methods

The methods of the IValuesListAdapter class are listed here.

Public Methods

	Name	Description
	GetListImage (see page 1015)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1016)	Returns the image object associated with a specified value.
	GetListName (see page 1016)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1016)	Returns the name associated with a specified value.
	GetListValue (see page 1017)	Returns the value of the current name/value pair.

10.2.56.2.1 IValuesListAdapter.GetListImage Method

Returns the image object associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the object that represents an image associated with the current name/value pair. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

If the current name/value pair does not have an associated image, GetListImage returns nil (Delphi) or NULL (C++).

Note: GetListImage provides the support for the Image property of the values list adapter, which can be addressed by server-side script.

See Also

GetListImageOfValue (see page 1016)

GetListName (see page 1016)

GetListValue (see page 1017)

TWebRequestFile

10.2.56.2.2 IValuesListAdapter.GetListImageOfValue Method

Returns the image object associated with a specified value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue looks for a name/value pair with the specified value and returns the object that represents an image associated with that value. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

Value is the value part of the name/value pair whose image you want.

If Value does not identify a name/value pair with an associated image, GetListImageOfValue returns nil (Delphi) or NULL (C++).

See Also

GetListImage (see page 1015)

GetListNameOfValue (see page 1016)

GetListValue (see page 1017)

TWebRequestFile

10.2.56.2.3 IValuesListAdapter.GetListName Method

Returns the name part of the current name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall UnicodeString GetListName();
```

Description

GetListName returns the name part of the current name/value pair. This method provides the support for the Name property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a name, GetListName returns an empty string.

See Also

GetListNameOfValue (see page 1016)

GetListImage (see page 1015)

GetListValue (see page 1017)

10.2.56.2.4 IValuesListAdapter.GetListNameOfValue Method

Returns the name associated with a specified value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall UnicodeString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue looks for a name/value pair with the specified value and returns the name part.

Value is the value part of the name/value pair whose name you want.

If Value does not identify a name/value pair with a name, GetListNameOfValue returns an empty string.

See Also

GetListName (see page 1016)

GetListImageOfValue (see page 1016)

GetListValue (see page 1017)

10.2.56.2.5 IValuesListAdapter.GetListValue Method

Returns the value of the current name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the value part of the current name/value pair. This method provides the support for the Value property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a value, GetListValue returns either the name part, or an unassigned Variant (see page 3430), depending on the type of values list adapter.

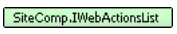
See Also

GetListName (see page 1016)

GetListImage (see page 1015)

10.2.57 IWebActionsList Interface

Class Hierarchy



Pascal

```
IWebActionsList = interface;
```

C++

```
__interface IWebActionsList;
```

File

SiteComp

Description

This is class SiteComp.IWebActionsList.

10.2.57.1 IWebActionsList Members

The following tables list the members exposed by IWebActionsList.

Public Methods

	Name	Description
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.

Public Properties

	Name	Description
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.

10.2.57.2 IWebActionsList Methods

The methods of the IWebActionsList class are listed here.

Public Methods

	Name	Description
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.

10.2.57.2.1 IWebActionsList.FindAction Method

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

This is FindAction, a member of class IWebActionsList.

10.2.57.2.2 IWebActionsList.GetActionsList Method

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

This is GetActionsList, a member of class IWebActionsList.

10.2.57.2.3 IWebActionsList.GetVisibleActions Method

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

This is GetVisibleActions, a member of class IWebActionsList.

10.2.57.2.4 IWebActionsList.IsActionInUse Method

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

This is IsActionInUse, a member of class IWebActionsList.

10.2.57.3 IWebActionsList Properties

The properties of the IWebActionsList class are listed here.

Public Properties

	Name	Description
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.

10.2.57.3.1 IWebActionsList.VisibleActions Property

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

Description

This is VisibleActions, a member of class IWebActionsList.

10.2.58 IWebApplicationInfo Interface

IWebApplicationInfo is the interface for the application adapter.

Class Hierarchy

Pascal

```
IWebApplicationInfo = interface;
```

C++

```
__interface IWebApplicationInfo;
```

File

SiteComp

Description

The TApplicationAdapter component implements IWebApplicationInfo. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to obtain application. IWebApplicationInfo declares an ApplicationTitle (see page 1022) property, which allows server-side scripts to use the title of the application. It also provides access to the application adapter object, which may provide additional application-level information.

Note: In C++, method declarations that use IWebApplicationInfo use the _di_IWebApplicationInfo type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebApplicationInfo interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebApplicationInfo > _di_IWebApplicationInfo;

See Also

TApplicationAdapter (see page 5427)

TCustomWebAppComponents (see page 6175)

TWebAPpComponents

10.2.58.1 IWebApplicationInfo Members

The following tables list the members exposed by IWebApplicationInfo.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 1021)	Returns the value of the ApplicationAdapter (see page 1021) property.
	GetApplicationTitle (see page 1021)	Returns the value of the ApplicationTitle (see page 1022) property.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 1021)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 1022)	Indicates the title of the WebSnap application.

10.2.58.2 IWebApplicationInfo Methods

The methods of the IWebApplicationInfo class are listed here.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 1021)	Returns the value of the ApplicationAdapter (see page 1021) property.
	GetApplicationTitle (see page 1021)	Returns the value of the ApplicationTitle (see page 1022) property.

10.2.58.2.1 IWebApplicationInfo.GetApplicationAdapter Method

Returns the value of the ApplicationAdapter (see page 1021) property.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter is the read implementation of the ApplicationAdapter (see page 1021) property.

See Also

ApplicationAdapter (see page 1021)

10.2.58.2.2 IWebApplicationInfo.GetApplicationTitle Method

Returns the value of the ApplicationTitle (see page 1022) property.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall UnicodeString GetApplicationTitle();
```

Description

GetApplicationTitle is the read implementation of the ApplicationTitle (see page 1022) property.

See Also

ApplicationTitle (see page 1022)

10.2.58.3 IWebApplicationInfo Properties

The properties of the IWebApplicationInfo class are listed here.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 1021)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 1022)	Indicates the title of the WebSnap application.

10.2.58.3.1 IWebApplicationInfo.ApplicationAdapter Property

Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.

Pascal

```
property ApplicationAdapter: TComponent;
```

C++

```
__property TComponent * ApplicationAdapter;
```

Description

Read (see page 3727) ApplicationAdapter to access the application adapter component. This is the adapter that implements the fields and actions of the Application variable in server-side scripts. Typically, this is an instance of TApplicationAdapter.

See Also

TApplicationAdapter (see page 5427)

10.2.58.3.2 IWebApplicationInfo.ApplicationTitle Property

Indicates the title of the WebSnap application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;
```

Description

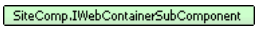
Read (see page 3727) ApplicationTitle to obtain the title of the WebSnap application. This property is the value of the Title field in the Application variable in server-side scripts.

See Also

Title (see page 5916)

10.2.59 IWebContainerSubComponent Interface

Class Hierarchy



Pascal

```
IWebContainerSubComponent = interface;
```

C++

```
__interface IWebContainerSubComponent;
```

File

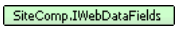
SiteComp

Description

This is class SiteComp.IWebContainerSubComponent.

10.2.60 IWebDataFields Interface

Class Hierarchy



Pascal

```
IWebDataFields = interface;
```

C++

```
__interface IWebDataFields;
```

File

SiteComp

Description

This is class SiteComp.IWebDataFields.

10.2.60.1 IWebDataFields Members

The following tables list the members exposed by IWebDataFields.

Public Methods

	Name	Description
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.

Public Properties

	Name	Description
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

10.2.60.2 IWebDataFields Methods

The methods of the IWebDataFields class are listed here.

Public Methods

	Name	Description
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.

10.2.60.2.1 IWebDataFields.FindField Method

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

This is FindField, a member of class IWebDataFields.

10.2.60.2.2 IWebDataFields.GetFieldsList Method

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

This is GetFieldsList, a member of class IWebDataFields.

10.2.60.2.3 IWebDataFields.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class IWebDataFields.

10.2.60.2.4 IWebDataFields.IsFieldInUse Method

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

This is IsFieldInUse, a member of class IWebDataFields.

10.2.60.3 IWebDataFields Properties

The properties of the IWebDataFields class are listed here.

Public Properties

	Name	Description
	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

10.2.60.3.1 IWebDataFields.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

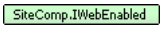
```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class IWebDataFields.

10.2.61 IWebEnabled Interface

Class Hierarchy



Pascal

```
IWebEnabled = interface;
```

C++

```
__interface IWebEnabled;
```

File

SiteComp

Description

This is class SiteComp.IWebEnabled.

10.2.61.1 IWebEnabled Members

The following tables list the members exposed by IWebEnabled.

Public Methods

	Name	Description
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

10.2.61.2 IWebEnabled Methods

The methods of the IWebEnabled class are listed here.

Public Methods

	Name	Description
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

10.2.61.2.1 IWebEnabled.WebEnabled Method

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

This is WebEnabled, a member of class IWebEnabled.

10.2.62 IWebFormComponentContainer Interface

Class Hierarchy

SiteComp.IWebFormComponentContainer

Pascal

```
IWebFormComponentContainer = interface;
```

C++

```
__interface IWebFormComponentContainer;
```

File

SiteComp

Description

This is class SiteComp.IWebFormComponentContainer.

10.2.62.1 IWebFormComponentContainer Members

The following tables list the members exposed by IWebFormComponentContainer.

Public Methods

	Name	Description
	GetCount (see page 1027)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 1027)	This is GetItem, a member of class IWebFormComponentContainer.

Public Properties

	Name	Description
R	Count (see page 1027)	This is Count, a member of class IWebFormComponentContainer.
R	Items (see page 1028)	This is Items, a member of class IWebFormComponentContainer.

10.2.62.2 IWebFormComponentContainer Methods

The methods of the IWebFormComponentContainer class are listed here.

Public Methods

	Name	Description
	GetCount (see page 1027)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 1027)	This is GetItem, a member of class IWebFormComponentContainer.

10.2.62.2.1 IWebFormComponentContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class IWebFormComponentContainer.

10.2.62.2.2 IWebFormComponentContainer.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class IWebFormComponentContainer.

10.2.62.3 IWebFormComponentContainer Properties

The properties of the IWebFormComponentContainer class are listed here.

Public Properties

	Name	Description
R	Count (see page 1027)	This is Count, a member of class IWebFormComponentContainer.
R	Items (see page 1028)	This is Items, a member of class IWebFormComponentContainer.

10.2.62.3.1 IWebFormComponentContainer.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IWebFormComponentContainer.

10.2.62.3.2 IWebFormComponentContainer.Items Property

Pascal

```
property Items [I: Integer]: TComponent;
```

C++

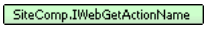
```
__property TComponent * Items[int I];
```

Description

This is Items, a member of class IWebFormComponentContainer.

10.2.63 IWebGetActionName Interface

Class Hierarchy



Pascal

```
IWebGetActionName = interface;
```

C++

```
__interface IWebGetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetActionName.

10.2.63.1 IWebGetActionName Members

The following tables list the members exposed by IWebGetActionName.

Public Methods

	Name	Description
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.

10.2.63.2 IWebGetActionName Methods

The methods of the IWebGetActionName class are listed here.

Public Methods

	Name	Description
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.

10.2.63.2.1 IWebGetActionName.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

This is GetActionName, a member of class IWebGetActionName.

10.2.63.3 IWebGetActionName Properties

The properties of the IWebGetActionName class are listed here.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.

10.2.63.3.1 IWebGetActionName.ActionName Property

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

This is ActionName, a member of class IWebGetActionName.

10.2.64 IWebGetDisplayLabel Interface

Class Hierarchy

SiteComp.IWebGetDisplayLabel

Pascal

```
IWebGetDisplayLabel = interface;
```

C++

```
__interface IWebGetDisplayLabel;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayLabel.

10.2.64.1 IWebGetDisplayLabel Members

The following tables list the members exposed by IWebGetDisplayLabel.

Public Methods

	Name	Description
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

10.2.64.2 IWebGetDisplayLabel Methods

The methods of the IWebGetDisplayLabel class are listed here.

Public Methods

	Name	Description
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

10.2.64.2.1 IWebGetDisplayLabel.GetDisplayLabel Method

Pascal

```
function GetDisplayLabel: string;
```

C++

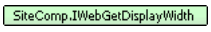
```
__fastcall UnicodeString GetDisplayLabel();
```

Description

This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

10.2.65 IWebGetDisplayWidth Interface

Class Hierarchy



Pascal

```
IWebGetDisplayWidth = interface;
```

C++

```
__interface IWebGetDisplayWidth;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayWidth.

10.2.65.1 IWebGetDisplayWidth Members

The following tables list the members exposed by IWebGetDisplayWidth.

Public Methods

	Name	Description
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

10.2.65.2 IWebGetDisplayWidth Methods

The methods of the IWebGetDisplayWidth class are listed here.

Public Methods

	Name	Description
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

10.2.65.2.1 IWebGetDisplayWidth.GetDisplayWidth Method

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

10.2.66 IWebGetFieldName Interface

Class Hierarchy

SiteComp.IWebGetFieldName

Pascal

```
IWebGetFieldName = interface;
```

C++

```
__interface IWebGetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldName.

10.2.66.1 IWebGetFieldName Members

The following tables list the members exposed by IWebGetFieldName.

Public Methods

	Name	Description
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.

Public Properties

	Name	Description
	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.

10.2.66.2 IWebGetFieldName Methods

The methods of the IWebGetFieldName class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.

10.2.66.2.1 IWebGetFieldName.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class IWebGetFieldName.

10.2.66.3 IWebGetFieldName Properties

The properties of the IWebGetFieldName class are listed here.

Public Properties

	Name	Description
	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.

10.2.66.3.1 IWebGetFieldName.FieldName Property

Pascal

```
property FieldName: string;
```

C++

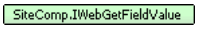
```
__property UnicodeString FieldName;
```

Description

This is FieldName, a member of class IWebGetFieldName.

10.2.67 IWebGetFieldValue Interface

Class Hierarchy



Pascal

```
IWebGetFieldValue = interface;
```

C++

```
__interface IWebGetFieldValue;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldValue.

10.2.67.1 IWebGetFieldValue Members

The following tables list the members exposed by IWebGetFieldValue.

Public Methods

	Name	Description
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.

Public Properties

	Name	Description
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.

10.2.67.2 IWebGetFieldValue Methods

The methods of the IWebGetFieldValue class are listed here.

Public Methods

	Name	Description
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.

10.2.67.2.1 IWebGetFieldValue.GetValue Method

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

This is GetValue, a member of class IWebGetFieldValue.

10.2.67.3 IWebGetFieldValue Properties

The properties of the IWebGetFieldValue class are listed here.

Public Properties

	Name	Description
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.

10.2.67.3.1 IWebGetFieldValue.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IWebGetFieldValue.

10.2.68 IWebGetFieldValues Interface

IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.

Class Hierarchy

Pascal

```
IWebGetFieldValues = interface;
```

C++

```
__interface IWebGetFieldValues;
```

File

SiteComp

Description

IWebGetFieldValues is implemented by adapter fields that may have zero or more associated values. Server-side script accesses these values using the adapter field's Values property.

10.2.68.1 IWebGetFieldValues Members

The following tables list the members exposed by IWebGetFieldValues.

Public Methods

	Name	Description
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.

10.2.68.2 IWebGetFieldValues Methods

The methods of the IWebGetFieldValues class are listed here.

Public Methods

	Name	Description
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.

10.2.68.2.1 IWebGetFieldValues.EndValueIterator Method

Indicate the iteration is finished.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

EndValueIterator is used with StartValueIterator (see page 1036) and NextValueIteration (see page 1036) to loop through the values associated with an adapter field. EndValueIterator should be called when iteration is complete. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 1036)

NextValueIteration (see page 1036)

10.2.68.2.2 IWebGetFieldValues.GetCurrentValue Method

Retrieves the current value as a variant.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue retrieves the value associated with internal iteration pointer. The FirstValueIterator and NextValueIteration (see page 1036) are called to position the internal iteration pointer. Server-side script accesses this value using the Values.Value property.

See Also

StartValueIterator (see page 1036)

NextValueIteration (see page 1036)

10.2.68.2.3 IWebGetFieldValues.HasValue Method

Indicates whether the adapter field has a matching value.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

Server-side script accesses this value using the Values.HasValue method. Server-side script to generate an HTML select list or check box group can use this method to determine whether an item should be selected or checked.

See Also

ViewMode (see page 6442)

10.2.68.2.4 IWebGetFieldValues.NextValueIteration Method

Position the iteration pointer on the next value.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration is used with StartValueIterator (see page 1036) and EndValueIterator (see page 1035) to loop through the values associated with an adapter field. NextValueIteration returns False if there are no more values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 1036)

EndValueIterator (see page 1035)

GetCurrentValue (see page 1035)

10.2.68.2.5 IWebGetFieldValues.StartValueIterator Method

Reset (see page 3734) the iteration pointer to the first item.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator is used with NextValueIteration (see page 1036) and EndValueIterator (see page 1035) to loop through the values associated with an adapter field. StartValueIterator returns false if there are no values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

NextValueIteration (see page 1036)

EndValueIterator (see page 1035)

GetCurrentValue (see page 1035)

10.2.69 IWebGetMaxLength Interface

Class Hierarchy

Pascal

```
IWebGetMaxLength = interface;
```

C++

```
__interface IWebGetMaxLength;
```

File

SiteComp

Description

This is class SiteComp.IWebGetMaxLength.

10.2.69.1 IWebGetMaxLength Members

The following tables list the members exposed by IWebGetMaxLength.

Public Methods

	Name	Description
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.

10.2.69.2 IWebGetMaxLength Methods

The methods of the IWebGetMaxLength class are listed here.

Public Methods

	Name	Description
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.

10.2.69.2.1 IWebGetMaxLength.GetMaxLength Method

Pascal

```
function GetMaxLength: integer;
```

C++

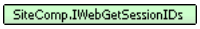
```
__fastcall int GetMaxLength();
```

Description

This is GetMaxLength, a member of class IWebGetMaxLength.

10.2.70 IWebGetSessionIDs Interface

Class Hierarchy



Pascal

```
IWebGetSessionIDs = interface;
```

C++

```
__interface IWebGetSessionIDs;
```

File

SiteComp

Description

This is class SiteComp.IWebGetSessionIDs.

10.2.70.1 IWebGetSessionIDs Members

The following tables list the members exposed by IWebGetSessionIDs.

Public Methods

	Name	Description
	GetSessionIDs (see page 1038)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

10.2.70.2 IWebGetSessionIDs Methods

The methods of the IWebGetSessionIDs class are listed here.

Public Methods

	Name	Description
	GetSessionIDs (see page 1038)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

10.2.70.2.1 IWebGetSessionIDs.GetSessionIDs Method

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

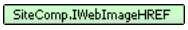
Description

This is GetSessionIDs, a member of class IWebGetSessionIDs.

10.2.71 IWebImageHREF Interface

IWebImageHREF is the interface for obtaining an HREF that addresses an image.

Class Hierarchy



Pascal

```
IWebImageHREF = interface;
```

C++

```
__interface IWebImageHREF;
```

File

SiteComp

Description

IWebImageHREF defines a single method, WebImageHREF (see page 1040), which supplies a URL for an image that should appear in HTML forms. This interface is used when server-side script addresses the Image.AsHREF property of an adapter.

Adapter field objects that represent images (that is, adapter fields with an Image property) implement IWebImageHREF so that the image reference can be obtained from server-side script. Such adapter fields include TAdapterImageField and TDataSetAdapterImageField.

Note: Method declarations that use IWebImageHREF use the _di_IWebImageHREF type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebImageHREF interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebImageHREF >  _di_IWebImageHREF;
```

See Also

TAdapterImageField (see page 5297)

TDataSetAdapterImageField

10.2.71.1 IWebImageHREF Members

The following tables list the members exposed by IWebImageHREF.

Public Methods

	Name	Description
	WebImageHREF (see page 1040)	Retrieves a URL for an external image file.

10.2.71.2 IWebImageHREF Methods

The methods of the IWebImageHREF class are listed here.

Public Methods

	Name	Description
	WebImageHREF (see page 1040)	Retrieves a URL for an external image file.

10.2.71.2.1 IWebImageHREF.WebImageHREF Method

Retrieves a URL for an external image file.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(UnicodeString AHREF);
```

Description

WebImageHREF is called when server-side script uses the Image.AsHREF property. WebImageHREF returns a URL that identifies a specific image that is available as a simple file reference. If images must be generated dynamically in response to requests, then WebImageHREF returns false and the adapter dispatcher generates an HREF based on the request parameters supplied by the object that implements this interface.

AHREF returns the URL that identifies the image of this field.

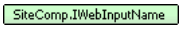
WebImageHREF returns true if it supplies a URL as the value of the AHREF parameter, false if the HTML form must request images using a set of request parameters instead.

See Also

EncodeAdapterItemRequestAsHREF (see page 6147)

10.2.72 IWebInputName Interface

Class Hierarchy



Pascal

```
IWebInputName = interface;
```

C++

```
__interface IWebInputName;
```

File

SiteComp

Description

This is class SiteComp.IWebInputName.

10.2.72.1 IWebInputName Members

The following tables list the members exposed by IWebInputName.

Public Methods

	Name	Description
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.

Public Properties

	Name	Description
	InputName (see page 1041)	This is InputName, a member of class IWebInputName.

10.2.72.2 IWebInputName Methods

The methods of the IWebInputName class are listed here.

Public Methods

	Name	Description
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.

10.2.72.2.1 IWebInputName.GetInputName Method

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall UnicodeString GetInputName();
```

Description

This is GetInputName, a member of class IWebInputName.

10.2.72.3 IWebInputName Properties

The properties of the IWebInputName class are listed here.

Public Properties

	Name	Description
	InputName (see page 1041)	This is InputName, a member of class IWebInputName.

10.2.72.3.1 IWebInputName.InputName Property

Pascal

```
property InputName: string;
```

C++

```
__property UnicodeString InputName;
```

Description

This is InputName, a member of class IWebInputName.

10.2.73 IWebIsDefaultAction Interface

Class Hierarchy

SiteComp.IWebIsDefaultAction

Pascal

```
IWebIsDefaultAction = interface;
```

C++

```
__interface IWebIsDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultAction.

10.2.73.1 IWebIsDefaultAction Members

The following tables list the members exposed by IWebIsDefaultAction.

Public Methods

	Name	Description
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

10.2.73.2 IWebIsDefaultAction Methods

The methods of the IWebIsDefaultAction class are listed here.

Public Methods

	Name	Description
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

10.2.73.2.1 IWebIsDefaultAction.IsDefaultAction Method

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

This is IsDefaultAction, a member of class IWebIsDefaultAction.

10.2.74 IWebIsDefaultField Interface

Class Hierarchy

SiteComp.IWebIsDefaultField

Pascal

```
IWebIsDefaultField = interface;
```

C++

```
__interface IWebIsDefaultField;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultField.

10.2.74.1 IWebIsDefaultField Members

The following tables list the members exposed by IWebIsDefaultField.

Public Methods

	Name	Description
	IsDefaultField (see page 1043)	This is IsDefaultField, a member of class IWebIsDefaultField.

10.2.74.2 IWebIsDefaultField Methods

The methods of the IWebIsDefaultField class are listed here.

Public Methods

	Name	Description
	IsDefaultField (see page 1043)	This is IsDefaultField, a member of class IWebIsDefaultField.

10.2.74.2.1 IWebIsDefaultField.IsDefaultField Method

Pascal

```
function IsDefaultField(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultField(TObject * ADisplay);
```

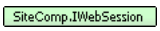
Description

This is IsDefaultField, a member of class IWebIsDefaultField.

10.2.75 IWebSession Interface

IWebSession allows an application to terminate a Web session.

Class Hierarchy



Pascal

```
IWebSession = interface;
```

C++

```
__interface IWebSession;
```

File

SiteComp

Description

Use IWebSession to terminate an active Web session. When you add a new session to a TSessionsService component using the ISessionsService (see page 1007) interface, it returns an IWebSession interface for the new session. You can use this interface to terminate the session. Once a session terminates, the application can no longer access its named values.

Note: In C++, method declarations that use IWebSession use the _di_IWebSession type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebSession interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebSession > _di_IWebSession;

See Also

TWebSession (see page 6225)

TSessionsService (see page 6722)

ISessionsService (see page 1007)

10.2.75.1 IWebSession Members

The following tables list the members exposed by IWebSession.

Public Methods

	Name	Description
	Terminate (see page 1044)	Terminates the session.

10.2.75.2 IWebSession Methods

The methods of the IWebSession class are listed here.

Public Methods

	Name	Description
	Terminate (see page 1044)	Terminates the session.

10.2.75.2.1 IWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

See Also

TimeoutMinutes

Values

10.2.76 IWebSessionAttributes Interface

Class Hierarchy

SiteComp.IWebSessionAttributes

Pascal

```
IWebSessionAttributes = interface;
```

C++

```
__interface IWebSessionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionAttributes.

10.2.76.1 IWebSessionAttributes Members

The following tables list the members exposed by IWebSessionAttributes.

Public Methods

	Name	Description
	GetExpirationTime (see page 1046)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 1046)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 1046)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 1047)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 1047)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 1047)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.
	GetTouchTime (see page 1047)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 1047)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

Public Properties

	Name	Description
R	ExpirationTime (see page 1048)	This is ExpirationTime, a member of class IWebSessionAttributes.
R	Expired (see page 1048)	This is Expired, a member of class IWebSessionAttributes.
R	IsActive (see page 1048)	This is IsActive, a member of class IWebSessionAttributes.
R	StartTime (see page 1049)	This is StartTime, a member of class IWebSessionAttributes.
R	Terminated (see page 1049)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 1049)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
R	TouchTime (see page 1049)	This is TouchTime, a member of class IWebSessionAttributes.

10.2.76.2 IWebSessionAttributes Methods

The methods of the IWebSessionAttributes class are listed here.

Public Methods

	Name	Description
	GetExpirationTime (see page 1046)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 1046)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 1046)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 1047)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 1047)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 1047)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.
	GetTouchTime (see page 1047)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 1047)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

10.2.76.2.1 IWebSessionAttributes.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class IWebSessionAttributes.

10.2.76.2.2 IWebSessionAttributes.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class IWebSessionAttributes.

10.2.76.2.3 IWebSessionAttributes.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class IWebSessionAttributes.

10.2.76.2.4 IWebSessionAttributes.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class IWebSessionAttributes.

10.2.76.2.5 IWebSessionAttributes.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class IWebSessionAttributes.

10.2.76.2.6 IWebSessionAttributes.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class IWebSessionAttributes.

10.2.76.2.7 IWebSessionAttributes.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class IWebSessionAttributes.

10.2.76.2.8 IWebSessionAttributes.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

10.2.76.3 IWebSessionAttributes Properties

The properties of the IWebSessionAttributes class are listed here.

Public Properties

	Name	Description
	ExpirationTime (see page 1048)	This is ExpirationTime, a member of class IWebSessionAttributes.
	Expired (see page 1048)	This is Expired, a member of class IWebSessionAttributes.
	IsActive (see page 1048)	This is IsActive, a member of class IWebSessionAttributes.
	StartTime (see page 1049)	This is StartTime, a member of class IWebSessionAttributes.
	Terminated (see page 1049)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 1049)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
	TouchTime (see page 1049)	This is TouchTime, a member of class IWebSessionAttributes.

10.2.76.3.1 IWebSessionAttributes.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class IWebSessionAttributes.

10.2.76.3.2 IWebSessionAttributes.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class IWebSessionAttributes.

10.2.76.3.3 IWebSessionAttributes.IsActive Property

Pascal

```
property IsActive: Boolean;
```

C++

```
__property Boolean IsActive;
```

Description

This is IsActive, a member of class IWebSessionAttributes.

10.2.76.3.4 IWebSessionAttributes.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class IWebSessionAttributes.

10.2.76.3.5 IWebSessionAttributes.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class IWebSessionAttributes.

10.2.76.3.6 IWebSessionAttributes.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class IWebSessionAttributes.

10.2.76.3.7 IWebSessionAttributes.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

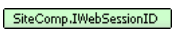
```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class IWebSessionAttributes.

10.2.77 IWebSessionID Interface

Class Hierarchy



Pascal

```
IWebSessionID = interface;
```

C++

```
__interface IWebSessionID;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionID.

10.2.77.1 IWebSessionID Members

The following tables list the members exposed by IWebSessionID.

Public Methods

	Name	Description
	GetID (see page 1050)	This is GetID, a member of class IWebSessionID.

Public Properties

	Name	Description
R	ID (see page 1051)	This is ID, a member of class IWebSessionID.

10.2.77.2 IWebSessionID Methods

The methods of the IWebSessionID class are listed here.

Public Methods

	Name	Description
	GetID (see page 1050)	This is GetID, a member of class IWebSessionID.

10.2.77.2.1 IWebSessionID.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class IWebSessionID.

10.2.77.3 IWebSessionID Properties

The properties of the IWebSessionID class are listed here.

Public Properties

	Name	Description
R	ID (see page 1051)	This is ID, a member of class IWebSessionID.

10.2.77.3.1 IWebSessionID.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class IWebSessionID.

10.2.78 IWebSessionIDs Interface

IWebSessionIDs lists the session IDs (see page 1053) for all sessions that TSessionsService stores in memory.

Class Hierarchy

Pascal

```
IWebSessionIDs = interface;
```

C++

```
__interface IWebSessionIDs;
```

File

SiteComp

Description

Use IWebSessionIDs to obtain the session IDs (see page 1053) for all the sessions in a sessions service.

Note: In C++, method declarations that use IWebSessionIDs use the _di_IWebSessionIDs type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebSessionIDs interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebSessionIDs > _di_IWebSessionIDs;

See Also

TSessionsService (see page 6722)

10.2.78.1 IWebSessionIDs Members

The following tables list the members exposed by IWebSessionIDs.

Public Methods

	Name	Description
	GetID (see page 1052)	Returns the value of the IDs (see page 1053) property.
	GetIDCount (see page 1052)	Returns the value of the IDCount (see page 1053) property.

Public Properties

	Name	Description
	IDCount (see page 1053)	Indicates the number of session IDs (see page 1053) available.
	IDs (see page 1053)	Lists the session IDs for all the sessions stored by the sessions service.

10.2.78.2 IWebSessionIDs Methods

The methods of the IWebSessionIDs class are listed here.

Public Methods

	Name	Description
	GetID (see page 1052)	Returns the value of the IDs (see page 1053) property.
	GetIDCount (see page 1052)	Returns the value of the IDCount (see page 1053) property.

10.2.78.2.1 IWebSessionIDs.GetID Method

Returns the value of the IDs (see page 1053) property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs (see page 1053) property.

I is the index of the session ID to return.

See Also

IDs (see page 1053)

10.2.78.2.2 IWebSessionIDs.GetIDCount Method

Returns the value of the IDCount (see page 1053) property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

GetIDCount is the read implementation of the IDCount (see page 1053) property.

See Also

IDCount (see page 1053)

10.2.78.3 IWebSessionIDs Properties

The properties of the IWebSessionIDs class are listed here.

Public Properties

	Name	Description
R	IDCount (see page 1053)	Indicates the number of session IDs (see page 1053) available.
R	IDs (see page 1053)	Lists the session IDs for all the sessions stored by the sessions service.

10.2.78.3.1 IWebSessionIDs.IDCount Property

Indicates the number of session IDs (see page 1053) available.

Pascal

```
property IDCount: Integer;
```

C++

```
__property int IDCount;
```

Description

Read (see page 3727) IDCount to obtain an upper bound for the Index parameter of the IDs (see page 1053) property. IDCount is the number of session IDs (see page 1053) (hence, the number of sessions) that the sessions service has stored in memory.

See Also

IDs (see page 1053)

10.2.78.3.2 IWebSessionIDs.IDs Property

Lists the session IDs for all the sessions stored by the sessions service.

Pascal

```
property IDs [I: Integer]: TSessionID;
```

C++

```
__property TSessionID IDs[int I];
```

Description

Use IDs to iterate over the session IDs of all the sessions that the sessions service stores.

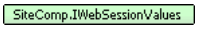
I is the index of the session ID. It can be any value from 0 to IDCount (see page 1053) – 1.

See Also

IDCount (see page 1053)

10.2.79 IWebSessionValues Interface

Class Hierarchy



Pascal

```
IWebSessionValues = interface;
```

C++

```
__interface IWebSessionValues;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionValues.

10.2.79.1 IWebSessionValues Members

The following tables list the members exposed by IWebSessionValues.

Public Methods

	Name	Description
	GetItemCount (see page 1054)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 1054)	This is GetItems, a member of class IWebSessionValues.
	GetValue (see page 1055)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 1055)	This is SetValue, a member of class IWebSessionValues.

Public Properties

	Name	Description
	Values (see page 1055)	This is Values, a member of class IWebSessionValues.

10.2.79.2 IWebSessionValues Methods

The methods of the IWebSessionValues class are listed here.

Public Methods

	Name	Description
	GetItemCount (see page 1054)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 1054)	This is GetItems, a member of class IWebSessionValues.
	GetValue (see page 1055)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 1055)	This is SetValue, a member of class IWebSessionValues.

10.2.79.2.1 IWebSessionValues.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class IWebSessionValues.

10.2.79.2.2 IWebSessionValues.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class IWebSessionValues.

10.2.79.2.3 IWebSessionValues.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const UnicodeString AName);
```

Description

This is GetValue, a member of class IWebSessionValues.

10.2.79.2.4 IWebSessionValues.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

This is SetValue, a member of class IWebSessionValues.

10.2.79.3 IWebSessionValues Properties

The properties of the IWebSessionValues class are listed here.

Public Properties

	Name	Description
	Values (see page 1055)	This is Values, a member of class IWebSessionValues.

10.2.79.3.1 IWebSessionValues.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const AName];
```

Description

This is Values, a member of class IWebSessionValues.

10.2.80 IWebSetActionName Interface

Class Hierarchy

SiteComp.IWebSetActionName

Pascal

```
IWebSetActionName = interface;
```

C++

```
__interface IWebSetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetActionName.

10.2.80.1 IWebSetActionName Members

The following tables list the members exposed by IWebSetActionName.

Public Methods

	Name	Description
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.

10.2.80.2 IWebSetActionName Methods

The methods of the IWebSetActionName class are listed here.

Public Methods

	Name	Description
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.

10.2.80.2.1 IWebSetActionName.SetActionName Method

Pascal

```
procedure SetActionName(const AFieldName: string);
```

C++

```
__fastcall SetActionName(const UnicodeString AFieldName);
```

Description

This is SetActionName, a member of class IWebSetActionName.

10.2.81 IWebSetFieldName Interface

Class Hierarchy

SiteComp.IWebSetFieldName

Pascal

```
IWebSetFieldName = interface;
```

C++

```
__interface IWebSetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetFieldName.

10.2.81.1 IWebSetFieldName Members

The following tables list the members exposed by IWebSetFieldName.

Public Methods

	Name	Description
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

10.2.81.2 IWebSetFieldName Methods

The methods of the IWebSetFieldName class are listed here.

Public Methods

	Name	Description
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

10.2.81.2.1 IWebSetFieldName.SetFieldName Method

Pascal

```
procedure SetFieldName(const AFieldName: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString AFieldName);
```

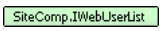
Description

This is SetFieldName, a member of class IWebSetFieldName.

10.2.82 IWebUserList Interface

IWebUserList is the interface for validating user logins and access rights.

Class Hierarchy



Pascal

```
IWebUserList = interface;
```

C++

```
__interface IWebUserList;
```

File

SiteComp

Description

The TWebUserList component implements IWebUserList. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to check whether a user supplied the correct password when logging in and whether the user has rights to perform the operation requested in an HTTP request message.

Note: In C++, method declarations that use IWebUserList use the _di_IWebUserList type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebUserList interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebUserList > _di_IWebUserList;

See Also

TWebUserList (see page 6759)

TCustomWebAppComponents (see page 6175)

TWebAPpComponents

10.2.82.1 IWebUserList Members

The following tables list the members exposed by IWebUserList.

Public Methods

	Name	Description
	CheckAccessRights (see page 1058)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 1059)	Validates a user login.

10.2.82.2 IWebUserList Methods

The methods of the IWebUserList class are listed here.

Public Methods

	Name	Description
	CheckAccessRights (see page 1058)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 1059)	Validates a user login.

10.2.82.2.1 IWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(UserID: Variant; Rights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant UserID, UnicodeString Rights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 5497)

HasModifyRights (see page 5497)

HasExecuteRights (see page 5496)

ViewAccess

ViewAccess (see page 5560)

10.2.82.2.2 IWebUserList.ValidateUser Method

Validates a user login.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

Call ValidateUser to check whether the information a user supplies on login matches a user in the user list.

Strings is a string list that lists the information that the user supplied when logging in. Each entry in the string list is of the form Name=Value. Typically, the names of strings in the list are 'UserName' and 'Password'.

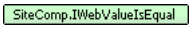
If the login information is valid, ValidateUser returns the UserID of the identified user. Otherwise, ValidateUser returns an unassigned Variant (see page 3430).

See Also

CheckAccessRights (see page 1058)

10.2.83 IWebValueIsEqual Interface

Class Hierarchy



Pascal

```
IWebValueIsEqual = interface;
```

C++

```
__interface IWebValueIsEqual;
```

File

SiteComp

Description

This is class SiteComp.IWebValueIsEqual.

10.2.83.1 IWebValueIsEqual Members

The following tables list the members exposed by IWebValueIsEqual.

Public Methods

	Name	Description
	IsEqual (see page 1060)	This is IsEqual, a member of class IWebValueIsEqual.

10.2.83.2 IWebValueIsEqual Methods

The methods of the IWebValueIsEqual class are listed here.

Public Methods

	Name	Description
	IsEqual (see page 1060)	This is IsEqual, a member of class IWebValueIsEqual.

10.2.83.2.1 IWebValueIsEqual.IsEqual Method

Pascal

```
function IsEqual(const Value: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant Value);
```

Description

This is IsEqual, a member of class IWebValueIsEqual.

10.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 1061)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 1061)	This is function SiteComp.FullyQualifiedName.

10.3.1 SiteComp.FullyQualifiedFieldName Function

Pascal

```
function FullyQualifiedFieldName(AComponent: TComponent): string;
```

C++

```
UnicodeString FullyQualifiedFieldName(TComponent * AComponent);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedFieldName.

10.3.2 SiteComp.FullyQualifiedName Function

Pascal

```
function FullyQualifiedName(AComponent: TComponent; AOption: TQualifyOption): string;
```

C++

```
UnicodeString FullyQualifiedName(TComponent * AComponent, TQualifyOption AOption);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedName.

10.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterActionHTMLElementType (see page 1062)	This is record SiteComp.TAdapterActionHTMLElementType.
	TAdapterDisplayCharacteristic (see page 1062)	This is record SiteComp.TAdapterDisplayCharacteristic.
	TAdapterDisplayHTMLElementType (see page 1063)	This is record SiteComp.TAdapterDisplayHTMLElementType.
	TAdapterDisplayViewModeType (see page 1063)	This is record SiteComp.TAdapterDisplayViewModeType.
	TAdapterInputHTMLElementType (see page 1064)	This is record SiteComp.TAdapterInputHTMLElementType.
	TAdapterItemRequestParamOption (see page 1065)	This is record SiteComp.TAdapterItemRequestParamOption.
	TDispatchPageFlag (see page 1065)	This is record SiteComp.TDispatchPageFlag.
	TQualifyOption (see page 1065)	This is record SiteComp.TQualifyOption.

10.4.1 SiteComp.TAdapterActionHTMLElementType Enumeration

Pascal

```
TAdapterActionHTMLElementType = (
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
);
```

C++

```
enum TAdapterActionHTMLElementType {
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
};
```

File

SiteComp

Description

This is record SiteComp.TAdapterActionHTMLElementType.

10.4.2 SiteComp.TAdapterDisplayCharacteristic Enumeration

Pascal

```
TAdapterDisplayCharacteristic = (
```

```
    dcMultipleRecordView,
    dcCurrentRecordView,
    dcChangeCurrentRecordView
  );
```

C++

```
  enum TAdapterDisplayCharacteristic {
    dcMultipleRecordView,
    dcCurrentRecordView,
    dcChangeCurrentRecordView
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayCharacteristic.

10.4.3 SiteComp.TAdapterDisplayHTMLElementType Enumeration

Pascal

```
  TAdapterDisplayHTMLElementType = (
    htmldText,
    htmldImage,
    htmldList,
    htmldNone
  );
```

C++

```
  enum TAdapterDisplayHTMLElementType {
    htmldText,
    htmldImage,
    htmldList,
    htmldNone
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayHTMLElementType.

10.4.4 SiteComp.TAdapterDisplayViewModeType Enumeration

Pascal

```
  TAdapterDisplayViewModeType = (
    htmlvmInput,
    htmlvmDisplay,
    htmlvmToggleOnAccess,
```

```
    htmlvmNone
  );
```

C++

```
  enum TAdapterDisplayViewModeType {
    htmlvmInput,
    htmlvmDisplay,
    htmlvmToggleOnAccess,
    htmlvmNone
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayViewModeType.

10.4.5 SiteComp.TAdapterInputHTMLElementType Enumeration

Pascal

```
  TAdapterInputHTMLElementType = (
    htmliNone,
    htmliTextInput,
    htmliPasswordInput,
    htmliSelect,
    htmliSelectMultiple,
    htmliRadio,
    htmliCheckBox,
    htmliTextArea,
    htmliFile
  );
```

C++

```
  enum TAdapterInputHTMLElementType {
    htmliNone,
    htmliTextInput,
    htmliPasswordInput,
    htmliSelect,
    htmliSelectMultiple,
    htmliRadio,
    htmliCheckBox,
    htmliTextArea,
    htmliFile
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterInputHTMLElementType.

10.4.6 SiteComp.TAdapterItemRequestParamOption Enumeration

Pascal

```
TAdapterItemRequestParamOption = (
  roExcludeID
);
```

C++

```
enum TAdapterItemRequestParamOption {
  roExcludeID
};
```

File

SiteComp

Description

This is record SiteComp.TAdapterItemRequestParamOption.

10.4.7 SiteComp.TDispatchPageFlag Enumeration

Pascal

```
TDispatchPageFlag = (
  dpPublished
);
```

C++

```
enum TDispatchPageFlag {
  dpPublished
};
```

File

SiteComp

Description

This is record SiteComp.TDispatchPageFlag.

10.4.8 SiteComp.TQualifyOption Enumeration

Pascal

```
TQualifyOption = (
  qaUseModulesVar,
  qaNoModulesVar,
  qaNoModuleName
);
```

C++

```
enum TQualifyOption {
  qaUseModulesVar,
  qaNoModulesVar,
  qaNoModuleName
};
```

File

SiteComp

Description

This is record SiteComp.TQualifyOption.

10.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 1067)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 1067)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 1068)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 1068)	This is type SiteComp.TSessionID.

10.5.1 SiteComp.TAdapterDisplayCharacteristics Type

Pascal

```
TAdapterDisplayCharacteristics = set of TAdapterDisplayCharacteristic;
```

C++

```
TAdapterDisplayCharacteristic TAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is type SiteComp.TAdapterDisplayCharacteristics.

10.5.2 SiteComp.TAdapterItemRequestParamOptions Type

Pascal

```
TAdapterItemRequestParamOptions = set of TAdapterItemRequestParamOption;
```

C++

```
TAdapterItemRequestParamOption TAdapterItemRequestParamOptions;
```

File

SiteComp

Description

This is type SiteComp.TAdapterItemRequestParamOptions.

10.5.3 SiteComp.TDispatchPageFlags Type

Pascal

```
TDispatchPageFlags = set of TDispatchPageFlag;
```

C++

```
TDispatchPageFlag TDispatchPageFlags;
```

File

SiteComp

Description

This is type SiteComp.TDispatchPageFlags.

10.5.4 SiteComp.TSessionID Type

Pascal

```
TSessionID = type string;
```

C++

```
type string TSessionID;
```

File

SiteComp

Description

This is type SiteComp.TSessionID.

10.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ExecuteScript (see page 1069)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 1069)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 1070)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 1070)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 1070)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 1071)	This is variable SiteComp.ScriptDebugger.

10.6.1 SiteComp.ExecuteScript Variable

Pascal

```
ExecuteScript: IInternetExecuteScript = nil;
```

C++

```
IInternetExecuteScript ExecuteScript = nil;
```

File

SiteComp

Description

This is variable SiteComp.ExecuteScript.

10.6.2 SiteComp.GetHTMLSampleImage Variable

Pascal

```
GetHTMLSampleImage: function: string = nil;
```

C++

```
function: string GetHTMLSampleImage = nil;
```

File

SiteComp

Description

This is variable SiteComp.GetHTMLSampleImage.

10.6.3 SiteComp.IDEFileManager Variable

Pascal

```
IDEFileManager: IIDEFileManager = nil;
```

C++

```
IIDEFileManager IDEFileManager = nil;
```

File

SiteComp

Description

This is variable SiteComp.IDEFileManager.

10.6.4 SiteComp.InetFileTypes Variable

Pascal

```
InetFileTypes: IInternetFileTypes = nil;
```

C++

```
IInternetFileTypes InetFileTypes = nil;
```

File

SiteComp

Description

This is variable SiteComp.InetFileTypes.

10.6.5 SiteComp.InternetEnvOptions Variable

Pascal

```
InternetEnvOptions: IInternetEnvOptions = nil;
```

C++

```
IInternetEnvOptions InternetEnvOptions = nil;
```

File

SiteComp

Description

This is variable SiteComp.InternetEnvOptions.

10.6.6 SiteComp.ScriptDebugger Variable

Pascal

```
ScriptDebugger: IInternetScriptDebugger = nil;
```

C++

```
IInternetScriptDebugger ScriptDebugger = nil;
```

File

SiteComp

Description

This is variable SiteComp.ScriptDebugger.

10.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 1072)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 1072)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 1073)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 1073)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

10.7.1 SiteComp.AdapterActionHTMLElementTypeNames Constant

Pascal

```
AdapterActionHTMLElementTypeNames: array[TAdapterActionHTMLElementType] of string = ('Button',
'Image', 'Anchor', 'EventImages');
```

C++

```
array[TAdapterActionHTMLElementType] of string AdapterActionHTMLElementTypeNames = ('Button',
'Image', 'Anchor', 'EventImages');
```

File

SiteComp

Description

This is constant SiteComp.AdapterActionHTMLElementTypeNames.

10.7.2 SiteComp.AdapterDisplayHTMLElementTypeNames Constant

Pascal

```
AdapterDisplayHTMLElementTypeNames: array[TAdapterDisplayHTMLElementType] of string = ('Text',
'Image', 'List', '');
```

C++

```
array[TAdapterDisplayHTMLElementType] of string AdapterDisplayHTMLElementTypeNames = ('Text',
'Image', 'List', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.

10.7.3 SiteComp.AdapterDisplayViewModeTypeNames Constant

Pascal

```
AdapterDisplayViewModeTypeNames: array[TAdapterDisplayViewModeType] of string = ('Input',
'Display', 'ToggleOnAccess', '');
```

C++

```
array[TAdapterDisplayViewModeType] of string AdapterDisplayViewModeTypeNames = ('Input',
'Display', 'ToggleOnAccess', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayViewModeTypeNames.

10.7.4 SiteComp.AdapterInputHTMLElementTypeNames Constant

Pascal

```
AdapterInputHTMLElementTypeNames: array[TAdapterInputHTMLElementType] of string = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

C++

```
array[TAdapterInputHTMLElementType] of string AdapterInputHTMLElementTypeNames = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

File

SiteComp

Description

This is constant SiteComp.AdapterInputHTMLElementTypeNames.

11 SiteProd Namespace

This is namespace SiteProd.

Topics

Name	Description
Classes (see page 1076)	The following table lists classes in this documentation.

Classes

	Name	Description
	TAdapterPageProducer (see page 1076)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 1083)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 1099)	tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 1107)	TSitePageElements stores part of the content that page producers generate.

11.1 Classes

The following table lists classes in this documentation.

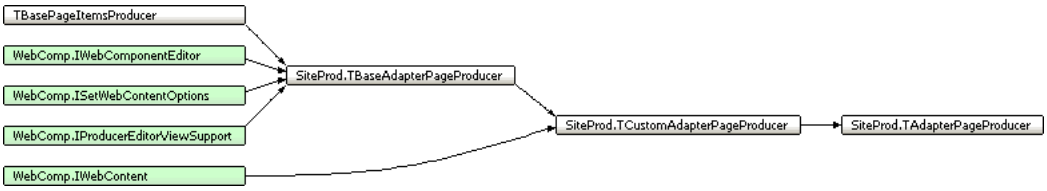
Classes

	Name	Description
	TAdapterPageProducer (see page 1076)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 1083)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 1099)	tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 1107)	TSitePageElements stores part of the content that page producers generate.

11.1.1 TAdapterPageProducer Class

TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TAdapterPageProducer = class(TCustomAdapterPageProducer);
```

C++

```
class TAdapterPageProducer : public TCustomAdapterPageProducer;
```

File

SiteProd

Description

TAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. Add components to this hierarchy to define HTML forms and tables. For example, to create an HTML form with a grid, add a TAdapterForm (see page 6373) and TAdapterGrid (see page 6386) components. The Web items that appear in the TAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use the Web Page Module wizard to create a web page that uses the TAdapterPageProducer. You can select TAdapterPageProducer from the drop-down list at the top of the Web Page Module wizard, or add a TAdapterPageProducer component to an existing web page modue.

Web page modules that use the TAdapterPageProducer provide an editor view called HTML Script. Use this view to see the HTML Script that TAdapterPageProducer generates. The script generated by TAdapterPageProducer is not editable. However, you can copy the generated script from the HTML Script view to a new Web Page Module. You can then edit the copy of the

script in the code editor or with an external HTML editor that supports embedded script tags (for example <% %>).

TAdapterPageProducer uses a template that contains the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1080) or StylesFile (see page 1081) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with the generated HTML/JavaScript. The TAdapterPageProducer will not generate any content if this tag is missing.

See Also

TAdapter (see page 5179)

TAdapterForm (see page 6373)

TAdapterGrid (see page 6386)

Page Producers

Server-side Scripting in WebSnap

11.1.1.1 TAdapterPageProducer Members

The following tables list the members exposed by TAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1086)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1087)	Creates the object that implements the PageElements (see page 1097) property.
V	DeclareStyles (see page 1088)	Returns an HTML Styles (see page 1097) tag that declares the styles specified by the Styles (see page 1097) or StylesFile (see page 1098) property.
V	DeclareWarnings (see page 1088)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1088)	Generates an OnAfterGetContent (see page 1098) event.
	DoBeforeGetContent (see page 1089)	Generates an OnBeforeGetContent (see page 1099) event.
V	DoTagEvent (see page 1089)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1090)	Fills the PageElements (see page 1097) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1090)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1090)	Returns the default template that the page producer can use to generate content.
V	GetTemplateFileName (see page 1102)	Returns the value of the HTMLFile (see page 1106) property.
	GetTemplateFileType (see page 1092)	Returns the type of file that can store the templates this object uses to generate its content.
V	GetTemplateStream (see page 1103)	Returns a stream from which to read the HTML template that this producer uses.
	GetXMLData (see page 1092)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1092)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1093)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1093)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1094)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1094)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1094)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.

	ImplCanAddClass (see page 1095)	Provides the underlying implementation for the CanAddClass (see page 1086) method.
	SetHTMLDoc (see page 1103)	Sets the HTMLDoc (see page 1105) property.
	SetHTMLFile (see page 1104)	Sets the HTMLFile (see page 1106) property.
	SetWebContentOptions (see page 1096)	Sets the options that limit how the page producer creates its generated HTML.
	WebContent (see page 1104)	Returns the HTML document assembled by the Adapter page producer.

Protected Properties

	Name	Description
	PageElements (see page 1097)	Stores information that the Adapter page producer uses when generating its content.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
	Content (see page 6076)	Returns the HTML that the Web item generates.
	ContentFromStream (see page 1087)	Returns the content produced using a template read from a stream.
	Create (see page 1085)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 1083).
	Create (see page 1101)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 1099).
	Destroy (see page 1085)	Frees an instance of TBaseAdapterPageProducer (see page 1083).
	Destroy (see page 1102)	Frees an instance of TCustomAdapterPageProducer (see page 1099).
	GetRequiredTags (see page 1091)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1091)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	SetStyles (see page 1095)	Assigns a value to the Styles (see page 1097) property.
	SetStylesFile (see page 1096)	Assigns a value to the StylesFile (see page 1098) property.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

Published Events

	Name	Description
	OnAfterGetContent (see page 1082)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1082)	Occurs before the page producer generates its content.

Published Properties

	Name	Description
	HTMLDoc (see page 1079)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1080)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1080)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1081)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 1081)	Maintains a list of components that generate HTML content for the producer.

11.1.1.2 TAdapterPageProducer Properties

The properties of the TAdapterPageProducer class are listed here.

Published Properties

	Name	Description
	HTMLDoc (see page 1079)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1080)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1080)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1081)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 1081)	Maintains a list of components that generate HTML content for the producer.

11.1.1.2.1 TAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles or StylesFile property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content

HTMLFile (see page 1106)

OnHTMLTag

11.1.1.2.2 TAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content

HTMLDoc (see page 1105)

OnHTMLTag

11.1.1.2.3 TAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer. Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile property instead. If StylesFile is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 1098)

Content

11.1.1.2.4 TAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles property instead.

See Also

Styles (see page 1097)

Content

11.1.1.2.5 TAdapterPageProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems;
```

C++

```
__property WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer. Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent (see page 6075)

Creating Web Pages with an InternetExpress Page Producer

11.1.1.3 TAdapterPageProducer Events

The events of the TAdapterPageProducer class are listed here.

Published Events

	Name	Description
	OnAfterGetContent (see page 1082)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1082)	Occurs before the page producer generates its content.

11.1.1.3.1 TAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 1087)

OnBeforeGetContent (see page 1099)

11.1.1.3.2 TAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

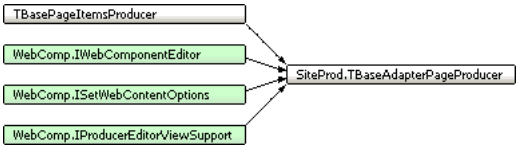
ContentFromStream (see page 1087)

OnAfterGetContent (see page 1098)

11.1.2 TBaseAdapterPageProducer Class

TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TBaseAdapterPageProducer = class(TBasePageItemsProducer, IWebComponentEditor,
ISetWebContentOptions, IProducerEditorViewSupport);
```

C++

```
class TBaseAdapterPageProducer : public TBasePageItemsProducer, public IWebComponentEditor,
public ISetWebContentOptions, public IProducerEditorViewSupport;
```

File

SiteProd

Description

TBaseAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the TBaseAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use TBaseAdapterPageProducer as a base class for page producer components that assemble their content from Web items that use Adapter components. TBaseAdapterPageProducer implements the logic for converting an HTML template into content it assembles from its hierarchy of Web items.

TBaseAdapterPageProducer implements the logic to translate the following special HTML-transparent tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1097) or StylesFile (see page 1098) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

11.1.2.1 TBaseAdapterPageProducer Members

The following tables list the members exposed by TBaseAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1086)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1087)	Creates the object that implements the PageElements (see page 1097) property.
V	DeclareStyles (see page 1088)	Returns an HTML Styles (see page 1097) tag that declares the styles specified by the Styles (see page 1097) or StylesFile (see page 1098) property.
V	DeclareWarnings (see page 1088)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1088)	Generates an OnAfterGetContent (see page 1098) event.
	DoBeforeGetContent (see page 1089)	Generates an OnBeforeGetContent (see page 1099) event.
V	DoTagEvent (see page 1089)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1090)	Fills the PageElements (see page 1097) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1090)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1090)	Returns the default template that the page producer can use to generate content.
	GetTemplateFileType (see page 1092)	Returns the type of file that can store the templates this object uses to generate its content.
	GetXMLData (see page 1092)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1092)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1093)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1093)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1094)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1094)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1094)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1095)	Provides the underlying implementation for the CanAddClass (see page 1086) method.
	SetWebContentOptions (see page 1096)	Sets the options that limit how the page producer creates its generated HTML.

Protected Properties

	Name	Description
R	PageElements (see page 1097)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 1098)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1099)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
V	ContentFromStream (see page 1087)	Returns the content produced using a template read from a stream.
V	Create (see page 1085)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 1083).
V	Destroy (see page 1085)	Frees an instance of TBaseAdapterPageProducer (see page 1083).
	GetRequiredTags (see page 1091)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.

	GetScriptEngines (see page 1091)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	SetStyles (see page 1095)	Assigns a value to the Styles (see page 1097) property.
	SetStylesFile (see page 1096)	Assigns a value to the StylesFile (see page 1098) property.

Public Properties

	Name	Description
	Styles (see page 1097)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1098)	Specifies a file containing the individual styles for the page producer's style sheet.

11.1.2.2 TBaseAdapterPageProducer.Create Constructor

Creates and initializes an instance of TBaseAdapterPageProducer (see page 1083).

Pascal

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

C++

```
virtual __fastcall TBaseAdapterPageProducer(TComponent * AComponent);
```

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TBaseAdapterPageProducer (see page 1083) instance.

See Also

Owner

11.1.2.3 TBaseAdapterPageProducer.Destroy Destructor

Frees an instance of TBaseAdapterPageProducer (see page 1083).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

11.1.2.4 TBaseAdapterPageProducer Methods

The methods of the TBaseAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 1086)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1087)	Creates the object that implements the PageElements (see page 1097) property.
V	DeclareStyles (see page 1088)	Returns an HTML Styles (see page 1097) tag that declares the styles specified by the Styles (see page 1097) or StylesFile (see page 1098) property.
V	DeclareWarnings (see page 1088)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1088)	Generates an OnAfterGetContent (see page 1098) event.
	DoBeforeGetContent (see page 1089)	Generates an OnBeforeGetContent (see page 1099) event.
V	DoTagEvent (see page 1089)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1090)	Fills the PageElements (see page 1097) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1090)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1090)	Returns the default template that the page producer can use to generate content.
	GetTemplateFileType (see page 1092)	Returns the type of file that can store the templates this object uses to generate its content.
	GetXMLData (see page 1092)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1092)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1093)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1093)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1094)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1094)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1094)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1095)	Provides the underlying implementation for the CanAddClass (see page 1086) method.
	SetWebContentOptions (see page 1096)	Sets the options that limit how the page producer creates its generated HTML.

Public Methods

	Name	Description
V	ContentFromStream (see page 1087)	Returns the content produced using a template read from a stream.
	GetRequiredTags (see page 1091)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1091)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	SetStyles (see page 1095)	Assigns a value to the Styles (see page 1097) property.
	SetStylesFile (see page 1096)	Assigns a value to the StylesFile (see page 1098) property.

11.1.2.4.1 TBaseAdapterPageProducer.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebPageItems property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to supply content for the TBaseAdapterPageProducer (see page 1083) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 1095) method to determine whether a class of Web item is valid.

AParent is the TBaseAdapterPageProducer (see page 1083) descendant to whose WebPageItems property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the TBaseAdapterPageProducer (see page 1083) descendant can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 1095)

WebPageItems

11.1.2.4.2 TBaseAdapterPageProducer.ContentFromStream Method

Returns the content produced using a template read from a stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; override;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream InStream);
```

Description

Call ContentFromStream to retrieve the content produced by the adapter page producer based on the template provided by the Stream parameter. ContentFromStream:

1. Generates an OnBeforeGetContent (see page 1099) event.
2. Translates the HTML template that is the value of Stream, using information produced by the items listed in the WebPageItems property.
3. Generates an OnAfterGetContent (see page 1098) event.
4. Returns the HTML produced in step 2.

See Also

Content

OnBeforeGetContent (see page 1099)

WebPageItems

OnAfterGetContent (see page 1098)

11.1.2.4.3 TBaseAdapterPageProducer.CreatePageElements Method

Creates the object that implements the PageElements (see page 1097) property.

Pascal

```
function CreatePageElements: TSitePageElements; virtual;
```

C++

```
virtual __fastcall TSitePageElements CreatePageElements();
```

Description

CreatePageElements creates the object that is used as the value of the PageElements (see page 1097) property. This object holds temporary information that the page producer uses to assemble its content.

See Also

PageElements (see page 1097)

GeneratePageElements (see page 1090)

11.1.2.4.4 TBaseAdapterPageProducer.DeclareStyles Method

Returns an HTML Styles (see page 1097) tag that declares the styles specified by the Styles (see page 1097) or StylesFile (see page 1098) property.

Pascal

```
function DeclareStyles: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareStyles();
```

Description

The GeneratePageElements (see page 1090) method calls DeclareStyles to generate the HTML that declares a stylesheet in the generated HTML page. DeclareStyles uses the StylesFile (see page 1098) property to determine the styles it declares. If StylesFile (see page 1098) is an empty string, DeclareStyles uses the Styles (see page 1097) property instead. If both StylesFile (see page 1098) and Styles (see page 1097) are set to empty strings, DeclareStyles returns an empty string.

See Also

Styles (see page 1097)

StylesFile (see page 1098)

GeneratePageElements (see page 1090)

11.1.2.4.5 TBaseAdapterPageProducer.DeclareWarnings Method

Returns the formatted warnings that appear at design time to indicate the status of content generation.

Pascal

```
function DeclareWarnings(Warnings: TStrings): string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareWarnings(TStrings Warnings);
```

Description

The GeneratePageElements (see page 1090) method calls DeclareWarnings to generate the text that describes any design-time warnings. These warnings indicate possible problems with the hierarchy of web items.

See Also

GeneratePageElements (see page 1090)

WebPageItems

11.1.2.4.6 TBaseAdapterPageProducer.DoAfterGetContent Method

Generates an OnAfterGetContent (see page 1098) event.

Pascal

```
procedure DoAfterGetContent;
```

C++

```
__fastcall DoAfterGetContent();
```

Description

TBaseAdapterPageProducer (see page 1083) calls DoAfterGetContent internally to generate the OnAfterGetContent (see page 1098) event.

See Also

OnAfterGetContent (see page 1098)

11.1.2.4.7 TBaseAdapterPageProducer.DoBeforeGetContent Method

Generates an OnBeforeGetContent (see page 1099) event.

Pascal

```
procedure DoBeforeGetContent;
```

C++

```
__fastcall DoBeforeGetContent();
```

Description

TBaseAdapterPageProducer (see page 1083) calls DoBeforeGetContent internally to generate the OnBeforeGetContent (see page 1099) event.

See Also

OnBeforeGetContent (see page 1099)

11.1.2.4.8 TBaseAdapterPageProducer.DoTagEvent Method

Translates HTML-transparent tags in the template.

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceText);
```

Description

When the page producer encounters an HTML-transparent tag in its template, it calls DoTagEvent to translate that tag into the appropriate value. TBaseAdapterPageProducer (see page 1083) overrides the inherited method to handle the <#STYLES>, <#WARNINGS>, and <#SERVERSCRIPT> tags. For any other tags, DoTagEvent generates an OnHTMLTag event.

Tag is the type of HTML-transparent tag that was encountered in the HTML template.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

See Also

OnHTMLTag

11.1.2.4.9 TBaseAdapterPageProducer.GeneratePageElements Method

Fills the PageElements (see page 1097) property with information based on the other properties of the adapter Page Producer.

Pascal

```
procedure GeneratePageElements; virtual;
```

C++

```
virtual __fastcall GeneratePageElements();
```

Description

GeneratePageElements assembles information about the styles, warnings, server script, server functions, and server variables that the adapter page producer uses to generate its content. It writes this information into the members of the object that is the value of the PageElements (see page 1097) property.

See Also

PageElements (see page 1097)

11.1.2.4.10 TBaseAdapterPageProducer.GetContentOptions Method

Returns an object that influences how the page producer generates its content.

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; override;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

TBaseAdapterPageProducer (see page 1083) calls GetContentOptions internally to look up the options that limit what can be included in the generated HTML.

Owned returns a value that indicates whether the return value should be freed by the caller. If Owned is true, the caller should free the TWebContentOptions (see page 6049) object after it has used it.

See Also

SetWebContentOptions (see page 1096)

11.1.2.4.11 TBaseAdapterPageProducer.GetDefaultTemplate Method

Returns the default template that the page producer can use to generate content.

Pascal

```
function GetDefaultTemplate: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultTemplate();
```

Description

GetDefaultTemplate returns the default template that the page producer can use to generate its content. This template has the form:

```
<HTML>
<HEAD>
```

```
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

TBaseAdapterPageProducer (see page 1083) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1097) or StylesFile (see page 1098) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

See Also

Styles (see page 1097)

StylesFile (see page 1098)

11.1.2.4.12 TBaseAdapterPageProducer.GetRequiredTags Method

Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.

Pascal

```
class function GetRequiredTags: string;
```

C++

```
__fastcall UnicodeString GetRequiredTags();
```

Description

GetRequiredTags returns a string containing the special tags that the page producer expects to find for generating its content. This method returns the string '<#STYLES> <#WARNINGS> <#SERVERSCRIPT>'.

See Also

ContentFromStream (see page 1087)

11.1.2.4.13 TBaseAdapterPageProducer.GetScriptEngines Method

Returns a list of script engines that can work with server-side scripts that produce content for the page producer.

Pascal

```
class procedure GetScriptEngines(S: TStrings);
```

C++

```
__fastcall GetScriptEngines(TStrings S);
```

Description

GetScriptEngines fills an existing string list with the set of all values that can be assigned to the ScriptEngine property. These determine the script engine that works with associated server-side scripts.

This method adds a single string, 'JSCRIPT', to the string list passed as the S parameter.

See Also

ScriptEngine

11.1.2.4.14 TBaseAdapterPageProducer.GetTemplateFileType Method

Returns the type of file that can store the templates this object uses to generate its content.

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

GetTemplateFileType returns the string 'HTML', because Adapter page producers generate their contents based on an HTML template that includes HTML-transparent tags.

See Also

GetDefaultTemplate (see page 1090)

11.1.2.4.15 TBaseAdapterPageProducer.GetXMLData Method

Returns a stream for reading any XML data associated with the page producer.

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

The IDE calls GetXMLData for producers that indicate that they support an XML browser view (that is, if the HasXMLBrowserView (see page 1094) method returns true). In TBaseAdapterPageProducer (see page 1083), GetXMLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use associated XML data.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant, because GetXMLData always returns nil (Delphi) or NULL (C++).

See Also

HasXMLBrowserView (see page 1094)

11.1.2.4.16 TBaseAdapterPageProducer.GetXSLData Method

Returns a stream for reading an XSL template associated with the page producer.

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

The IDE calls GetXSLData for producers that indicate that they support an XSL browser view (that is, if the HasXSLBrowserView (see page 1094) method returns true). In TBaseAdapterPageProducer (see page 1083), GetXSLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use an associated XSL template.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant,

because GetXSLData always returns nil (Delphi) or NULL (C++).

See Also

HasXSLBrowserView (see page 1094)

11.1.2.4.17 TBaseAdapterPageProducer.HandleScriptTag Method

Translates HTML-transparent tags for an associated server-side script.

Pascal

```
function HandleScriptTag(const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandleScriptTag(const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceString);
```

Description

When a server-side script encounters an HTML-transparent tag, it calls HandleScriptTag to allow the page producer to translate that tag for it. HandleScriptTag translates the <#SERVERSCRIPT> tag, supplying the content produced by the hierarchy of Web items that is the value of the WebPageItems property.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

HandleScriptTag returns true if it can translate the tag (that is, if TagString is 'SERVERSCRIPT'). It returns false if TagString identifies any other type of tag.

See Also

WebPageItems

11.1.2.4.18 TBaseAdapterPageProducer.HasHTMLBrowserView Method

Indicates whether the IDE should display an HTML browser view for the producer's content.

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

The IDE calls HasHTMLBrowserView to determine whether it should create an HTML tab page in the code editor for displaying the content of the page producer.

HasHTMLBrowserView returns true. As a result, at design time, the value that the Content method returns appears on the HTML tab page of the code editor when the adapter page producer is selected.

See Also

HasScriptView (see page 1094)

HasXMLBrowserView (see page 1094)

HasXSLBrowserView (see page 1094)

Content

11.1.2.4.19 TBaseAdapterPageProducer.HasScriptView Method

Indicates whether the IDE should display an associated server-side script.

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

The IDE calls HasScriptView to determine whether it should create a script tab page in the code editor for displaying the server-side script associated with the page producer.

HasScriptView returns true. As a result, at design time, the associated server-side script appears on a tab page of the code editor when the adapter page producer is selected.

See Also

HasHTMLBrowserView (see page 1093)

HasXMLBrowserView (see page 1094)

HasXSLBrowserView (see page 1094)

11.1.2.4.20 TBaseAdapterPageProducer.HasXMLBrowserView Method

Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

The IDE calls HasXMLBrowserView to determine whether it should create an XML tab page in the code editor for displaying the XML data that the page producer uses.

HasXMLBrowserView returns false because adapter page producers do not use associated XML data.

See Also

HasScriptView (see page 1094)

HasHTMLBrowserView (see page 1093)

HasXSLBrowserView (see page 1094)

GetXMLData (see page 1092)

11.1.2.4.21 TBaseAdapterPageProducer.HasXSLBrowserView Method

Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

The IDE calls HasXSLBrowserView to determine whether it should create an XSL tab page in the code editor for displaying the XSL template that the page producer uses.

HasXSLBrowserView returns false because adapter page producers do not use associated XSL templates.

See Also

HasScriptView (see page 1094)

HasHTMLBrowserView (see page 1093)

HasXMLBrowserView (see page 1094)

GetXSLData (see page 1092)

11.1.2.4.22 TBaseAdapterPageProducer.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 1086) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 1086) method calls ImpleCanAddClass to determine whether the TBaseAdapterPageProducer (see page 1083) descendant can add a component of type AClass to its WebPageItems property list. AParent is the TBaseAdapterPageProducer (see page 1083) instance to which the component will be added.

In TBaseAdapterPageProducer (see page 1083), ImplCanAddClass returns true if AClass descends from TCustomAdapterForm (see page 6456) or TCustomLayoutGroup.

See Also

CanAddClass (see page 1086)

WebPageItems

TCustomAdapterForm (see page 6456)

TCustomLayoutGroup

11.1.2.4.23 TBaseAdapterPageProducer.SetStyles Method

Assigns a value to the Styles (see page 1097) property.

Pascal

```
procedure SetStyles(Value: TStrings);
```

C++

```
__fastcall SetStyles(TStrings Value);
```

Description

SetStyles is the write implementation of the Styles (see page 1097) property.

See Also

Styles (see page 1097)

11.1.2.4.24 TBaseAdapterPageProducer.SetStylesFile Method

Assigns a value to the StylesFile (see page 1098) property.

Pascal

```
procedure SetStylesFile(const Value: TFileName);
```

C++

```
__fastcall SetStylesFile(const TFileName Value);
```

Description

SetStylesFile is the write implementation of the StylesFile (see page 1098) property.

See Also

StylesFile (see page 1098)

11.1.2.4.25 TBaseAdapterPageProducer.SetWebContentOptions Method

Sets the options that limit how the page producer creates its generated HTML.

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

SetWebContentOptions assigns a set of flags to the adapter page producer that limit what it can include in generated HTML.

AOptions is the TWebContentOptions (see page 6049) object that represents the set of options for this page producer. TBaseAdapterPageProducer (see page 1083) takes ownership of the object, so the caller should not free it.

See Also

GetContentOptions (see page 1090)

11.1.2.5 TBaseAdapterPageProducer Properties

The properties of the TBaseAdapterPageProducer class are listed here.

Protected Properties

	Name	Description
	PageElements (see page 1097)	Stores information that the Adapter page producer uses when generating its content.

Public Properties

	Name	Description
	Styles (see page 1097)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1098)	Specifies a file containing the individual styles for the page producer's style sheet.

11.1.2.5.1 TBaseAdapterPageProducer.PageElements Property

Stores information that the Adapter page producer uses when generating its content.

Pascal

```
property PageElements: TSitePageElements;
```

C++

```
__property TSitePageElements PageElements;
```

Description

TBaseAdapterPageProducer (see page 1083) uses PageElements internally to store styles, warnings, server script, server functions, and server variables. It uses the properties of PageElements to assemble its content.

See Also

Content

OnHTMLTag

11.1.2.5.2 TBaseAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer (see page 1083). Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile (see page 1098) property instead. If StylesFile (see page 1098) is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 1098)

Content

11.1.2.5.3 TBaseAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles (see page 1097) for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles (see page 1097) property instead.

See Also

Styles (see page 1097)

Content

11.1.2.6 TBaseAdapterPageProducer Events

The events of the TBaseAdapterPageProducer class are listed here.

Public Events

	Name	Description
	OnAfterGetContent (see page 1098)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1099)	Occurs before the page producer generates its content.

11.1.2.6.1 TBaseAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write (see page 3789) an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 1087)

OnBeforeGetContent (see page 1099)

11.1.2.6.2 TBaseAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write (see page 3789) an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

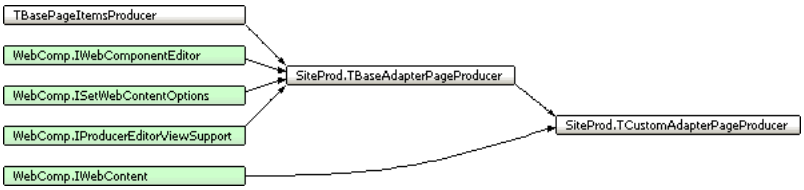
ContentFromStream (see page 1087)

OnAfterGetContent (see page 1098)

11.1.3 TCustomAdapterPageProducer Class

tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterPageProducer = class(TBaseAdapterPageProducer, IWebContent);
```

C++

```
class TCustomAdapterPageProducer : public TBaseAdapterPageProducer, public IWebContent;
```

File

SiteProd

Description

tcustomadapterpageproducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the tcustomadapterpageproducer's hierarchy use Adapter components to fetch the information that they display.

Use tcustomadapterpageproducer as a base class for page producer components that assemble their content from Web items

that use Adapter components.

tcustomadapterpageproducer introduces two properties, HTMLDoc (see page 1105) and HTMLFile (see page 1106), each of which can specify a template that the page producer uses to generate its content. By default, this template is set to include the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1097) or StylesFile (see page 1098) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. tcustomadapterpageproducer will not generate any content if this tag is missing.

Applications can alter the template to further embellish the content that the page producer generates.

11.1.3.1 TCustomAdapterPageProducer Members

The following tables list the members exposed by TCustomAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1086)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1087)	Creates the object that implements the PageElements (see page 1097) property.
V	DeclareStyles (see page 1088)	Returns an HTML Styles (see page 1097) tag that declares the styles specified by the Styles (see page 1097) or StylesFile (see page 1098) property.
V	DeclareWarnings (see page 1088)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1088)	Generates an OnAfterGetContent (see page 1098) event.
	DoBeforeGetContent (see page 1089)	Generates an OnBeforeGetContent (see page 1099) event.
V	DoTagEvent (see page 1089)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1090)	Fills the PageElements (see page 1097) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1090)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1090)	Returns the default template that the page producer can use to generate content.
V	GetTemplateFileName (see page 1102)	Returns the value of the HTMLFile (see page 1106) property.
	GetTemplateFileType (see page 1092)	Returns the type of file that can store the templates this object uses to generate its content.
V	GetTemplateStream (see page 1103)	Returns a stream from which to read the HTML template that this producer uses.
	GetXMLData (see page 1092)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1092)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1093)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1093)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1094)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1094)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1094)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1095)	Provides the underlying implementation for the CanAddClass (see page 1086) method.
	SetHTMLDoc (see page 1103)	Sets the HTMLDoc (see page 1105) property.
	SetHTMLFile (see page 1104)	Sets the HTMLFile (see page 1106) property.
	SetWebContentOptions (see page 1096)	Sets the options that limit how the page producer creates its generated HTML.
	WebContent (see page 1104)	Returns the HTML document assembled by the Adapter page producer.

Protected Properties

	Name	Description
	PageElements (see page 1097)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 1098)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1099)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
	Content (see page 6076)	Returns the HTML that the Web item generates.
	ContentFromStream (see page 1087)	Returns the content produced using a template read from a stream.
	Create (see page 1101)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 1099).
	Destroy (see page 1102)	Frees an instance of TCustomAdapterPageProducer (see page 1099).
	GetRequiredTags (see page 1091)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1091)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	SetStyles (see page 1095)	Assigns a value to the Styles (see page 1097) property.
	SetStylesFile (see page 1096)	Assigns a value to the StylesFile (see page 1098) property.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

Public Properties

	Name	Description
	HTMLDoc (see page 1105)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1106)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1097)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1098)	Specifies a file containing the individual styles for the page producer's style sheet.

11.1.3.2 TCustomAdapterPageProducer.Create Constructor

Creates and initializes an instance of TCustomAdapterPageProducer (see page 1099).

Pascal

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

C++

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

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TCustomAdapterPageProducer (see page 1099) instance.

See Also

Owner

11.1.3.3 TCustomAdapterPageProducer.Destroy Destructor

Frees an instance of TCustomAdapterPageProducer (see page 1099).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

11.1.3.4 TCustomAdapterPageProducer Methods

The methods of the TCustomAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 1102)	Returns the value of the HTMLFile (see page 1106) property.
	GetTemplateStream (see page 1103)	Returns a stream from which to read the HTML template that this producer uses.
	SetHTMLDoc (see page 1103)	Sets the HTMLDoc (see page 1105) property.
	SetHTMLFile (see page 1104)	Sets the HTMLFile (see page 1106) property.
	WebContent (see page 1104)	Returns the HTML document assembled by the Adapter page producer.

11.1.3.4.1 TCustomAdapterPageProducer.GetTemplateFileName Method

Returns the value of the HTMLFile (see page 1106) property.

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetTemplateFileName();
```

Description

TBasePageProducer introduces the protected GetTemplateFileName method to obtain the name of a file that contains a template from which to produce content. Each page producer component overrides GetTemplateFileName to supply this template in a class-specific way. TCustomAdapterPageProducer (see page 1099) returns the value of the HTMLFile (see page 1106) property.

See Also

GetTemplateStream (see page 1103)

HTMLFile (see page 1106)

11.1.3.4.2 TCustomAdapterPageProducer.GetTemplateStream Method

Returns a stream from which to read the HTML template that this producer uses.

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

TBasePageProducer introduces the protected GetTemplateStream method to obtain a stream from which to read the producer component's HTML template. Each page producer component overrides GetTemplateStream to supply this stream in a class-specific way.

AOwned returns a value that indicates whether the caller should free the returned stream after reading the template. When AOwned is true, the caller should free the stream object.

TCustomAdapterPageProducer (see page 1099) returns a stream from which the template specified by the HTMLFile (see page 1106) or HTMLDoc (see page 1105) property can be read.

See Also

GetTemplateFileName (see page 1102)

HTMLFile (see page 1106)

HTMLDoc (see page 1105)

11.1.3.4.3 TCustomAdapterPageProducer.SetHTMLDoc Method

Sets the HTMLDoc (see page 1105) property.

Pascal

```
procedure SetHTMLDoc(const Value: TStrings);
```

C++

```
__fastcall SetHTMLDoc(const TStrings Value);
```

Description

SetHTMLDoc is the protected write implementation of the HTMLDoc (see page 1105) property.

Value is a TStrings descendant that contains the new HTML template. SetHTMLDoc assigns the strings in this object to the TStrings object that implements the HTMLDoc (see page 1105) property. Note that this method does not take ownership of the specified object.

See Also

HTMLDoc (see page 1105)

Assign

11.1.3.4.4 TCustomAdapterPageProducer.SetHTMLFile Method

Sets the HTMLFile (see page 1106) property.

Pascal

```
procedure SetHTMLFile(const Value: TFileName);
```

C++

```
__fastcall SetHTMLFile(const TFileName Value);
```

Description

SetHTMLFile is the protected write implementation of the HTMLFile (see page 1106) property.

Value is the name of the template that the page producer should use to generate its content.

See Also

HTMLFile (see page 1106)

11.1.3.4.5 TCustomAdapterPageProducer.WebContent Method

Returns the HTML document assembled by the Adapter page producer.

Pascal

```
function WebContent(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString WebContent(TWebContentOptions Options, TLayout Layout);
```

Description

Call WebContent to get the HTML document produced by TCustomAdapterPageProducer (see page 1099). TCustomAdapterPageProducer (see page 1099) uses WebContent to implement the Content method of its IWebContent (see page 6075) interface.

Options contains a set of flags that limit what can be included in the generated HTML.

Layout is ignored by the WebContent method. It is included as a parameter for compatibility with other content producers that use a helper object to coordinate the layout of content produced by multiple Web items.

The Adapter page producer creates its content by replacing HTML-transparent tags in the template specified by HTMLDoc (see page 1105) or HTMLFile (see page 1106).

By default, this template has the form

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Although the template can be changed to include any combination of these tags with custom HTML or tags that the application translates in an OnHTMLTag event handler.

The WebContent method translates the tags listed above as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1097) or StylesFile (see page 1098) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items specified by the WebPageItems property.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer (see page 1099) will not generate any content if this tag is missing.

The string returned by WebContent can be used as the Content property of the response to an HTTP request message.

See Also

IWebContent (see page 6075)

Styles (see page 1097)

StylesFile (see page 1098)

OnHTMLTag

HTMLDoc (see page 1105)

HTMLFile (see page 1106)

WebPageItems

11.1.3.5 TCustomAdapterPageProducer Properties

The properties of the TCustomAdapterPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 1105)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1106)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

11.1.3.5.1 TCustomAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The

pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer (see page 1099) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1097) or StylesFile (see page 1098) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer (see page 1099) will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile (see page 1106) property.

See Also

Content

HTMLFile (see page 1106)

OnHTMLTag

11.1.3.5.2 TCustomAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 1105) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content

HTMLDoc (see page 1105)

OnHTMLTag

11.1.4 TSitePageElements Class

TSitePageElements stores part of the content that page producers generate.

Class Hierarchy

Pascal

```
TSitePageElements = class(TObject);
```

C++

```
class TSitePageElements : public TObject;
```

File

SiteProd

Description

TSitePageElements stores styles, warnings, server script, server functions, and server variables. These values are stored as class members of type string (not properties). The adapter page producer combines them to make up its generated content.

11.1.4.1 TSitePageElements Members

The following tables list the members exposed by TSitePageElements.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 1108)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 1109)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 1109)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 1109)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 1109)	This is WarningsContent, a member of class TSitePageElements.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

11.1.4.2 TSitePageElements Fields

The fields of the TSitePageElements class are listed here.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 1108)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 1109)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 1109)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 1109)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 1109)	This is WarningsContent, a member of class TSitePageElements.

11.1.4.2.1 TSitePageElements.ServerFunctionsContent Field

Pascal

```
ServerFunctionsContent: string;
```

C++

```
UnicodeString ServerFunctionsContent;
```

Description

This is ServerFunctionsContent, a member of class TSitePageElements.

11.1.4.2.2 TSitePageElements.ServerScriptContent Field

Pascal

```
ServerScriptContent: string;
```

C++

```
UnicodeString ServerScriptContent;
```

Description

This is ServerScriptContent, a member of class TSitePageElements.

11.1.4.2.3 TSitePageElements.ServerVariablesContent Field

Pascal

```
ServerVariablesContent: string;
```

C++

```
UnicodeString ServerVariablesContent;
```

Description

This is ServerVariablesContent, a member of class TSitePageElements.

11.1.4.2.4 TSitePageElements.StylesContent Field

Pascal

```
StylesContent: string;
```

C++

```
UnicodeString StylesContent;
```

Description

This is StylesContent, a member of class TSitePageElements.

11.1.4.2.5 TSitePageElements.WarningsContent Field

Pascal

```
WarningsContent: string;
```

C++

```
UnicodeString WarningsContent;
```

Description

This is WarningsContent, a member of class TSitePageElements.

12 SOAPAttachIntf Namespace

This is namespace SOAPAttachIntf.

Topics

Name	Description
Classes (see page 1112)	The following table lists classes in this documentation.
Interfaces (see page 1116)	The following table lists interfaces in this documentation.
Types (see page 1125)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 1112)	TSoapDataList maintains a list of SOAP attachments.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 1116)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Types

Name	Description
TOnGetAttachmentEvent (see page 1125)	This is type SOAPAttachIntf.TOnGetAttachmentEvent.
TOnSendAttachmentEvent (see page 1125)	This is type SOAPAttachIntf.TOnSendAttachmentEvent.

12.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 1112)	TSoapDataList maintains a list of SOAP attachments.

12.1.1 TSoapDataList Class

TSoapDataList maintains a list of SOAP attachments.

Class Hierarchy

Pascal

```
TSoapDataList = class(TList);
```

C++

```
class TSoapDataList : public TList;
```

File

SOAPAttachIntf

Description

Use TSoapDataList to store and maintain a list of SOAP attachments (TSOAPAttachment objects).

See Also

TList

TSoapAttachment

12.1.1.1 TSoapDataList Members

The following tables list the members exposed by TSoapDataList.

Public Methods

	Name	Description
	Add (see page 1113)	Adds a new attachment to the end of the list.
	Assign (see page 1113)	Raises an exception.
	Clear (see page 1114)	Empties the list and frees the TSOAPAttachment objects.
	Insert (see page 1114)	Raises an exception.
	Remove (see page 1114)	Raises an exception.

12.1.1.2 TSoapDataList Methods

The methods of the TSoapDataList class are listed here.

Public Methods

	Name	Description
	Add (see page 1113)	Adds a new attachment to the end of the list.
	Assign (see page 1113)	Raises an exception.
V	Clear (see page 1114)	Empties the list and frees the TSOAPAttachment objects.
	Insert (see page 1114)	Raises an exception.
	Remove (see page 1114)	Raises an exception.

12.1.1.2.1 TSoapDataList.Add Method

Adds a new attachment to the end of the list.

Pascal

```
function Add(Item: Pointer): Integer; reintroduce;
```

C++

```
__fastcall int Add(void * Item);
```

Description

Do not call the Add method. This method is used internally to add attachments to the data list when the application first reads an incoming message that includes attachments or when marshaling a message to be sent that includes attachments. Applications should not change the list of attachments, or they will interfere with the marshaling and unmarshaling of attachment data.

Item is a pointer to the TSoapAttachment instance that is added to the list.

Add returns the index of the new attachment.

See Also

Capacity

Count

Insert (see page 1114)

Items

Clear (see page 1114)

12.1.1.2.2 TSoapDataList.Assign Method

Raises an exception.

Pascal

```
procedure Assign(ListA: TList; AOperator: TListAssignOp = laCopy; ListB: TList = Nil);
reintroduce;
```

C++

```
__fastcall Assign(TList ListA, TListAssignOp AOperator = laCopy, TList ListB = Nil);
```

Description

Do not call the Assign method. TSoapDataList (see page 1112) replaces the inherited method to raise an exception if you call

this method because applications should not change the items in a TSoapDataList (see page 1112) object.

See Also

Add (see page 1113)

12.1.1.2.3 TSoapDataList.Clear Method

Empties the list and frees the TSOAPAttachment objects.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Clear removes all of the objects from the list and frees each one. Do not call this method. It is used internally when marshaling and unmarshaling messages that contain attachments. Applications should not change the items in a TSoapDataList (see page 1112) object.

See Also

Delete

Items

Remove

12.1.1.2.4 TSoapDataList.Insert Method

Raises an exception.

Pascal

```
procedure Insert(Index: Integer; aClass: TClass); reintroduce;
```

C++

```
__fastcall Insert(int Index, TClass aClass);
```

Description

Do not call the Insert method. TSoapDataList (see page 1112) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 1112) object.

See Also

Add (see page 1113)

12.1.1.2.5 TSoapDataList.Remove Method

Raises an exception.

Pascal

```
function Remove(aClass: TClass): Integer; reintroduce;
```

C++

```
__fastcall int Remove(TClass aClass);
```

Description

Do not call the Remove method. TSoapDataList (see page 1112) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 1112) object.

See Also

Clear (see page 1114)

12.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 1116)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

12.2.1 IMimeAttachmentHandler Interface

IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Class Hierarchy

SOAPAttachIntf.IMimeAttachmentHandler

Pascal

```
IMimeAttachmentHandler = interface;
```

C++

```
__interface IMimeAttachmentHandler;
```

File

SOAPAttachIntf

Description

IMimeAttachmentHandler defines methods for working with multipart forms that represent SOAP requests which include attachements. It is used internally to process the headers of the SOAP envelope part of multipart forms, to generate events when the application sends or receives a multipart form that represents a Web Service call which includes attachments, and to parse or assemble such a multipart form.

Note: C++ method declarations that use IMimeAttachmentHandler use the _di_IMimeAttachmentHandler type instead. This type is a DelphiInterface (see page 3263) wrapper around the IMimeAttachmentHandler interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IMimeAttachmentHandler >
_di_IMimeAttachmentHandler;
```

See Also

TSoapAttachment

12.2.1.1 IMimeAttachmentHandler Members

The following tables list the members exposed by IMimeAttachmentHandler.

Public Events

	Name	Description
	OnGetAttachment (see page 1123)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 1123)	Occurs for each attachment that is added to a multipart form to be sent.

Public Methods

	Name	Description
	AddSoapHeader (see page 1117)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 1118)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 1118)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 1119)	Returns the value of the MIMEBoundary (see page 1122) property.
	GetMIMEStream (see page 1119)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 1120)	Returns the OnGetAttachment (see page 1123) event handler.
	GetOnSendAttachmentEvent (see page 1120)	Returns the OnSendAttachment (see page 1123) event handler.
	ProcessMultiPartForm (see page 1120)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 1121)	Sets the value of the MIMEBoundary (see page 1122) property.
	SetOnGetAttachmentEvent (see page 1121)	Assigns the OnGetAttachment (see page 1123) event handler.
	SetOnSendAttachmentEvent (see page 1122)	Assigns the OnSendAttachment (see page 1123) event handler.

Public Properties

	Name	Description
	MIMEBoundary (see page 1122)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

12.2.1.2 IMimeAttachmentHandler Methods

The methods of the IMimeAttachmentHandler class are listed here.

Public Methods

	Name	Description
	AddSoapHeader (see page 1117)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 1118)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 1118)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 1119)	Returns the value of the MIMEBoundary (see page 1122) property.
	GetMIMEStream (see page 1119)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 1120)	Returns the OnGetAttachment (see page 1123) event handler.
	GetOnSendAttachmentEvent (see page 1120)	Returns the OnSendAttachment (see page 1123) event handler.
	ProcessMultiPartForm (see page 1120)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 1121)	Sets the value of the MIMEBoundary (see page 1122) property.
	SetOnGetAttachmentEvent (see page 1121)	Assigns the OnGetAttachment (see page 1123) event handler.
	SetOnSendAttachmentEvent (see page 1122)	Assigns the OnSendAttachment (see page 1123) event handler.

12.2.1.2.1 IMimeAttachmentHandler.AddSoapHeader Method

Adds a specified header to the SOAP envelope portion of the emerging .

Pascal

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

C++

```
__fastcall AddSoapHeader(UnicodeString Value);
```

Description

Call AddSoapHeader to add an arbitrary header to the SOAP envelope portion of an emerging multipart form. This method should not be called until after a call to the CreateMimeStream (see page 1118) method initializes the emerging multipart form.

Value is the text of the header to add to the SOAP envelope.

Typically, this method is called to add the ContentType header to the SOAP envelope when its value is not known at the point when the application calls CreateMimeStream (see page 1118).

Note: After all calls to AddSoapHeader have been made to add headers to the SOAP envelope, the application must call the FinalizeStream (see page 1118) method to allow the MIME attachment handler to finish generating the multipart form. Only after the call to FinalizeStream (see page 1118) can the application read the content of the multipart form by calling the GetMIMEStream (see page 1119) method.

See Also

CreateMimeStream (see page 1118)

FinalizeStream (see page 1118)

GetMIMEStream (see page 1119)

12.2.1.2.2 IMimeAttachmentHandler.CreateMimeStream Method

Creates and initializes a stream that will eventually contain an assembled multipart form.

Pascal

```
procedure CreateMimeStream(Envelope: TStream; Attachments: TSoapDataList);
```

C++

```
__fastcall CreateMimeStream(TStream Envelope, TSoapDataList Attachments);
```

Description

Call CreateMimeStream as the first step when generating a multipart form to represent a SOAP packet and its associated attachments.

Envelope is a stream from which CreateMimeStream can read the SOAP envelope portion of the multipart form. This must be a complete encoding with the exception of headers, which can be added by subsequent calls to the AddSoapHeader (see page 1117) method.

Attachments is a list of TSoapAttachment objects that represent the attachments that should be included in the multipart form.

CreateMimeStream creates a new stream object and initializes it with the data from Envelope and Attachments. This new stream can then be completed by calling the AddSoapHeader (see page 1117) method to add any headers to the SOAP envelope portion and then calling the FinalizeStream (see page 1118) method. After calling FinalizeStream (see page 1118), the application can access the generated stream by calling the GetMIMEStream (see page 1119) method.

See Also

FinalizeStream (see page 1118)

AddSoapHeader (see page 1117)

GetMIMEStream (see page 1119)

12.2.1.2.3 IMimeAttachmentHandler.FinalizeStream Method

Performs any necessary final steps in creating the emerging multipart form.

Pascal

```
procedure FinalizeStream;
```

C++

```
__fastcall FinalizeStream();
```

Description

Call FinalizeStream as a final step when assembling a multipart form to represent a SOAP packet and its associated attachments. When an application assembles a multipart form to represent a SOAP packet and its associated attachments, it takes the following steps:

1. Call CreateMimeStream (see page 1118) to create the stream object that includes data from the SOAP envelope and the content of any attachments.
2. Call AddSoapHeader (see page 1117) to add any headers to the SOAP envelope portion of the stream.
3. Call FinalizeStream to indicate that all changes are complete and allow the MIME attachment handler to make any final changes.

Only after a call to FinalizeStream can the application read the assembled multipart form by calling GetMIMEStream (see page 1119).

See Also

CreateMimeStream (see page 1118)

AddSoapHeader (see page 1117)

GetMIMEStream (see page 1119)

12.2.1.2.4 IMimeAttachmentHandler.GetMIMEBoundary Method

Returns the value of the MIMEBoundary (see page 1122) property.

Pascal

```
function GetMIMEBoundary: String;
```

C++

```
__fastcall UnicodeString GetMIMEBoundary();
```

Description

GetMIMEBoundary is the read implementation for the MIMEBoundary (see page 1122) property.

See Also

MIMEBoundary (see page 1122)

12.2.1.2.5 IMimeAttachmentHandler.GetMIMEStream Method

Returns a stream from which to read the assembled multipart form.

Pascal

```
function GetMIMEStream(Release: Boolean = False): TStream;
```

C++

```
__fastcall TStream GetMIMEStream(Boolean Release = False);
```

Description

Call GetMIMEStream after a call to the FinalizeStream (see page 1118) method to access the stream from which to read a newly-assembled multipart form. This stream should not be used until after the multipart form is fully assembled.

Release (see page 3732) indicates whether the caller is responsible for freeing the stream object. When Release (see page 3732) is true, the caller takes ownership of the stream and is responsible for freeing it. When Release (see page 3732) is false, the caller should not free the stream that is returned, because the object that implements IMimeAttachmentHandler (see page 1116) will free it.

See Also

CreateMimeStream (see page 1118)

AddSoapHeader (see page 1117)

FinalizeStream (see page 1118)

12.2.1.2.6 IMimeAttachmentHandler.GetOnGetAttachmentEvent Method

Returns the OnGetAttachment (see page 1123) event handler.

Pascal

```
function GetOnGetAttachmentEvent: TOnGetAttachmentEvent;
```

C++

```
__fastcall TOnGetAttachmentEvent GetOnGetAttachmentEvent();
```

Description

GetOnGetAttachmentEvent is the read implementation for the OnGetAttachment (see page 1123) event.

See Also

OnGetAttachment (see page 1123)

12.2.1.2.7 IMimeAttachmentHandler.GetOnSendAttachmentEvent Method

Returns the OnSendAttachment (see page 1123) event handler.

Pascal

```
function GetOnSendAttachmentEvent: TOnSendAttachmentEvent;
```

C++

```
__fastcall TOnSendAttachmentEvent GetOnSendAttachmentEvent();
```

Description

GetOnSendAttachmentEvent is the read implementation for the OnSendAttachment (see page 1123) event handler.

See Also

OnSendAttachment (see page 1123)

12.2.1.2.8 IMimeAttachmentHandler.ProcessMultiPartForm Method

Parses a multipart form into the SOAP envelope and a list of attachments.

Pascal

```
procedure ProcessMultiPartForm(const ASourceStream: TStream; const ADestStream: TStream;
```

```
AMimeBoundary: string; SoapHeaders: TStrings; Attachments: TSoapDataList; TempDir: string);
```

C++

```
__fastcall ProcessMultiPartForm(const TStream ASourceStream, const TStream ADestStream,
UnicodeString AMimeBoundary, TStrings SoapHeaders, TSoapDataList Attachments, UnicodeString
TempDir);
```

Description

Call ProcessMultiPartForm to interpret a multipart form that is received by the client or server in a Web Service application. This method separates the SOAP envelope portion of the multipart form from the portions that represent attachments, and parses the headers of the SOAP envelope into a convenient string list.

ASourceStream is a stream from which ProcessMultiPartForm can read the incoming multipart form. This is the content of the incoming message.

ADestStream is a stream to which ProcessMultiPartForm writes the SOAP envelope portion of ASourceStream.

AMimeBoundary is the mime boundary string that identifies the boundaries between the parts of the multipart form. Typically, it is obtained from the headers of the incoming HTTP message whose content is the content of ASourceStream.

SoapHeaders is an existing TStrings descendant to which ProcessMultiPartForm writes the headers of the SOAP envelope portion of the multipart form. Each header is added as a string of the form HeaderName=Value.

Attachments is an empty TSoapDataList (see page 1112) object to which ProcessMultiPartForm adds the TSoapAttachment objects that represent the attachments in the multipart form.

TempDir is the name of the directory where ProcessMultiPartForm writes the temporary files it generates for each of the attachments.

12.2.1.2.9 IMimeAttachmentHandler.SetMIMEBoundary Method

Sets the value of the MIMEBoundary (see page 1122) property.

Pascal

```
procedure SetMIMEBoundary(const MimeBndry: String);
```

C++

```
__fastcall SetMIMEBoundary(const UnicodeString MimeBndry);
```

Description

SetMIMEBoundary is the write implementation for the MIMEBoundary (see page 1122) property.

MimeBndry is the string that should identify the boundary between parts of the multipart form.

See Also

MIMEBoundary (see page 1122)

12.2.1.2.10 IMimeAttachmentHandler.SetOnGetAttachmentEvent Method

Assigns the OnGetAttachment (see page 1123) event handler.

Pascal

```
procedure SetOnGetAttachmentEvent(OnGetAttachment: TOnGetAttachmentEvent);
```

C++

```
__fastcall SetOnGetAttachmentEvent(TOnGetAttachmentEvent OnGetAttachment);
```

Description

SetOnGetAttachmentEvent is the write implementation for the OnGetAttachment (see page 1123) event.

OnGetAttachment (see page 1123) is the event handler that should to respond to the event.

See Also

OnGetAttachment (see page 1123)

12.2.1.2.11 IMimeAttachmentHandler.SetOnSendAttachmentEvent Method

Assigns the OnSendAttachment (see page 1123) event handler.

Pascal

```
procedure SetOnSendAttachmentEvent(OnSendAttachment: TOnSendAttachmentEvent);
```

C++

```
__fastcall SetOnSendAttachmentEvent(TOnSendAttachmentEvent OnSendAttachment);
```

Description

SetOnSendAttachmentEvent is the write implementation for the OnSendAttachment (see page 1123) event.

OnSendAttachment (see page 1123) is the event handler that should to respond to the event.

See Also

OnSendAttachment (see page 1123)

12.2.1.3 IMimeAttachmentHandler Properties

The properties of the IMimeAttachmentHandler class are listed here.

Public Properties

	Name	Description
	MIMEBoundary (see page 1122)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

12.2.1.3.1 IMimeAttachmentHandler.MIMEBoundary Property

Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

Pascal

```
property MIMEBoundary: string;
```

C++

```
__property UnicodeString MIMEBoundary;
```

Description

MIMEBoundary is the string that identifies the boundary between the parts of the multipart form. It is an attribute of the ContentType header for the entire form.

When the MIME attachment handler is processing a received form, this property is not available until after the application has called the ProcessMultiPartForm (see page 1120) method. When the MIME attachment handler is assembling a multipart form in order to send a message that includes attachments, the handler generates this string for the new form, and the CreateMimeStream (see page 1118) method uses it when assembling the form.

See Also

ProcessMultiPartForm (see page 1120)

CreateMimeStream (see page 1118)

12.2.1.4 IMimeAttachmentHandler Events

The events of the IMimeAttachmentHandler class are listed here.

Public Events

	Name	Description
	OnGetAttachment (see page 1123)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 1123)	Occurs for each attachment that is added to a multipart form to be sent.

12.2.1.4.1 IMimeAttachmentHandler.OnGetAttachment Event

Occurs for each attachment as it is read.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

The MIME attachment handler generates this event as it reads each incoming attachment. Classes that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnSendAttachment (see page 1123)

OnGetAttachment

OnGetAttachment (see page 1292)

12.2.1.4.2 IMimeAttachmentHandler.OnSendAttachment Event

Occurs for each attachment that is added to a multipart form to be sent.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

The MIME attachment handler generates this event as it adds each outgoing attachment to the multipart form. Classes that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnGetAttachment (see page 1123)

OnSendAttachment

OnSendAttachment (see page 1292)

12.3 Types

The following table lists types in this documentation.

Types

Name	Description
TOnGetAttachmentEvent (see page 1125)	This is type SOAPAttachIntf.TOnGetAttachmentEvent.
TOnSendAttachmentEvent (see page 1125)	This is type SOAPAttachIntf.TOnSendAttachmentEvent.

12.3.1 SOAPAttachIntf.TOnGetAttachmentEvent Type

Pascal

```
TOnGetAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnGetAttachmentEvent)();
```

File

SOAPAttachIntf

Description

This is type SOAPAttachIntf.TOnGetAttachmentEvent.

12.3.2 SOAPAttachIntf.TOnSendAttachmentEvent Type

Pascal

```
TOnSendAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnSendAttachmentEvent)();
```

File

SOAPAttachIntf

Description

This is type SOAPAttachIntf.TOnSendAttachmentEvent.

13 SOAPConn Namespace

This is namespace SOAPConn.

Topics

Name	Description
Classes (see page 1128)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapConnection (see page 1128)	TSoapConnection connects to an application server that is implemented as a Web Service.

13.1 Classes

The following table lists classes in this documentation.

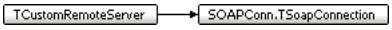
Classes

	Name	Description
	TSoapConnection (see page 1128)	TSoapConnection connects to an application server that is implemented as a Web Service.

13.1.1 TSoapConnection Class

TSoapConnection connects to an application server that is implemented as a Web Service.

Class Hierarchy



Pascal

```
TSoapConnection = class(TCustomRemoteServer);
```

C++

```
class TSoapConnection : public TCustomRemoteServer;
```

File

SOAPConn

Description

TSoapConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server that is implemented as a Web Service. The SOAP connection object can

Establish an initial connection to a remote application server.

Obtain an interface for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSoapConnection establishes the initial connection between the client application and a remote application server using SOAP.

Note: To use TSoapConnection on Windows, you must have wininet.dll installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

TSoapConnection uses an internal THTTPRio object to obtain an interface from the Web Service application. This interface is either IAppServer or IAppServerSOAP (see page 1276) (or one of their descendants), depending on the value of the UseSOAPAdapter (see page 1140) property. The recommended approach is to use the IAppServerSOAP (see page 1276) interface, which is better adapted to SOAP as a transport protocol. However, some application servers (those created using Delphi 6 or Kylix 2) only support an IAppServer interface. If you are using an application server that does not support IAppServerSOAP (see page 1276), set UseSOAPAdapter (see page 1140) to false.

Regardless of which interface TSoapConnection uses to communicate with the application server, it makes the information available to client datasets in the client application by providing an IAppServer interface. Client datasets use the IAppServer

interface from the SOAP connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

Note: Unlike when using other DataSnap connection components, you can't use the AppServer property of TSoapConnection to call methods of the application server's interface that are not IAppServer methods. Instead, to communicate with a SOAP data module on the application interface, use a separate THTTPRIO (see page 1186) object or specify the interface using the SOAPServerIID (see page 1138) property and then call the GetSOAPServer (see page 1134) method

. To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection component instead. To connect to remote application servers using HTTP (but not SOAP), use TWebConnection.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TSocketConnection (see page 176)

TWebConnection (see page 209)

THTTPRio

IAppServerSoap

Extending the Interface of the Application Server

Specifying a Connection Using SOAP

13.1.1.1 TSoapConnection Members

The following tables list the members exposed by TSoapConnection.

Protected Methods

	Name	Description
	DoConnect (see page 1131)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 1132)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 1132)	Returns the value of the Connected property.
	GetSOAPServerIID (see page 1134)	This is GetSOAPServerIID, a member of class TSoapConnection.
	Notification (see page 1134)	Forwards notification messages to all owned components.
	SetSOAPServerIID (see page 1135)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 1135)	This is SetUseSoapAdapter, a member of class TSoapConnection.

Public Methods

	Name	Description
	Create (see page 1130)	Creates and initializes an instance of TSoapConnection (see page 1128).
	Destroy (see page 1131)	Destroys an instance of TSoapConnection (see page 1128).
	GetProviderNames (see page 1132)	Executes a callback for all provider objects on the application server.
	GetServer (see page 1133)	Returns an IAppServer interface for communicating with the application server.
	GetServerList (see page 1133)	Returns a list of application servers (remote data modules) to which the component can connect.

	GetSOAPServer (see page 1134)	Returns the IAppServerSOAP (see page 1276) interface of the application server.

Public Properties

	Name	Description
	RIO (see page 1138)	Provides access to the internal remote interfaced object that communicates with the application server.

Published Events

	Name	Description
	AfterConnect (see page 1141)	Occurs after a connection is established.
	AfterDisconnect (see page 1141)	Occurs after the connection closes.
	BeforeConnect (see page 1142)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1142)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 1143)	Occurs when TSoapConnection (see page 1128) receives the results from executing a method on the application server.
	OnBeforeExecute (see page 1143)	Occurs before TSoapConnection (see page 1128) sends a method call to the application server for execution.

Published Properties

	Name	Description
	Agent (see page 1136)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 1136)	Indicates whether the SOAP connection component is connected to the application server.
	Password (see page 1137)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 1137)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1137)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 1138)	Identifies the invokable interface that TSoapConnection (see page 1128) uses to communicate with the application server.
	URL (see page 1139)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 1140)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 1140)	Specifies how the SOAP connection component communicates with the application server.

13.1.1.2 TSoapConnection.Create Constructor

Creates and initializes an instance of TSoapConnection (see page 1128).

Pascal

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

C++

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

Description

Call Create to instantiate a TSoapConnection (see page 1128) object at runtime. SOAP connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the SOAP connection component.

See Also

Owner

13.1.1.3 TSoapConnection.Destroy Destructor

Destroys an instance of TSoapConnection (see page 1128).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the SOAP connection is not nil before calling Destroy.

13.1.1.4 TSoapConnection Methods

The methods of the TSoapConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 1131)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 1132)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 1132)	Returns the value of the Connected property.
	GetSOAPServerIID (see page 1134)	This is GetSOAPServerIID, a member of class TSoapConnection.
	Notification (see page 1134)	Forwards notification messages to all owned components.
	SetSOAPServerIID (see page 1135)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 1135)	This is SetUseSoapAdapter, a member of class TSoapConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 1132)	Executes a callback for all provider objects on the application server.
	GetServer (see page 1133)	Returns an IAppServer interface for communicating with the application server.
	GetServerList (see page 1133)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetSOAPServer (see page 1134)	Returns the IAppServerSOAP (see page 1276) interface of the application server.

13.1.1.4.1 TSoapConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

13.1.1.4.2 TSoapConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

13.1.1.4.3 TSoapConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

13.1.1.4.4 TSoapConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames

AppServer

ProviderName

13.1.1.4.5 TSoapConnection.GetServer Method

Returns an IAppServer interface for communicating with the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Client datasets call GetServer to set the value of their AppServer property. This allows them to access providers on the remote application server.

GetServer ensures that there is a connection and then returns an IAppServer interface that can be used to communicate with providers on the application server. To use other methods of the application server's interface, use a separate THTTPRIO (see page 1186) object or call the GetSOAPServer (see page 1134) method after setting the SOAPServerIID (see page 1138) property.

See Also

Connected (see page 1136)

AppServer

THTTPRIO (see page 1186)

UseSOAPAdapter (see page 1140)

SOAPServerIID (see page 1138)

13.1.1.4.6 TSoapConnection.GetServerList Method

Returns a list of application servers (remote data modules) to which the component can connect.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName property.

As implemented in TCustomRemoteServer, GetServerList always returns NULL. Descendant classes override this method to return a list of application servers that are on the local machine and registered in the system registry (TDispatchConnection), or fetched from the server machine (TStreamedConnection).

See Also

ServerName

13.1.1.4.7 TSoapConnection.GetSOAPServer Method

Returns the IAppServerSOAP (see page 1276) interface of the application server.

Pascal

```
function GetSOAPServer: IAppServerSOAP;
```

C++

```
__fastcall IAppServerSOAP GetSOAPServer();
```

Description

Call GetSOAPServer to access the IAppServerSOAP (see page 1276) interface that the SOAP connection component uses when UseSOAPAdapter (see page 1140) is true. The internal adapter uses this interface to implement the IAppServer interface that the GetServer (see page 1133) method returns.

Tip: If you have set the SOAPServerIID (see page 1138) property, you can use the as operator (Delphi) or the Supports (see page 4335) method (C++) to convert this interface into the application server's interface.

See Also

Connected (see page 1136)

GetServer (see page 1133)

UseSOAPAdapter (see page 1140)

THTTPRio

SOAPServerIID (see page 1138)

13.1.1.4.8 TSoapConnection.GetSOAPServerIID Method

Pascal

```
function GetSOAPServerIID: TGUID;
```

C++

```
__fastcall TGUID GetSOAPServerIID();
```

Description

This is GetSOAPServerIID, a member of class TSoapConnection.

13.1.1.4.9 TSoapConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

13.1.1.4.10 TSoapConnection.SetSOAPServerIID Method

Pascal

```
procedure SetSOAPServerIID(const IID: String);
```

C++

```
__fastcall SetSOAPServerIID(const UnicodeString IID);
```

Description

This is SetSOAPServerIID, a member of class TSoapConnection.

13.1.1.4.11 TSoapConnection.SetUseSoapAdapter Method

Pascal

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

C++

```
__fastcall SetUseSoapAdapter(Boolean Value);
```

Description

This is SetUseSoapAdapter, a member of class TSoapConnection.

13.1.1.5 TSoapConnection Properties

The properties of the TSoapConnection class are listed here.

Public Properties

	Name	Description
	RIO (see page 1138)	Provides access to the internal remote interfaced object that communicates with the application server.

Published Properties

	Name	Description
	Agent (see page 1136)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 1136)	Indicates whether the SOAP connection component is connected to the application server.
	Password (see page 1137)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 1137)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1137)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 1138)	Identifies the invokable interface that TSoapConnection (see page 1128) uses to communicate with the application server.
	URL (see page 1139)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 1140)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 1140)	Specifies how the SOAP connection component communicates with the application server.

13.1.1.5.1 TSoapConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply HTTP messages generated by the SOAP connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title

13.1.1.5.2 TSoapConnection.Connected Property

Indicates whether the SOAP connection component is connected to the application server.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to true to establish a connection to the application server. Set Connected to false to terminate the connection.

When changing the value of Connected, TSoapConnection (see page 1128) generates a BeforeConnect (see page 1142) and an AfterConnect (see page 1141) event, or a BeforeDisconnect (see page 1142) and an AfterDisconnect (see page 1141) event.

See Also

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

13.1.1.5.3 TSoapConnection.Password Property

Specifies the password that allows the client application to log on to a local host or proxy server.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When the SOAP connection component connects to the application server, it supplies the values of the UserName (see page 1140) and Password properties so that it can log on to the local host or proxy. The value of Password can be an empty string if neither the local host nor proxy requires authentication.

Note: On Linux, if the proxy does not require authentication, you can use the UserName (see page 1140) and Password properties to log on to the Web Server.

See Also

UserName (see page 1140)

Specifying a Connection Using SOAP

13.1.1.5.4 TSoapConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the SOAP connection component can't resolve the host portion of URL (see page 1139) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 1137)

Specifying a Connection Using SOAP

13.1.1.5.5 TSoapConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the SOAP connection component can make a direct connection. When the URL (see page 1139) property identifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 1137) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and bypasses the proxy server if the host portion of URL (see page 1139) refers to a registered local server.

See Also

Proxy (see page 1137)

URL (see page 1139)

13.1.1.5.6 TSoapConnection.RIO Property

Provides access to the internal remote interfaced object that communicates with the application server.

Pascal

```
property RIO: THTTPRIO;
```

C++

```
__property THTTPRIO RIO;
```

Description

TSoapConnection (see page 1128) uses an internal remote interfaced object (THTTPRIO (see page 1186)) to communicate with the application server. This internal object obtains a reference to an invokable interface that is specified by the SOAPServerIID (see page 1138) property.

Use RIO to access the internal THTTPRIO (see page 1186) instance.

See Also

SOAPServerIID (see page 1138)

13.1.1.5.7 TSoapConnection.SOAPServerIID Property

Identifies the invokable interface that TSoapConnection (see page 1128) uses to communicate with the application server.

Pascal

```
property SOAPServerIID: String;
```

C++

```
__property UnicodeString SOAPServerIID;
```

Description

Set SOAPServerIID to identify the interface of the application server. SOAPServerIID is the GUID of the application server's interface. This interface is a descendant of IAppServerSOAP (see page 1276). Once you have formed a connection, you can access this interface by first calling the GetSOAPServer (see page 1134) method and then using the global Supports (see page 4335) function or as operator (Delphi) or the Supports (see page 4335) method of the interface (C++).

Warning: Use the GUID for the interface that is generated by the WSDL importer when you import the application server's definitions. This may differ from the GUID that the server uses in its implementation.

Note: In Delphi, at runtime, you can use the interface name in code and the compiler automatically converts that to the GUID. At design time (in the Object Inspector) or in C++ code, you must specify the GUID itself, not the interface name.

Note: SOAPServerIID is only used if the UseSOAPAdapter (see page 1140) property is true. In addition, if you set SOAPServerIID to anything other than IAppServerSOAP (see page 1276), you must import the definition of the application server's interface so that it is registered with the invocation registry.

See Also

URL (see page 1139)

IAppServerSOAP (see page 1276)

UseSOAPAdapter (see page 1140)

GetSOAPServer (see page 1134)

Specifying a Connection Using SOAP

13.1.1.5.8 TSoapConnection.URL Property

Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the SOAP connection component uses to locate the application server.

URL includes the protocol (http or https), host name, and scriptname for the application server, as well as a path portion that identifies the THTTPSoapDispatcher on the application server. Typically, it has the form

```
http://DataHost.org/scripts/AppServer.dll/SOAP/
http://DataHost.org/scripts/AppServer.so/SOAP/
```

Note: By default, URL does not include the interface name. The name of the interface is assumed to be IAppServerSOAP (see page 1276) (or IAppServer). If the server is written in the Delphi language and the target remote data module supports an IAppServerSOAP (see page 1276) descendant, Delphi clients can optionally include the target interface name at the end of the URL. This allows clients to identify a SOAP data module when the server contains multiple SOAP data modules, or allows you to access the server's interface by calling the GetSOAPServer (see page 1134) method. If the client or server is not written using Delphi, you can accomplish this same effect using the SOAPServerIID (see page 1138) property.

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

THTTPSoapDispatcher (see page 5920)

SOAPServerIID (see page 1138)

GetSOAPServer (see page 1134)

Specifying a Connection Using SOAP

13.1.1.5.9 TSoapConnection.UserName Property

Specifies the user name that allows the client application to log on to a local host or proxy server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When the SOAP connection component connects to the application server on a local host or to a proxy server, it supplies the values of the UserName and Password (see page 1137) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

Note: On Linux, if the proxy does not require authentication, you can use the UserName and Password (see page 1137) properties to log on to the Web Server.

See Also

Password (see page 1137)

Proxy (see page 1137)

Specifying a Connection Using SOAP

13.1.1.5.10 TSoapConnection.UseSOAPAdapter Property

Specifies how the SOAP connection component communicates with the application server.

Pascal

```
property UseSOAPAdapter: Boolean;
```

C++

```
__property Boolean UseSOAPAdapter;
```

Description

When UseSOAPAdapter is true, TSOAPConnection communicates with the application server using an IAppServerSOAP (see page 1276) interface. It then uses an internal adapter which implements the IAppServer interface for client datasets. In this case, the GetServer (see page 1133) method returns the IAppServer interface that the internal adapter implements.

When UseSOAPAdpater is false, TSOAPConnection uses an IAppServer interface to communicate with the application server and exposes that interface with the GetServer (see page 1133) method. In this case, the GetSOAPServer (see page 1134) method returns nil (Delphi) or NULL (C++).

The recommended setting for UseSOAPAdapter is true. Unlike IAppServer, the IAppServerSOAP (see page 1276) interface does not use the safecall calling convention. This makes it easier to marshal interface calls when the client or server is written in C++. IAppServerSOAP (see page 1276) is not supported, however, when the application server is built using Delphi 6 (before update patch 2) or Kylix 2. If you are connecting to a server that does not support IAppServerSOAP (see page 1276), set UseSOAPAdapter to false.

See Also

IAppServer

IAppServerSOAP (see page 1276)

GetServer (see page 1133)

GetSOAPServer (see page 1134)

Specifying a Connection Using SOAP

13.1.1.6 TSoapConnection Events

The events of the TSoapConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 1141)	Occurs after a connection is established.
	AfterDisconnect (see page 1141)	Occurs after the connection closes.
	BeforeConnect (see page 1142)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1142)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 1143)	Occurs when TSoapConnection (see page 1128) receives the results from executing a method on the application server.
	OnBeforeExecute (see page 1143)	Occurs before TSoapConnection (see page 1128) sends a method call to the application server for execution.

13.1.1.6.1 TSoapConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

13.1.1.6.2 TSoapConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

13.1.1.6.3 TSoapConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

13.1.1.6.4 TSoapConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

13.1.1.6.5 TSoapConnection.OnAfterExecute Event

Occurs when TSoapConnection (see page 1128) receives the results from executing a method on the application server.

Pascal

```
property OnAfterExecute: TAfterExecuteEvent;
```

C++

```
__property TAfterExecuteEvent OnAfterExecute;
```

Description

Write (see page 3789) an OnAfterExecute event handler to examine or alter the encoded results from executing a method on the application server. OnAfterExecute is assigned as the OnAfterExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnBeforeExecute (see page 1143)

OnAfterExecute

GetServer (see page 1133)

13.1.1.6.6 TSoapConnection.OnBeforeExecute Event

Occurs before TSoapConnection (see page 1128) sends a method call to the application server for execution.

Pascal

```
property OnBeforeExecute: TBeforeExecuteEvent;
```

C++

```
__property TBeforeExecuteEvent OnBeforeExecute;
```

Description

Write (see page 3789) an OnBeforeExecute event handler to examine or alter an encoded method call before it is sent to the server. OnBeforeExecute occurs after the call has been marshaled into a SOAP request packet, but before that SOAP request packet has been sent to the application server.

OnBeforeExecute is assigned as the OnBeforeExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnAfterExecute (see page 1143)

OnBeforeExecute

GetServer (see page 1133)

14 SOAPDm Namespace

This is namespace SOAPDm.

Topics

Name	Description
Classes (see page 1146)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataModule (see page 1146)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

14.1 Classes

The following table lists classes in this documentation.

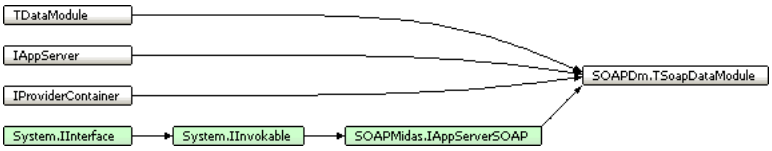
Classes

	Name	Description
	TSoapDataModule (see page 1146)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

14.1.1 TSoapDataModule Class

TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

Class Hierarchy



Pascal

```
TSoapDataModule = class(TDataModule, IAppServer, IAppServerSOAP, IProviderContainer);
```

C++

```
class TSoapDataModule : public TDataModule, public IAppServer, public IAppServerSOAP, public
IProviderContainer;
```

File

SOAPDm

Description

Use a TSoapDataModule object as the central repository for all objects in a SOAP-based application server. SOAP data modules can contain any nonvisual components. Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. They can also contain other nonvisual components, such as TTimer, or TImageList.

At design time a TSoapDataModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. In the unit file for the SOAP data module a developer can also code any business rules that are to be centralized in the middle tier of a multi-tiered application.

TSoapDataModule implements the IAppServer and IAppServerSOAP (see page 1276) interfaces. These are both invokable interfaces, so if you add TSoapDataModule to a Web Service application, the invoker automatically forwards all IAppServer or IAppServerSOAP (see page 1276) calls to the SOAP data module. The client application uses a TSoapConnection component to generate these calls.

To create a SOAP data module at design time,

1. Choose File|New (see page 3712)|Other, and select Web Services application from the WebServices page of the new items dialog. This generates a Web module that contains a dispatcher and an invoker component. These components work together to interpret and execute incoming SOAP requests.
2. Choose File|New (see page 3712)|Other, and select SOAP DataModule from the Web Services page of the new items dialog. This adds a TSoapDataModule descendant to your application. The TSoapDataModule descendant supports an invokable interface that descends from IAppServerSOAP (see page 1276) as well as an IAppServer interface.

   Note: Web Service applications should not include more than one TSoapDataModule descendant. When the invoker component that appears in the primary Web module receives a request for the IAppServer or IAppServerSOAP (see page 1276) interface, it can't distinguish between multiple implementation classes.

See Also

IAppServer

TSoapConnection (see page 1128)

IAppServerSOAP (see page 1276)

14.1.1.1 TSoapDataModule Members

The following tables list the members exposed by TSoapDataModule.

Protected Methods

	Name	Description
	ApplyUpdates (see page 1151)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 1151)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 1152)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 1153)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 1154)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 1155)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 1155)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 1156)	Returns information from a specified record of the provider's dataset.
	DataRequest (see page 1157)	Generates an OnDataRequest event on the specified provider component.
	Execute (see page 1158)	Executes the query or stored procedure bound to a specified provider.
	GetParams (see page 1158)	Fetches current parameter values from the dataset bound to a specified provider.
	GetProvider (see page 1159)	Returns a registered provider component given its name.
	GetProviderNames (see page 1159)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 1160)	Returns a data packet that contains the specified data.
	RowRequest (see page 1162)	Returns information from a specified record of the provider's dataset.
	SAS_ApplyUpdates (see page 1163)	This is SAS_ApplyUpdates, a member of class TSoapDataModule.
	SAS_DataRequest (see page 1163)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 1164)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 1165)	Fetches current parameter values from the dataset bound to a specified provider.
	SAS_GetProviderNames (see page 1165)	Returns a list of all the available providers on the SOAP data module.
	SAS_GetRecords (see page 1166)	This is SAS_GetRecords, a member of class TSoapDataModule.
	SAS_RowRequest (see page 1166)	This is SAS_RowRequest, a member of class TSoapDataModule.

Public Methods

	Name	Description
	_AddRef (see page 1149)	Increments the reference count for this data module's interface.
	_Release (see page 1150)	Decrements the reference count for this data module's interface.
	AfterConstruction (see page 1150)	Generates an OnCreate event.
	BeforeDestruction (see page 1157)	Generates an OnDestroy event.
	Create (see page 1148)	Creates a SOAP data module instance.

	Destroy (see page 1148)	Frees a SOAP data module instance.
	Lock (see page 1161)	Locks the SOAP data module so that it can't be locked by other threads.
	NewInstance (see page 1161)	Allocates memory for each instance of TSoapDataModule (see page 1146).
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterProvider (see page 1162)	This is RegisterProvider, a member of class TSoapDataModule.
	SafeCallException (see page 1162)	Handles exceptions in methods declared using the safecall calling convention.
	Unlock (see page 1167)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1161) method.
	UnRegisterProvider (see page 1167)	Removes a provider from the list returned by AS_GetProviderNames (see page 1155).

Public Properties

	Name	Description
	ProviderCount (see page 1168)	Indicates the number of providers registered with the SOAP data module.
	Providers (see page 1168)	Lists the registered provider components that are contained in the SOAP data module.
	RefCount (see page 1168)	Reference counts the data module's interface.

14.1.1.2 TSoapDataModule.Create Constructor

Creates a SOAP data module instance.

Pascal

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

C++

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

Description

Most applications do not need to call the constructor for TSoapDataModule (see page 1146). Instead, add a SOAP data module to a Web service application using the SOAP data module wizard on the SOAP page of the new items dialog.

AOwner specifies the component that is responsible for freeing the SOAP data module (typically the application). If you supply an owner, the SOAP data module acts as a component, and is freed by that owner. If you pass nil for the AOwner parameter, the SOAP data module acts like an interfaced object, and maintains a reference count for every use of its interface. When that reference count goes to zero, the SOAP data module automatically frees itself. Note that is you set AOwner to nil but never obtain a reference to this object's interface, you must explicitly free the data module.

See Also

Create

14.1.1.3 TSoapDataModule.Destroy Destructor

Frees a SOAP data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapDataModule();
```

Description

Most applications do not need to call the destructor for TSoapDataModule (see page 1146). The SOAP data module is freed by its owner when the application shuts down.

If you must destroy a SOAP data module at runtime, call the Free method instead.

See Also

System::TObject::Free (see page 3382)

14.1.1.4 TSoapDataModule Methods

The methods of the TSoapDataModule class are listed here.

Protected Methods

	Name	Description
	ApplyUpdates (see page 1151)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 1151)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 1152)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 1153)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 1154)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 1155)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 1155)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 1156)	Returns information from a specified record of the provider's dataset.
	DataRequest (see page 1157)	Generates an OnDataRequest event on the specified provider component.
	Execute (see page 1158)	Executes the query or stored procedure bound to a specified provider.
	GetParams (see page 1158)	Fetches current parameter values from the dataset bound to a specified provider.
V	GetProvider (see page 1159)	Returns a registered provider component given its name.
	GetProviderNames (see page 1159)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 1160)	Returns a data packet that contains the specified data.
	RowRequest (see page 1162)	Returns information from a specified record of the provider's dataset.
V	SAS_ApplyUpdates (see page 1163)	This is SAS_ApplyUpdates, a member of class TSoapDataModule.
V	SAS_DataRequest (see page 1163)	Generates an OnSAS_DataRequest event on the specified provider component.
V	SAS_Execute (see page 1164)	SAS_Executes the query or stored procedure bound to a specified provider.
V	SAS_GetParams (see page 1165)	Fetches current parameter values from the dataset bound to a specified provider.
V	SAS_GetProviderNames (see page 1165)	Returns a list of all the available providers on the SOAP data module.
V	SAS_GetRecords (see page 1166)	This is SAS_GetRecords, a member of class TSoapDataModule.
V	SAS_RowRequest (see page 1166)	This is SAS_RowRequest, a member of class TSoapDataModule.

Public Methods

	Name	Description
	_AddRef (see page 1149)	Increments the reference count for this data module's interface.
	_Release (see page 1150)	Decrements the reference count for this data module's interface.
V	AfterConstruction (see page 1150)	Generates an OnCreate event.
V	BeforeDestruction (see page 1157)	Generates an OnDestroy event.
V	Lock (see page 1161)	Locks the SOAP data module so that it can't be locked by other threads.
V	NewInstance (see page 1161)	Allocates memory for each instance of TSoapDataModule (see page 1146).
	RegisterProvider (see page 1162)	This is RegisterProvider, a member of class TSoapDataModule.
V	SafeCallException (see page 1162)	Handles exceptions in methods declared using the safecall calling convention.
V	Unlock (see page 1167)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1161) method.
	UnRegisterProvider (see page 1167)	Removes a provider from the list returned by AS_GetProviderNames (see page 1155).

14.1.1.4.1 TSoapDataModule._AddRef Method

Increments the reference count for this data module's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

Do not call the __AddRef method directly. This method is called when an application obtains a reference to the SOAP data module's interface pointer. __AddRef increments the RefCount (see page 1168) property.

When the caller is through with the interface, it calls _Release (see page 1150). This allows the data module to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 1150)

RefCount (see page 1168)

IAppServer

14.1.1.4.2 TSoapDataModule._Release Method

Decrements the reference count for this data module's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

Do not call the __Release method directly. This method is called when an application releases a reference to the SOAP data module's interface pointer. __Release decrements the RefCount (see page 1168) property.

If the SOAP data module does not have an Owner, that is, if the value of the Owner property is nil (Delphi) or NULL (C++), then if the call to __Release changes the value of RefCount (see page 1168) to zero, the SOAP data module automatically frees itself.

See Also

_AddRef (see page 1149)

RefCount (see page 1168)

IAppServer

Owner

14.1.1.4.3 TSoapDataModule.AfterConstruction Method

Generates an OnCreate event.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called after the data module's constructor has finished. Do not call it explicitly in applications.

TDataModule overrides AfterConstruction to generate an OnCreate event when OldCreateOrder is false.

See Also

BeforeDestruction

OnCreate

OldCreateOrder

14.1.1.4.4 TSoapDataModule.ApplyUpdates Method

Applies updates received from a client using a specified provider.

Pascal

```
function ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors: Integer;
out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int MaxErrors,
int ErrorCount, OleVariant OwnerData);
```

Description

ApplyUpdates provides the underlying implementation for the AS_ApplyUpates method (on the IAppServer interface) and the SAS_ApplyUpdates (see page 1163) method (on the IAppServerSoap interface). It writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant (see page 3430) containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before rolling back the update operation

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeApplyUpdates event handler and returned to the client application in an AfterApplyUpdates event handler. It is passed to the provider's BeforeApplyUpdates event handler and returned from the provider's AfterApplyUpdates event handler.

ApplyUpdates returns a Variant (see page 3430) that is a data packet containing all records that could not be applied.

See Also

as_applyupdates (see page 1151)

sas_applyupdates (see page 1163)

14.1.1.4.5 TSoapDataModule.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
```

```
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates

applyupdates

BeforeApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

AS_ApplyUpdates

as_applyupdates

Communicating with the Client Dataset

14.1.1.4.6 TSoapDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or AS_DataRequest_result parameter (C++).

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest

Providers (see page 1168)

DataRequest

IAppServer

14.1.1.4.7 TSoapDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output (see page 3901) parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: AS_AS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

Note: Applications can only call the protected AS_AS_Execute method using the IAppServer interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet

beforeexecute

AfterExecute

Params

Execute

Execute

Providers (see page 1168)

IAppServer

BeforeExecute

AfterExecute

PackageParams

Options

14.1.1.4.8 TSoapDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider. The ProviderName parameter specifies the name of the provider component whose dataset has the parameters. The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

Note: In Delphi, AS_AS_GetParams returns the parameters, encoded as a Variant (see page 3430) array, or, if the provider does not support parameter fetching, a Null (see page 5075) Variant (see page 3430).

Note: In C++, the AS_GetParams_result parameter the Variant (see page 3430) array containing the parameters (or the Null (see page 5075) Variant (see page 3430)).

Use the UnpackParams procedure to convert parameter values returned by AS_AS_GetParams to a TParams object.

Warning: It is not a good idea to call AS_AS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the AS_Execute (see page 1153) method instead.

Note: Applications can only call the protected AS_AS_GetParams method using the IAppServer interface.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

AfterGetParams

Providers (see page 1168)

IAppServer

AS_Execute (see page 1153)

UnpackParams

14.1.1.4.9 TSoapDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant (see page 3430) array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName

Providers (see page 1168)

IAppServer

Exported

14.1.1.4.10 TSoapDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. Because SOAP data modules are stateless, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant (see page 3430) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant (see page 3430) (as the return value in Delphi or the AS_GetRecords_result parameter in C++).

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data

Params

GetNextPacket

GetRecords

BeforeGetRecords

AfterGetRecords

Providers (see page 1168)

IAppServer

14.1.1.4.11 TSoapDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant (see page 3281) that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
```

```
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchDetails

RefreshRecord

Providers (see page 1168)

IAppServer

14.1.1.4.12 TSoapDataModule.BeforeDestruction Method

Generates an OnDestroy event.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called immediately before the data module's destructor. Do not call it explicitly in applications.

TDataModule overrides BeforeDestruction to generate an OnDestroy event when OldCreateOrder is false.

See Also

AfterConstruction

OnDestroy

OldCreateOrder

14.1.1.4.13 TSoapDataModule.DataRequest Method

Generates an OnDataRequest event on the specified provider component.

Pascal

```
function DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

DataRequest provides the underlying implementation for the AS_DataRequest (see page 1152) method (on the IAppServer interface) and the SAS_DataRequest (see page 1163) method (on the IAppServerSOAP (see page 1276) interface). This method calls any OnDataRequest event handler of the specified provider.

There is no predefined meaning for the Data parameter. It represents custom information supplied by the client and passed to the OnDataRequest event handler.

DataRequest returns the value that is assigned to the OwnerData parameter of the event handler on exit.

See Also

AS_DataRequest (see page 1152)

SAS_DataRequest (see page 1163)

OnDataRequest

14.1.1.4.14 TSoapDataModule.Execute Method

Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure Execute(const ProviderName: WideString; const CommandText: WideString; var Params:
OleVariant; var OwnerData: OleVariant);
```

C++

```
__fastcall Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Execute provides the underlying implementation for the AS_Execute (see page 1153) method (on the IAppServer interface) and the SAS_Execute (see page 1164) method (on the IAppServerSOAP (see page 1276) interface). It calls the Execute method of the specified provider after assigning any parameters. Output (see page 3901) parameters that result from executing a query or stored procedure are returned in Params.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeExecute and AfterExecute event handlers. This information originates in a client dataset's BeforeExecute event handler. The value returned in OwnerData is passed to the client dataset's AfterExecute event handler.

See Also

AS_Execute (see page 1153)

SAS_Execute (see page 1164)

14.1.1.4.15 TSoapDataModule.GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

GetParams provides the underlying implementation of the AS_GetParams (see page 1154) method (on the IAppServer interface) and the SAS_GetParams (see page 1165) method (on the IAppServerSOAP (see page 1276) interface).

ProviderName specifies the name of the provider component whose dataset has the parameters to fetch.

OwnerData represents custom information that originates in a client dataset's BeforeGetParams event handler and returns information that is passed to the client dataset's AfterGetParams event handler.

GetParams returns the parameters, encoded as a Variant (see page 3430) array, or, if the provider does not support parameter fetching, GetParams returns a Null (see page 5075) Variant (see page 3430). Use the UnpackParams procedure to convert parameter values returned by GetParams to a TParams object.

See Also

SAS_GetParams (see page 1165)

AS_GetParams (see page 1154)

14.1.1.4.16 TSoapDataModule.GetProvider Method

Returns a registered provider component given its name.

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const UnicodeString ProviderName);
```

Description

GetProvider is the protected read implementation of the Providers (see page 1168) property.

ProviderName specifies the name of a provider that is returned. If ProviderName is not one of the names returned by AS_GetProviderNames (see page 1155) (for example, if the provider's Exported property is false), GetProvider raises an exception.

See Also

Providers (see page 1168)

Exported

AS_GetProviderNames (see page 1155)

14.1.1.4.17 TSoapDataModule.GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function GetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant GetProviderNames();
```

Description

GetProviderNames provides the underlying implementation for the AS_GetProviderNames (see page 1155) method (on the IAppServer interface) and the SAS_GetProviderNames (see page 1165) method (on the IAppServerSOAP (see page 1276) interface). It returns a list of all registered providers available from a SOAP data module.

GetProviderNames returns the names of the providers in a Variant (see page 3430) array.

See Also

AS_GetProviderNames (see page 1155)

SAS_GetProviderNames (see page 1165)

14.1.1.4.18 TSoapDataModule.GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetRecords(const BSTR ProviderName, int Count, int RecsOut, int Options,
const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

GetRecords provides the underlying implementation for the AS_GetRecords (see page 1155) method (on the IAppServer interface) and the SAS_GetRecords (see page 1166) method (on the IAppServerSOAP (see page 1276) interface). GetRecords returns the requested records, starting with the current record of the provider's dataset.

- ProviderName gives the name of the provider component that supplies the records.
- Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.
- RecsOut returns the actual number of records retrieved.
- Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants.:
- CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.
- Params is a Variant (see page 3430) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.
- OwnerData contains custom information that is supplied by a client dataset's BeforeGetRecords event handler. This information is passed to the provider's BeforeGetRecords event handler. OwnerData returns information supplied by the provider's AfterGetRecords event handler.
- Records are returned as a data packet in a Variant (see page 3430).

See Also

AS_GetRecords (see page 1155)

sas_getrecords (see page 1166)

14.1.1.4.19 TSoapDataModule.Lock Method

Locks the SOAP data module so that it can't be locked by other threads.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

SOAP data modules use Lock internally to ensure that all IAppServer calls are thread-safe. When implementing a SOAP data module descendant, call Lock before accessing any instance data, including the providers contained in the data module. When finished with the block of code that must be protected from thread conflicts, call the Unlock (see page 1167) method to release the lock.

Lock uses a critical section to guard against thread conflicts. This means that any code that accesses the SOAP data module's properties or contained objects must call Lock or it will introduce thread conflicts.

See Also

Providers (see page 1168)

Unlock (see page 1167)

IAppServer

TCriticalSection (see page 3068)

14.1.1.4.20 TSoapDataModule.NewInstance Method

Allocates memory for each instance of TSoapDataModule (see page 1146).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSoapDataModule (see page 1146) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1150) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3391)

System::TObject::InstanceSize (see page 3389)

afterconstruction (see page 1150)

14.1.1.4.21 TSoapDataModule.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Value);
```

Description

This is RegisterProvider, a member of class TSoapDataModule.

14.1.1.4.22 TSoapDataModule.RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer; var
OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

RowRequest provides the underlying implementation for the AS_RowRequest (see page 1156) method (on the IAppServer interface) and the SAS_RowRequest (see page 1166) method (on the IAppServerSOAP (see page 1276) interface).

ProviderName indicates the provider associated with the dataset from which information should be fetched.

Row is an OleVariant (see page 3281) that describes the current record on the client dataset.

RequestType indicates the type of information required. It is an integer version of the TFetchOptions type.

OwnerData contains custom information that is supplied by a client dataset's BeforeRowRequest event handler. This information is passed to the provider's BeforeRowRequest event handler. OwnerData returns custom information supplied by the provider's AfterRowRequest event handler.

The requested data is returned as a delta packet.

See Also

AS_RowRequest (see page 1156)

sas_rowrequest (see page 1166)

14.1.1.4.23 TSoapDataModule.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException is used for handling exceptions that are raised by methods such as the AS... methods, which use the safecall

calling convention. This method ensures that the SOAP fault packet includes all the necessary error information so that clients can respond appropriately to exceptions in safecall methods.

ExceptObject (see page 3664) is the exception that was raised.

ExceptAddr (see page 3663) is the address where the exception occurred.

SafeCallException returns an error code that indicates what error occurred.

See Also

SafeCallException

14.1.1.4.24 TSoapDataModule.SAS_ApplyUpdates Method

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant
Delta, int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is SAS_ApplyUpdates, a member of class TSoapDataModule.

14.1.1.4.25 TSoapDataModule.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant
Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

Note: Applications can only call the protected SAS_SAS_DataRequest method using the IAppServerSOAP (see page 1276) interface.

See Also

OnDataRequest

Providers (see page 1168)

DataRequest

IAppServerSOAP (see page 1276)

14.1.1.4.26 TSoapDataModule.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls SAS_Execute, which calls the SAS_Execute method of the specified provider after assigning any parameters. Output (see page 3901) parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: SAS_SAS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

Note: Applications can only call the protected SAS_SAS_Execute method using the IAppServerSOAP (see page 1276) interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet

beforeexecute

AfterExecute

Params

Execute

Execute

Providers (see page 1168)

IAppServerSOAP (see page 1276)

BeforeExecute

AfterExecute

PackageParams

Options

14.1.1.4.27 TSoapDataModule.SAS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant
OwnerData);
```

Description

Use SAS_SAS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

ProviderName specifies the name of the provider component whose dataset has the parameters.

OwnerData represents custom information that originates in a client dataset's BeforeSAS_GetParams event handler and returns information that is passed to the client dataset's AfterSAS_GetParams event handler.

SAS_SAS_GetParams returns the parameters, encoded as a Variant (see page 3430) array, or, if the provider does not support parameter fetching, SAS_SAS_GetParams returns a Null (see page 5075) Variant (see page 3430). Use the UnpackParams procedure to convert parameter values returned by SAS_SAS_GetParams to a TParams object.

Warning: It is not a good idea to call SAS_SAS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the SAS_Execute (see page 1164) method instead.

Note: Applications can only call the protected SAS_SAS_GetParams method using the IAppServerSOAP (see page 1276) interface.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

AfterGetParams

Providers (see page 1168)

IAppServerSOAP (see page 1276)

SAS_Execute (see page 1164)

UnpackParams

14.1.1.4.28 TSoapDataModule.SAS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP (see page 1276) interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected SAS_SAS_GetProviderNames method using the IAppServerSOAP (see page 1276) interface.

See Also

ProviderName

Providers (see page 1168)

IAppServerSOAP (see page 1276)

Exported

14.1.1.4.29 TSoapDataModule.SAS_GetRecords Method

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is SAS_GetRecords, a member of class TSoapDataModule.

14.1.1.4.30 TSoapDataModule.SAS_RowRequest Method

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant
Row, int RequestType, OleVariant OwnerData);
```

Description

This is SAS_RowRequest, a member of class TSoapDataModule.

14.1.1.4.31 TSoapDataModule.Unlock Method

Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1161) method.

Pascal

```
procedure Unlock; virtual;
```

C++

```
virtual __fastcall Unlock();
```

Description

Use Unlock to release the critical section that was entered by a call to the Lock (see page 1161) method. Every call to the Lock (see page 1161) method must be paired with a corresponding call to Unlock.

See Also

Lock (see page 1161)

14.1.1.4.32 TSoapDataModule.UnRegisterProvider Method

Removes a provider from the list returned by AS_GetProviderNames (see page 1155).

Pascal

```
procedure UnRegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Value);
```

Description

Call UnRegisterProvider to remove a provider from the SOAP data module's list of available providers. Pass a reference to the provider that should be removed as the Value parameter. When a provider is not registered, it is not returned by the AS_GetProviderNames (see page 1155) method, which can be accessed through the IAppServer interface.

To temporarily disable a provider, use the provider's Exported property. Unexported providers can't be accessed through any IAppServer method calls.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider method. Providers (see page 1168) that have the SOAP data module as an Owner are unregistered automatically from their destructors.

See Also

registerprovider (see page 1162)

AS_GetProviderNames (see page 1155)

IAppServer

Destroy

Exported

14.1.1.5 TSoapDataModule Properties

The properties of the TSoapDataModule class are listed here.

Public Properties

	Name	Description
	ProviderCount (see page 1168)	Indicates the number of providers registered with the SOAP data module.
	Providers (see page 1168)	Lists the registered provider components that are contained in the SOAP data module.
	RefCount (see page 1168)	Reference counts the data module's interface.

14.1.1.5.1 TSoapDataModule.ProviderCount Property

Indicates the number of providers registered with the SOAP data module.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

ProviderCount is the number of providers that can be accessed using the Providers (see page 1168) property.

See Also

Providers (see page 1168)

registerprovider (see page 1162)

14.1.1.5.2 TSoapDataModule.Providers Property

Lists the registered provider components that are contained in the SOAP data module.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[UnicodeString const ProviderName];
```

Description

Use Providers to access a provider component from the SOAP data module given its name. The SOAP data module uses Providers to locate the target of method calls made using the IAppServer interface.

If the value passed as is not one of the names returned by AS_GetProviderNames (see page 1155) (for example, if the provider's Exported property is false) reading Providers raises an exception.

See Also

AS_GetProviderNames (see page 1155)

Exported

registerprovider (see page 1162)

ProviderCount (see page 1168)

14.1.1.5.3 TSoapDataModule.RefCount Property

Reference counts the data module's interface.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount keeps track of the number of times the application takes a reference to the SOAP data module's interface. When application code obtains this data module's interface, a call is generated to the _AddRef (see page 1149) method, which increments RefCount. When application code releases the interface, a call is generated to the _Release (see page 1150) method, which decrements RefCount. When RefCount drops to zero, the SOAP data module frees itself if its Owner property is not nil (Delphi) or NULL (C++).

See Also

_AddRef (see page 1149)

_Release (see page 1150)

Owner

15 SOAPDomConv Namespace

This is namespace SOAPDomConv.

Topics

Name	Description
Classes (see page 1172)	The following table lists classes in this documentation.
Interfaces (see page 1179)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1182)	The following table lists structs, records, enums in this documentation.
Types (see page 1183)	The following table lists types in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 1172)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 1179)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Structs, Records, Enums

	Name	Description
	TDOMHeaderProcessorEntry (see page 1182)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

Types

Name	Description
TDOMHeaderProcessorArray (see page 1183)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

15.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 1172)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

15.1.1 TSOAPDOMProcessor Class

TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Class Hierarchy

Pascal

```
TSOAPDOMProcessor = class(TComponent, IInterface);
```

C++

```
class TSOAPDOMProcessor : public TComponent, public IInterface;
```

File

SOAPDomConv

Description

TSOAPDOMProcessor is the base class for DOM-based components that interpret and create SOAP method calls. It manages a list of interfaces that it uses to create or interpret specialized headers in the SOAP message. Such headers can include application-specific information such as authentication, compression/decompression, encryption/decryption, and so on.

Do not create instances of TSOAPDOMProcessor. It is intended only as a base class for components that use the information in SOAP headers when interpreting or creating SOAP method calls.

See Also

TOPToSoapDomConvert

15.1.1.1 TSOAPDOMProcessor Members

The following tables list the members exposed by TSOAPDOMProcessor.

Protected Fields

	Name	Description
	FHeaderProcessors (see page 1173)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

Protected Methods

	Name	Description
	_AddRef (see page 1174)	Called when an application uses a component interface.
	_Release (see page 1175)	Called when an application releases a component interface.
	FindHeaderProcessor (see page 1177)	Locates a processor to interpret or generate a specified SOAP header.
	MakeHeaderNodes (see page 1177)	Generates all the SOAP headers that this object knows how to create.

Public Methods

	Name	Description
	AddHeaderProcessor (see page 1175)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
	AfterConstruction (see page 1176)	Responds after the last constructor executes.
	DefaultProcessHeader (see page 1176)	Handles the interpretation of a SOAP header when no processor can be found.
	Destroy (see page 1173)	Frees the memory associated with the TSOAPDOMProcessor (see page 1172) object.
	NewInstance (see page 1177)	Allocates memory for each instance of TSOAPDOMProcessor (see page 1172).
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

15.1.1.2 TSOAPDOMProcessor.Destroy Destructor

Frees the memory associated with the TSOAPDOMProcessor (see page 1172) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPDOMProcessor();
```

Description

Do not call Destroy to free an instance of TSOAPDOMProcessor (see page 1172). Instead, use the Free method, which first checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3382)

15.1.1.3 TSOAPDOMProcessor Fields

The fields of the TSOAPDOMProcessor class are listed here.

Protected Fields

	Name	Description
	FHeaderProcessors (see page 1173)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

15.1.1.3.1 TSOAPDOMProcessor.FHeaderProcessors Field

Pascal

```
FHeaderProcessors: TDOMHeaderProcessorArray;
```

C++

```
TDOMHeaderProcessorArray FHeaderProcessors;
```

Description

This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

15.1.1.4 TSOAPDOMProcessor Methods

The methods of the TSOAPDOMProcessor class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 1174)	Called when an application uses a component interface.
	_Release (see page 1175)	Called when an application releases a component interface.
V	FindHeaderProcessor (see page 1177)	Locates a processor to interpret or generate a specified SOAP header.
	MakeHeaderNodes (see page 1177)	Generates all the SOAP headers that this object knows how to create.

Public Methods

	Name	Description
V	AddHeaderProcessor (see page 1175)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
V	AfterConstruction (see page 1176)	Responds after the last constructor executes.
V	DefaultProcessHeader (see page 1176)	Handles the interpretation of a SOAP header when no processor can be found.
V	NewInstance (see page 1177)	Allocates memory for each instance of TSOAPDOMProcessor (see page 1172).

15.1.1.4.1 TSOAPDOMProcessor._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke

QueryInterface

_Release

ComObject

_AddRef (see page 3504)

15.1.1.4.2 TSOAPDOMProcessor._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke

QueryInterface

_AddRef

ComObject

15.1.1.4.3 TSOAPDOMProcessor.AddHeaderProcessor Method

Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.

Pascal

```
procedure AddHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString; Processor: IDOMHeaderProcessor); virtual;
```

C++

```
virtual __fastcall AddHeaderProcessor(BSTR Namespace, BSTR HeaderName, BSTR TypeName,
IDOMHeaderProcessor Processor);
```

Description

AddHeaderProcessor stores a specified processor interface and the information necessary to identify the SOAP headers that it handles.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Processor is the interface for a helper object that creates or interprets SOAP headers that match Namespace, HeaderName, and TypeName.

Note: AddHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 1177)

15.1.1.4.4 TSOAPDOMProcessor.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of TSOAPDOMProcessor (see page 1172) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

System::TObject::BeforeDestruction (see page 3368)

NewInstance (see page 1177)

15.1.1.4.5 TSOAPDOMProcessor.DefaultProcessHeader Method

Handles the interpretation of a SOAP header when no processor can be found.

Pascal

```
procedure DefaultProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest:
Boolean); virtual;
```

C++

```
virtual __fastcall DefaultProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean
AbortRequest);
```

Description

DefaultProcessHeader is called to provide a default interpretation of a SOAP header when it is not processed by one of the processors that FindHeaderProcessor (see page 1177) returns.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if DefaultProcessHeader handles the header node.

AbortRequest returns true if DefaultProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request.

Note: DefaultProcessHeader is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 1177)

15.1.1.4.6 TSOAPDOMProcessor.FindHeaderProcessor Method

Locates a processor to interpret or generate a specified SOAP header.

Pascal

```
function FindHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString): IDOMHeaderProcessor; virtual;
```

C++

```
virtual __fastcall IDOMHeaderProcessor FindHeaderProcessor(BSTR Namespace, BSTR HeaderName,
BSTR TypeName);
```

Description

FindHeaderProcessor returns the interface of the helper object that can generate or interpret a specified SOAP header. This processor must be previously added to the TSOAPDOMProcessor (see page 1172) object's internal list of processors by a call to the AddHeaderProcessor (see page 1175) method.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Note: FindHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 1175)

15.1.1.4.7 TSOAPDOMProcessor.MakeHeaderNodes Method

Generates all the SOAP headers that this object knows how to create.

Pascal

```
function MakeHeaderNodes(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode MakeHeaderNodes(IXMLNode HeaderNode);
```

Description

MakeHeaderNodes uses the processors that have been added by the AddHeaderProcessor (see page 1175) method to generate a set of SOAP header nodes. These generated nodes are added as child nodes to the node specified by HeaderNode.

Note: MakeHeaderNodes is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 1175)

15.1.1.4.8 TSOAPDOMProcessor.NewInstance Method

Allocates memory for each instance of TSOAPDOMProcessor (see page 1172).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSOAPDOMProcessor (see page 1172) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1176) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3391)

System::TObject::InstanceSize (see page 3389)

AfterConstruction (see page 1176)

15.2 Interfaces

The following table lists interfaces in this documentation.

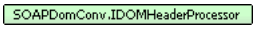
Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 1179)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

15.2.1 IDOMHeaderProcessor Interface

IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Class Hierarchy



Pascal

```
IDOMHeaderProcessor = interface;
```

C++

```
__interface IDOMHeaderProcessor;
```

File

SOAPDomConv

Description

IDOMHeaderProcessor provides an interface for working with specialized SOAP headers. Each object that implements IDOMHeaderProcessor handles a specific type of soap header. The TSOAPDOMProcessor (see page 1172) class calls on this interface when it needs to create or interpret the headers of a SOAP packet.

IDOMHeaderProcessor declares two methods, CreateHeader (see page 1180) and ProcessHeader (see page 1180), which create or respond to a given SOAP header, respectively.

Note: C++ method declarations that use IDOMHeaderProcessor use the _di_IDOMHeaderProcessor type instead. This type is a DelphiInterface (see page 3263) wrapper around the IDOMHeaderProcessor interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMHeaderProcessor >
_di_IDOMHeaderProcessor;
```

Note: IDOMHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using the ISOAPHeaders interface.

See Also

TSoapDomProcessor

ISOAPHeaders

15.2.1.1 IDOMHeaderProcessor Members

The following tables list the members exposed by IDOMHeaderProcessor.

Public Methods

	Name	Description
	CreateHeader (see page 1180)	Creates a new SOAP header node.
	ProcessHeader (see page 1180)	Interprets and responds to a SOAP header.

15.2.1.2 IDOMHeaderProcessor Methods

The methods of the IDOMHeaderProcessor class are listed here.

Public Methods

	Name	Description
	CreateHeader (see page 1180)	Creates a new SOAP header node.
	ProcessHeader (see page 1180)	Interprets and responds to a SOAP header.

15.2.1.2.1 IDOMHeaderProcessor.CreateHeader Method

Creates a new SOAP header node.

Pascal

```
function CreateHeader(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateHeader(IXMLNode HeaderNode);
```

Description

CreateHeader creates a new header node.

HeaderNode is the node in the emerging SOAP packet that acts as a parent to the new header.

CreateHeader returns the new header node.

See Also

MakeHeaderNodes (see page 1177)

15.2.1.2.2 IDOMHeaderProcessor.ProcessHeader Method

Interprets and responds to a SOAP header.

Pascal

```
procedure ProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest: Boolean);
```

C++

```
__fastcall ProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean AbortRequest);
```

Description

ProcessHeader provides a response to a given SOAP header node.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if ProcessHeader handles the header node.

AbortRequest returns true if ProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request. For example, if the header node represents authentication information, ProcessHeader could set AbortRequest to true if the header supplied an invalid password.

15.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TDOMHeaderProcessorEntry (see page 1182)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

15.3.1 SOAPDomConv.TDOMHeaderProcessorEntry Record

Pascal

```
TDOMHeaderProcessorEntry = record
  Processor: IDOMHeaderProcessor;
  NameSpace: WideString;
  HeaderName: WideString;
  TypeName: WideString;
end;
```

C++

```
struct TDOMHeaderProcessorEntry {
  IDOMHeaderProcessor Processor;
  BSTR NameSpace;
  BSTR HeaderName;
  BSTR TypeName;
};
```

File

SOAPDomConv

Description

This is record SOAPDomConv.TDOMHeaderProcessorEntry.

15.4 Types

The following table lists types in this documentation.

Types

Name	Description
TDOMHeaderProcessorArray (see page 1183)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

15.4.1 SOAPDomConv.TDOMHeaderProcessorArray Type

Pascal

```
TDOMHeaderProcessorArray = array of TDOMHeaderProcessorEntry;
```

C++

```
array of TDOMHeaderProcessorEntry TDOMHeaderProcessorArray;
```

File

SOAPDomConv

Description

This is type SOAPDomConv.TDOMHeaderProcessorArray.

16 SOAPHTTPClient Namespace

This is namespace SOAPHTTPClient.

Topics

Name	Description
Classes (see page 1186)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 1186)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

16.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 1186)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

16.1.1 THTTPRIO Class

THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

Class Hierarchy

Pascal

```
THTTPRIO = class(TRIO);
```

C++

```
class THTTPRIO : public TRIO;
```

File

SOAPHTTPClient

Description

Use THTTPRIO to generate statically-linked calls to invokable interfaces on a remote Web Service (see page 1191) application. When an application casts THTTPRIO to a registered invokable interface, it dynamically generates an in-memory method table, providing an implementation to that invokable interface. THTTPRIO executes the methods in this method table by encoding the method call as a SOAP request and sending an HTTP request message to the Web Service (see page 1191) application. It unpacks the resulting HTTP response message to obtain the return value and any output parameters, or to raise an exception if the request generated an exception on the server.

Use the published properties of THTTPRIO to indicate how connect to the Web Service (see page 1191) application. There are two ways to identify where the server application is located:

You can use the URL (see page 1192) property to specify the URL (see page 1192) where the server application is located,.

If you want to look up connection information dynamically at runtime from a WSDL document, you can set the WSDLLocation (see page 1193) property. After setting WSDLLocation (see page 1193), select values for the Service (see page 1191) and Port (see page 1191) properties from drop-down lists in the object inspector to fully specify the binding you want to use.

If you need to use a proxy server, or if the server requires authentication, use the properties of the THTTPReqResp (see page 1214) object that is the value of the HTTPWebNode (see page 1190) property to provide the necessary information.

Before THTTPRIO can generate a method table for an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry

Calling Invokable Interfaces

Understanding Invokable Interfaces

16.1.1.1 THTTPRIO Members

The following tables list the members exposed by THTTPRIO.

Protected Methods

	Name	Description
	Notification (see page 1188)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Create (see page 1187)	Creates an instance of THTTPRIO (see page 1186).
	Destroy (see page 1188)	Destroys the instance of a THTTPRIO (see page 1186) object.
	QueryInterface (see page 1189)	Implements the IInterface (see page 3503) QueryInterface method.

Public Properties

	Name	Description
	WSDLItems (see page 1193)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1193).

Published Properties

	Name	Description
	Converter (see page 1190)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 1190)	Handles the execution of method calls on the invokable interface.
	Port (see page 1191)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 1191)	Specifies the Service to use when sending the SOAP message to execute a method call.
	URL (see page 1192)	Specifies the URL that identifies the target Web Service (see page 1191) application.
	WSDLLocation (see page 1193)	Specifies the location of the WSDL document that describes the target Web Service (see page 1191).

16.1.1.2 THTTPRIO.Create Constructor

Creates an instance of THTTPRIO (see page 1186).

Pascal

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

C++

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

Description

Call Create to instantiate a THTTPRIO (see page 1186) object at runtime. THTTPRIO (see page 1186) components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the THTTPRIO (see page 1186) instance. It becomes the value of the Owner property. If the THTTPRIO (see page 1186) instance is created with AOwner set to nil (Delphi) or NULL (C++), then it is automatically freed when its reference count drops to zero.

See Also

RefCount

Owner

16.1.1.3 THTTPRIO.Destroy Destructor

Destroys the instance of a THTTPRIO (see page 1186) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPRIO();
```

Description

There is no need to explicitly free a THTTPRIO (see page 1186) instance.

If the THTTPRIO (see page 1186) object is created with an Owner, then that Owner handles the freeing of the THTTPRIO (see page 1186) instance. If the THTTPRIO (see page 1186) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 1187)

Owner

RefCount

16.1.1.4 THTTPRIO Methods

The methods of the THTTPRIO class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1188)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	QueryInterface (see page 1189)	Implements the IInterface (see page 3503) QueryInterface method.

16.1.1.4.1 THTTPRIO.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

16.1.1.4.2 THTTPRIO.QueryInterface Method

Implements the IInterface (see page 3503) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface (see page 3503) interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts THTTPRIO (see page 1186) to an invokable interface using the as operator.

THTTPRIO (see page 1186) can only return a single type of interface pointer. The first time an application calls QueryInterface, THTTPRIO (see page 1186) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this THTTPRIO (see page 1186) instance. If the THTTPRIO (see page 1186) object has already been typecast to an interface, then IID must identify that interface. If the THTTPRIO (see page 1186) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless THTTPRIO (see page 1186) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK (see page 3927)) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE (see page 3912) otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface (see page 3263) wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry

RegisterInterface

RefCount

16.1.1.5 THTTPRIO Properties

The properties of the THTTPRIO class are listed here.

Public Properties

	Name	Description
	WSDLItems (see page 1193)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1193).

Published Properties

	Name	Description
	Converter (see page 1190)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 1190)	Handles the execution of method calls on the invokable interface.
	Port (see page 1191)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 1191)	Specifies the Service to use when sending the SOAP message to execute a method call.
	URL (see page 1192)	Specifies the URL that identifies the target Web Service (see page 1191) application.
	WSDLLocation (see page 1193)	Specifies the location of the WSDL document that describes the target Web Service (see page 1191).

16.1.1.5.1 THTTPRIO.Converter Property

Handles the marshaling and unmarshaling of calls to an invokable interface.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

THTTPRIO (see page 1186) uses Converter internally to marshal and unmarshal calls to an invokable interface. That is, when the THTTPRIO (see page 1186) object has been typecast to an invokable interface, it implements each method call by using Converter to marshal the call from an internal format into an encoding protocol, and to unmarshal any return values after that call is executed.

See Also

HTTPWebNode (see page 1190)

16.1.1.5.2 THTTPRIO.HTTPWebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property HTTPWebNode: THTTPReqResp;
```

C++

```
__property THTTPReqResp HTTPWebNode;
```

Description

THTTPRIO (see page 1186) uses HTTPWebNode internally to execute method calls. It is the implementation of the WebNode (see page 6581) property.

THTTPRio executes a method call by sending a SOAP request as an HTTP message to the Web Service (see page 1191) identified by either the URL (see page 1192) property or the WSDLLocation (see page 1193), Service (see page 1191), and Port (see page 1191) properties. If you need to use a proxy server to access the Web Service (see page 1191) application, or if the server requires authentication, use the properties of the THTTPReqResp (see page 1214) object to provide the necessary information.

See Also

WebNode

URL (see page 1192)

WSDLLocation (see page 1193)

Service (see page 1191)

Port (see page 1191)

16.1.1.5.3 THTTPRIO.Port Property

Specifies the port to use when sending the SOAP message to execute a method call.

Pascal

```
property Port: string;
```

C++

```
__property UnicodeString Port;
```

Description

Set Port to indicate the desired port within the service specified by the Service (see page 1191) property. Port is the name of a port defined in the WSDL document specified by the WSDLLocation (see page 1193) property. Once you have set the WSDLLocation (see page 1193) property, you can select a value for Port from a drop-down list in the Object Inspector.

A port defines a binding (which indicates the interface to call and the encoding and transport protocols to use) and the address of the server machine.

See Also

WSDLLocation (see page 1193)

Service (see page 1191)

HTTPWebNode (see page 1190)

16.1.1.5.4 THTTPRIO.Service Property

Specifies the Service to use when sending the SOAP message to execute a method call.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Set Service to indicate the Service that you want to use by making a remote method call. Service is the name of a Service defined in the WSDL document specified by the WSDLLocation (see page 1193) property. Once you have set the WSDLLocation (see page 1193) property, you can select a value for Service from a drop-down list in the Object Inspector.

A Service is a set of related interfaces provided by a Web Service application. The interface that THTTPRIO (see page 1186) represents must be on one of the interfaces that make up the specified service.

Note: When defining the target Web Service application by referring to a WSDL document, you must set the Port (see page 1191) property as well as the Service and WSDLLocation (see page 1193) properties.

See Also

WSDLLocation (see page 1193)

Port (see page 1191)

HTTPWebNode (see page 1190)

16.1.1.5.5 THTTPRIO.URL Property

Specifies the URL that identifies the target Web Service (see page 1191) application.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to identify the target application that implements the invokable interface when there is no need to look up connection information from a WSDL document dynamically at runtime. If the server was written using the VCL, the path portion of this URL must match the target path name of the THTTPSoapDispatcher component on the Web Service (see page 1191) application. The protocol portion must specify http or https. Thus, for servers written in the VCL, URL has the form

`http://MyHost.com/scripts/MyWebService.dllso/SOAP/IMyInterface`

When you use URL to identify the server, THTTPRIO (see page 1186) obtains a namespace for the target interface from the invocation registry. It is necessary, therefore, that the target interface be registered with the same namespace in both client and server applications

Note: In Delphi, this happens automatically if you

Use the same unit to declare and register the target interface in both client and server applications.

Use the same value for the AppNamespacePrefix global variable.

Use the automatically-generated namespace.

If you are using C++, or if one of the above criteria is not met, you can ensure that the target interface is registered with the same namespace by explicitly specifying a namespace in the code that registers the interface.

Note: If you want your application to look up connection information from a WSDL document dynamically at runtime, you must use the WSDLLocation (see page 1193), Service (see page 1191), and Port (see page 1191) properties instead. Using the URL property is mutually exclusive with the use of WSDLLocation (see page 1193), Service (see page 1191) and Port (see page 1191).

See Also

WSDLLocation (see page 1193)

WebDispatch (see page 5924)

HTTPWebNode (see page 1190)

AppNamespacePrefix

16.1.1.5.6 THTTPRIO.WSDLItems Property

Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1193).

Pascal

```
property WSDLItems: TWSDLItems;
```

C++

```
__property TWSDLItems WSDLItems;
```

Description

WSDLItems is a helper object whose methods return the definitions included in the WSDL file specified by WSDLLocation (see page 1193). Applications should not need to use this object. Rather, THTTPRio uses this property to obtain information about how to connect to the Web Service (see page 1191) application and how to encode request messages.

See Also

WSDLLocation (see page 1193)

16.1.1.5.7 THTTPRIO.WSDLLocation Property

Specifies the location of the WSDL document that describes the target Web Service (see page 1191).

Pascal

```
property WSDLLocation: string;
```

C++

```
__property UnicodeString WSDLLocation;
```

Description

Use WSDLLocation to specify a WSDL document that defines the target interface (port type). WSDLLocation can be either a fully qualified path name, or a URL (see page 1192) that indicates where the WSDL document can be found.

When using a WSDL document to indicate the target Web Service (see page 1191), you must also specify values for the Service (see page 1191) and Port (see page 1191) properties. Once WSDLLocation is set, you can use a drop-down list in the Object Inspector to select values for Port (see page 1191) and Service (see page 1191).

Note: If you do not need to look up connection information from the WSDL document at runtime, you can use the URL (see page 1192) property instead. Using the WSDLLocation property is mutually exclusive with the use of the URL (see page 1192) property.

See Also

WSDLLocation

Port (see page 1191)

Service (see page 1191)

URL (see page 1192)

17 SOAPHTTPDisp Namespace

This is namespace SOAPHTTPDisp.

Topics

Name	Description
Classes (see page 1196)	The following table lists classes in this documentation.
Interfaces (see page 1200)	The following table lists interfaces in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 1196)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 1200)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

17.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 1196)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

17.1.1 THTTPSoapDispatchNode Class

THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Class Hierarchy

Pascal

```
THTTPSoapDispatchNode = class(TComponent);
```

C++

```
class THTTPSoapDispatchNode : public TComponent;
```

File

SOAPHTTPDisp

Description

Use THTTPSoapDispatchNode as a base class for objects that use an invoker component to interpret and execute SOAP message calls. THTTPSoapDispatchNode maintains a reference to the interface of a component that can interpret and execute SOAP method calls. These calls are requests to execute a method on an invokable interface that is registered in the application. You can register an invokable interface by calling the RegisterInterface method of the TInvokableClassRegistry object that the global InvRegistry function returns.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

IHTTPSoapDispatch (see page 1200)

InvRegistry

TInvokableClassRegistry

17.1.1.1 THTTPSoapDispatchNode Members

The following tables list the members exposed by THTTPSoapDispatchNode.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1197)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	Notification (see page 1198)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
	DispatchSOAP (see page 1197)	Forwards the SOAP message on to the invoker for interpretation and execution.

Published Properties

	Name	Description
	Dispatcher (see page 1199)	Specifies the interface of the component that interprets SOAP messages.

17.1.1.2 THTTPSoapDispatchNode Fields

The fields of the THTTPSoapDispatchNode class are listed here.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1197)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

17.1.1.2.1 THTTPSoapDispatchNode.FSoapDispatcher Field

Pascal

```
FSoapDispatcher: IHTTPSoapDispatch;
```

C++

```
IHTTPSoapDispatch FSoapDispatcher;
```

Description

This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

17.1.1.3 THTTPSoapDispatchNode Methods

The methods of the THTTPSoapDispatchNode class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1198)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
	DispatchSOAP (see page 1197)	Forwards the SOAP message on to the invoker for interpretation and execution.

17.1.1.3.1 THTTPSoapDispatchNode.DispatchSOAP Method

Forwards the SOAP message on to the invoker for interpretation and execution.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response);
```

Description

DispatchSOAP calls the DispatchSOAP method of the interface specified by the Dispatcher (see page 1199) property. The invoker that implements that interface interprets the message, executes it, and returns the results as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

See Also

DispatchSOAP (see page 1201)

Dispatcher (see page 1199)

17.1.1.3.2 THTTPSoapDispatchNode.Notification Method

Responds when objects are added or removed from the Web module.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. When Operation is opRemove, Notification checks whether AComponent is the implementer of the Dispatcher (see page 1199) property. If so, it sets Dispatcher (see page 1199) to nil (Delphi) or NULL (C++).

See Also

Dispatcher (see page 1199)

17.1.1.4 THTTPSoapDispatchNode Properties

The properties of the THTTPSoapDispatchNode class are listed here.

Published Properties

	Name	Description
	Dispatcher (see page 1199)	Specifies the interface of the component that interprets SOAP messages.

17.1.1.4.1 THTTPSoapDispatchNode.Dispatcher Property

Specifies the interface of the component that interprets SOAP messages.

Pascal

```
property Dispatcher: IHTTPSoapDispatch;
```

C++

```
__property IHTTPSoapDispatch Dispatcher;
```

Description

Set Dispatcher to the interface of a component that interprets SOAP messages and passes them on to the target invokable interface. Dispatcher is the interface of an invoker component, such as THTTPSoapPascalInvoker (Delphi) or THTTPSoapCppInvoker (C++). Invoker components interpret incoming SOAP calls, identify the invokable interface to which they are directed, execute the call, and assemble a response message.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

17.2 Interfaces

The following table lists interfaces in this documentation.

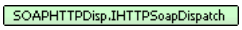
Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 1200)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

17.2.1 IHTTPSoapDispatch Interface

IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

Class Hierarchy



Pascal

```
IHTTPSoapDispatch = interface;
```

C++

```
__interface IHTTPSoapDispatch;
```

File

SOAPHTTPDisp

Description

The THTTPSoapDispatcher component uses this interface to call the object that interprets and responds to the incoming SOAP messages that it receives. By using an interface, THTTPSoapDispatcher can work with any component that interprets and executes SOAP messages.

This interface is implemented by THTTPSoapPascalInvoker and THTTPSoapCppInvoker.

Note: C++ method declarations that use IHTTPSoapDispatch use the _di_IHTTPSoapDispatch type instead. This type is a DelphiInterface (see page 3263) wrapper around the IHTTPSoapDispatch interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IHTTPSoapDispatch >
_di_IHTTPSoapDispatch;
```

See Also

THTTPSoapPascalInvoker (see page 1205)

THTTPSoapDispatchNode (see page 1196)

17.2.1.1 IHTTPSoapDispatch Members

The following tables list the members exposed by IHTTPSoapDispatch.

Public Methods

	Name	Description
	DispatchSOAP (see page 1201)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

17.2.1.2 IHTTPSoapDispatch Methods

The methods of the IHTTPSoapDispatch class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 1201)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

17.2.1.2.1 IHTTPSoapDispatch.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType);
```

C++

```
__fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request, TStream
Response, TWebServiceBindingType BindingType);
```

Description

DispatchSOAP interprets a SOAP message, executes the interface call it represents, and encodes any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 5920)

18 SOAPHTTPPasInv Namespace

This is namespace SOAPHTTPPasInv.

Topics

Name	Description
Classes (see page 1204)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapCppInvoker (see page 1204)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 1205)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

18.1 Classes

The following table lists classes in this documentation.

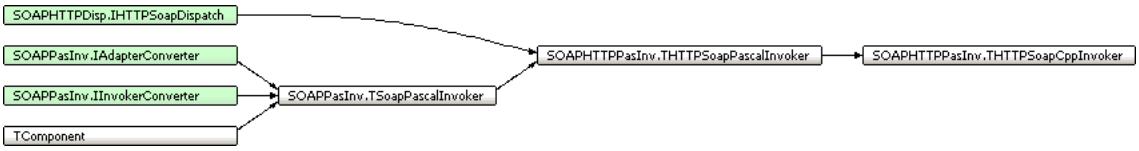
Classes

	Name	Description
	THTTPSoapCppInvoker (see page 1204)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 1205)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

18.1.1 THTTPSoapCppInvoker Class

THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy



Pascal

```
THTTPSoapCppInvoker = class(THTTPSoapPascalInvoker);
```

C++

```
class THTTPSoapCppInvoker : public THTTPSoapPascalInvoker;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapCppInvoker in a C++ application that publishes a Web Service using SOAP. THTTPSoapCppInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

THTTPSoapCppInvoker implements the IHTTPSoapDispatch (see page 1200) interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapCppInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 3505)

InvRegistry

TInvokableClassRegistry

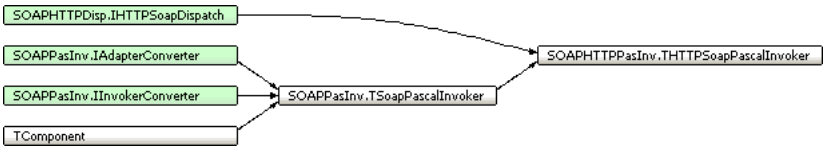
THTTPSoapDispatcher (see page 5920)

IHTTPSoapDispatch (see page 1200)

18.1.2 THTTPSoapPascalInvoker Class

THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy



Pascal

```
THTTPSoapPascalInvoker = class(TSoapPascalInvoker, IHTTPSoapDispatch);
```

C++

```
class THTTPSoapPascalInvoker : public TSoapPascalInvoker, public IHTTPSoapDispatch;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapPascalInvoker in an application that publishes a Web Service using SOAP. THTTPSoapPascalInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

In Delphi, you use THTTPSoapPascalInvoker directly as the invoker in your Web Service application. In C++, you use THTTPSoapCppInvoker (see page 1204), a descendant class, instead.

THTTPSoapPascalInvoker implements the IHTTPSoapDispatch (see page 1200) interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapPascalInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 3505)

InvRegistry

TInvokableClassRegistry

THTTPSoapDispatcher (see page 5920)

IHTTPSoapDispatch (see page 1200)

Understanding Invokable Interfaces

Using the SOAP Application Wizard

Writing Servers that Support Web Services

18.1.2.1 THTTPSoapPascalInvoker Members

The following tables list the members exposed by THTTPSoapPascalInvoker.

Protected Fields

	Name	Description
	FConverter (see page 1286)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1286)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1287)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1287)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1287)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	GetDomConverter (see page 1288)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1288)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 1289)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	Create (see page 1285)	Creates a new TSoapPascalInvoker (see page 1284) object.
V	Destroy (see page 1286)	Frees the memory associated with the TSoapPascalInvoker (see page 1284) object.
V	DispatchSOAP (see page 1207)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.
	GetDomConverter (see page 1294)	This is GetDomConverter, a member of class IAdapterConverter.
	GetInvoker (see page 1295)	This is GetInvoker, a member of class IInvokerConverter.
V	Invoke (see page 1288)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1290)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1291)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 1289)	Specifies the interface of the component that handles the marshaling of SOAP messages.

18.1.2.2 THTTPSoapPascalInvoker Methods

The methods of the THTTPSoapPascalInvoker class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 1207)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

18.1.2.2.1 THTTPSoapPascalInvoker.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response, TWebServiceBindingType BindingType);
```

Description

The SOAP dispatcher component (THTTPSoapDispatcher) calls DispatchSOAP when it receives an incoming SOAP message. DispatchSOAP interprets the message, looks up the target interface in the invocation registry, and calls the Invoke method to execute the call and encode any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the SOAP encoding of any return values. THTTPSoapPascalInvoker (see page 1205) uses this as the content for the outgoing response message.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 5920)

IHTTPSoapDispatch (see page 1200)

InvRegistry

Invoke (see page 1288)

19 SOAPHTTPTrans Namespace

This is namespace SOAPHTTPTrans.

Topics

Name	Description
Classes (see page 1210)	The following table lists classes in this documentation.
Interfaces (see page 1234)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1236)	The following table lists structs, records, enums in this documentation.
Types (see page 1237)	The following table lists types in this documentation.

Classes

	Name	Description
	ESOAPHTTPException (see page 1210)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 1214)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Interfaces

	Name	Description
	IHTTPReqResp (see page 1234)	This is class SOAPHTTPTrans.IHTTPReqResp.

Structs, Records, Enums

	Name	Description
	SOAPInvokeOptions (see page 1236)	This is record SOAPHTTPTrans.SOAPInvokeOptions.

Types

Name	Description
TBeforePostEvent (see page 1237)	This is type SOAPHTTPTrans.TBeforePostEvent.
TPostingDataEvent (see page 1237)	This is type SOAPHTTPTrans.TPostingDataEvent.
TReceivingDataEvent (see page 1238)	This is type SOAPHTTPTrans.TReceivingDataEvent.
TSOAPInvokeOptions (see page 1238)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.
TWinInetErrorEvent (see page 1238)	This is type SOAPHTTPTrans.TWinInetErrorEvent.

19.1 Classes

The following table lists classes in this documentation.

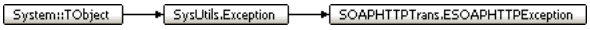
Classes

	Name	Description
	ESOAPHTTPException (see page 1210)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 1214)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

19.1.1 ESOAPHTTPException Class

ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.

Class Hierarchy



Pascal

```
ESOAPHTTPException = class(Exception);
```

C++

```
class ESOAPHTTPException : public Exception;
```

File

SOAPHTTPTrans

Description

ESOAPHTTPException is raised when there is a problem with the HTTP messages that communicate a SOAP request or response in a Web Services application. Such problems include

The application can't send or receive the HTTP message.

The URL that describes the server is missing or badly formed.

The Content-type header of an HTTP message is not consistent with SOAP.

See Also

THTTPReqResp (see page 1214)

19.1.1.1 ESOAPHTTPException Members

The following tables list the members exposed by ESOAPHTTPException.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.

	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1212)	Instantiates an instance of an ESOAPHTTPException (see page 1210) with a simple message string.
	CreateFmt (see page 1212)	Instantiates an instance of an ESOAPHTTPException (see page 1210) with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3391)	Returns a string containing the class name.
V	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.
	StatusCode (see page 1213)	Indicates the HTTP status code of the message that gave rise to this exception.

19.1.1.2 ESOAPHTTPException.Create Constructor

Instantiates an instance of an ESOAPHTTPException (see page 1210) with a simple message string.

Pascal

```
constructor Create(const Msg: string; SCode: Integer = 0; Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const UnicodeString Msg, int SCode = 0, int Dummy = 0);
```

Description

Call Create to construct an ESOAPHTTPException (see page 1210) object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 1213) property.

See Also

CreateFmt (see page 1212)

StatusCode (see page 1213)

Message (see page 4031)

19.1.1.3 ESOAPHTTPException.CreateFmt Constructor

Instantiates an instance of an ESOAPHTTPException (see page 1210) with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const; SCode: Integer = 0;
Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const UnicodeString Msg, const array of const Args, int SCode =
```

```
0, int Dummy = 0);
```

Description

Call CreateFmt to construct an ESOAPHTTPException (see page 1210) with a formatted message string.

Msg is a string constant containing format specifiers to be replaced with values from Args.

Args is an array of constants containing values to format according to format specifiers in Msg. CreateFmt calls the Format (see page 4232) function to transform Msg with the values in Args.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 1213) property.

See Also

Create (see page 1212)

StatusCode (see page 1213)

Message (see page 4031)

19.1.1.4 ESOAPHTTPException Properties

The properties of the ESOAPHTTPException class are listed here.

Public Properties

	Name	Description
	StatusCode (see page 1213)	Indicates the HTTP status code of the message that gave rise to this exception.

19.1.1.4.1 ESOAPHTTPException.StatusCode Property

Indicates the HTTP status code of the message that gave rise to this exception.

Pascal

```
property StatusCode: Integer;
```

C++

```
__property int StatusCode;
```

Description

StatusCode indicates the HTTP status code of the message that caused the application to raise an ESOAPHTTPException (see page 1210) exception. If the exception does not represent an HTTP response with a status code that indicates an error, StatusCode is 0.

The value of StatusCode is set by the ESOAPHTTPException (see page 1210) constructor.

Note: In Delphi code, this is set by the CreateFmt (see page 1212) or Create (see page 1212) constructor.

Note: In C++ code, this is set using one of the two constructor syntaxes that take a status code parameter.

See Also

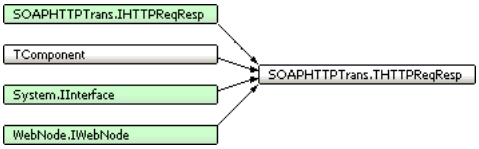
Create (see page 1212)

CreateFmt (see page 1212)

19.1.2 THTTPReqResp Class

THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Class Hierarchy



Pascal

```
THTTPReqResp = class(TComponent, IInterface, IWebNode, IHTTPReqResp);
```

C++

```
class THTTPReqResp : public TComponent, public IInterface, public IWebNode, public
IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

THTTPReqResp handles HTTP-based communication with a Web Service provider on behalf of a THTTPRio instance. THTTPRio uses this component to establish a connection to the Web Service provider and perform two tasks:

It can issue a "Get (see page 1222)" request to obtain information from a WSDL document.

It can issue a "Post" request to pass a method call to the server for execution and retrieve the results.

The Web Service that is the target of these HTTP messages can be specified directly, using the SoapAction (see page 1229) and URL (see page 1230) properties, or it can be taken from the SoapAction (see page 1229) in a WSDL document, as specified by the WSDLView (see page 1231) property. Additional properties provide details that are included in the HTTP request message header.

Note: On Windows, THTTPReqResp uses WinInet to establish a connection to the server. Wininet.dll must be installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed. WinInet has the advantage that it provides support for secure connections (https). To use WinInet, compile your project without the USE_INDY symbol defined.

Note: On Linux, THTTPReqResp uses an internal instance of the Indy HTTP component (TIdHTTP) to communicate with the server. The Indy component has the advantage that it can be used from an application that acts as both client and server, whereas WinInet only works for pure client code. To use an internal TIdHTTP, compile your project with the USE_INDY symbol defined.

See Also

IWebNode (see page 6582)

THTTPRio

TRIO

19.1.2.1 THTTPReqResp Members

The following tables list the members exposed by THTTPReqResp.

Protected Methods

	Name	Description
	_AddRef (see page 1217)	Called when an application uses a component interface.
	_Release (see page 1218)	Called when an application releases a component interface.
	GetMimeBoundary (see page 1222)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 1225)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	AfterConstruction (see page 1218)	Responds after the last constructor executes.
	BeforeExecute (see page 1219)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 1219)	Raises an exception when an error is detected.
	CheckContentType (see page 1220)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	Connect (see page 1220)	Establishes or terminates a connection to the Web Service application.
	Create (see page 1216)	Creates a new THTTPReqResp (see page 1214) object.
	Destroy (see page 1216)	Frees the memory associated with the THTTPReqResp (see page 1214) object.
	Execute (see page 1221)	This is the overview for the Execute method overload.
	Execute (see page 6584)	This is the overview for the Execute method overload.
	Get (see page 1222)	Sends a "Get" request to the server and retrieves the results.
	GetHTTPReqResp (see page 1222)	Returns a reference to this component.
	HandleWinInetError (see page 1223)	This is HandleWinInetError, a member of class THTTPReqResp.
	NewInstance (see page 1223)	Allocates memory for each instance of THTTPReqResp (see page 1214).
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Receive (see page 1223)	Receives the contents of an HTTP response to the message sent by the Send (see page 1224) message.
	Send (see page 1224)	Sends the encoded method call as an HTTP-based SOAP message.
	SendGet (see page 1224)	SendGets sends a "GET" request to the server.

Public Properties

	Name	Description
	ConnectTimeout (see page 1226)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	MaxSinglePostSize (see page 1227)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	MimeBoundary (see page 6585)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
	ReceiveTimeout (see page 1228)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 1229)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 1229)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 1230)	Specifies the URL that identifies the target interface on the target Web Service application.

Published Events

	Name	Description
	OnBeforePost (see page 1232)	Occurs before THTTPReqResp (see page 1214) sends its message.
	OnPostingData (see page 1232)	Occurs before THTTPReqResp (see page 1214) sends each chunk of data when the content is large.

	OnReceivingData (see page 1233)	Occurs after THTTPReqResp (see page 1214) receives each chunk of data when the content is large.
	OnWinInetError (see page 1233)	This is OnWinInetError, a member of class THTTPReqResp.

Published Properties

	Name	Description
	Agent (see page 1226)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.
	InvokeOptions (see page 1226)	Customizes the way THTTPReqResp (see page 1214) generates or handles headers in the messages it sends to the server.
	Password (see page 1227)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 1227)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1228)	Lists the host names of locally registered servers that do not require a proxy.
	UDDIBindingKey (see page 1229)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 1230)	Indicates the URL (see page 1230) of the UDDI registry to use for fail-over support.
	UserName (see page 1231)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 1231)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 1231)	Indicates the target interface method using a WSDL document.

19.1.2.2 THTTPReqResp.Create Constructor

Creates a new THTTPReqResp (see page 1214) object.

Pascal

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

C++

```
virtual __fastcall THTTPReqResp(TComponent * Owner);
```

Description

Most applications do not create instances of THTTPReqResp (see page 1214). It is created and used by the THTTPRio class.

The Owner parameter specifies the component that is responsible for freeing the THTTPReqResp (see page 1214) instance.

See Also

Owner

THTTPRio

19.1.2.3 THTTPReqResp.Destroy Destructor

Frees the memory associated with the THTTPReqResp (see page 1214) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPReqResp();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

The THTTPReqResp (see page 1214) destructor ensures that the connection to the server is closed before the component is

freed.

See Also

System::TObject::Free (see page 3382)

19.1.2.4 THTTPReqResp Methods

The methods of the THTTPReqResp class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 1217)	Called when an application uses a component interface.
	_Release (see page 1218)	Called when an application releases a component interface.
	GetMimeBoundary (see page 1222)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 1225)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	AfterConstruction (see page 1218)	Responds after the last constructor executes.
	BeforeExecute (see page 1219)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 1219)	Raises an exception when an error is detected.
	CheckContentType (see page 1220)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	Connect (see page 1220)	Establishes or terminates a connection to the Web Service application.
	Execute (see page 1221)	This is the overview for the Execute method overload.
	Get (see page 1222)	Sends a "Get" request to the server and retrieves the results.
	GetHTTPReqResp (see page 1222)	Returns a reference to this component.
	HandleWinInetError (see page 1223)	This is HandleWinInetError, a member of class THTTPReqResp.
	NewInstance (see page 1223)	Allocates memory for each instance of THTTPReqResp (see page 1214).
	Receive (see page 1223)	Receives the contents of an HTTP response to the message sent by the Send (see page 1224) message.
	Send (see page 1224)	Sends the encoded method call as an HTTP-based SOAP message.
	SendGet (see page 1224)	SendGets sends a "GET" request to the server.

19.1.2.4.1 THTTPReqResp._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke

QueryInterface

_Release

ComObject

_AddRef (see page 3504)

19.1.2.4.2 THTTPReqResp._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke

QueryInterface

_AddRef

ComObject

19.1.2.4.3 THTTPReqResp.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of THTTPReqResp (see page 1214) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

System::TObject::BeforeDestruction (see page 3368)

NewInstance (see page 1223)

19.1.2.4.4 THTTPReqResp.BeforeExecute Method

Performs transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode (see page 6582) method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 1221) method to send a request to the server. BeforeExecute looks up the information needed to call the server from the invocation registry, based on either the URL (see page 1230) that was supplied to the remote interfaced object or on the WSDLLocation, Service, and Port that were supplied to the remote interfaced object. If the method call requires an attachment, BeforeExecute looks up the MIME boundary for the attachment and adds headers to indicate the MIME boundary and content type of the attachment.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 1221)

OnBeforeExecute

BeforeExecute (see page 6583)

URL (see page 1192)

WSDLLocation (see page 1193)

Service (see page 1191)

Port (see page 1191)

19.1.2.4.5 THTTPReqResp.Check Method

Raises an exception when an error is detected.

Pascal

```
procedure Check(Error: Boolean; ShowSOAPAction: Boolean = False);
```

C++

```
__fastcall Check(Boolean Error, Boolean ShowSOAPAction = False);
```

Description

Note: On Windows (when using WinInet), THTTPReqResp (see page 1214) calls Check internally to raise an exception when it encounters an error.

Note: Check is not available on Linux.

Error (see page 3662) indicates whether an error condition was detected.

ShowSOAPAction indicates whether the error message should include information about the SOAP Action header.

See Also

Exception (see page 4015)

19.1.2.4.6 THTTPReqResp.CheckContentType Method

Raises an exception if the response to a SOAP method call has a common invalid content-type header.

Pascal

```
procedure CheckContentType;
```

C++

```
__fastcall CheckContentType();
```

Description

CheckContentType checks the Content-type header of the response that THTTPReqResp (see page 1214) receives after sending a request to the Web Service application. If it recognizes the content-type as invalid, CheckContentType raises an ESOAPHTTPException (see page 1210).

CheckContentType does not rule out all invalid content types, only the most common.

See Also

ESOAPHTTPException (see page 1210)

19.1.2.4.7 THTTPReqResp.Connect Method

Establishes or terminates a connection to the Web Service application.

Pascal

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

C++

```
__fastcall Connect(Boolean Value);
```

Description

Note: On Windows, call Connect to open or close a connection to the Web Service application identified by the URL (see page 1230) property. Connect opens a connection using WinInet.

Note: Connect is not available on Linux.

Value indicates whether to open or close the connection. When Value is true, Connect establishes a connection. When Value is false, Connect terminates the connection.

See Also

URL (see page 1230)

19.1.2.4.8 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	THTTPReqResp.Execute (String, TStream) (see page 1221)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream) (see page 1221)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream, TStream) (see page 1221)	Executes a SOAP method call and returns the encoded results.

19.1.2.4.8.1 THTTPReqResp.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; virtual; overload;
```

C++

```
virtual __fastcall Execute(const UnicodeString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
virtual __fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode (see page 6582) method of the same name. THTTPRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Connects to the Web Service application identified by the URL (see page 1230) property.
2. Creates an HTTP message with DataMsg or Request as its content and SoapAction (see page 1229) as its SoapAction (see page 1229) header.
3. Sends it to the Web Service application and retrieves the results.
4. Writes the results to the stream specified by Resp or returns a stream from which you can read the results. If using the third syntax, which returns a stream, the caller is responsible for freeing the stream object.

See Also

Execute (see page 6584)

THTTPRio

URL (see page 1230)

Send (see page 1224)

Receive (see page 1223)

SoapAction (see page 1229)

19.1.2.4.9 THTTPReqResp.Get Method

Sends a "Get" request to the server and retrieves the results.

Pascal

```
procedure Get(Resp: TStream); virtual;
```

C++

```
virtual __fastcall Get(TStream Resp);
```

Description

Get sends an HTTP request message to the server with a method type of GET. THTTPRio uses this method to fetch the WSDL document that underlies the WSDLView (see page 1231) property.

Resp is a stream that receives the contents of the response to the "Get" request message.

See Also

THTTPRio

URL (see page 1230)

Send (see page 1224)

Receive (see page 1223)

Execute (see page 1221)

WSDLView (see page 1231)

19.1.2.4.10 THTTPReqResp.GetHTTPReqResp Method

Returns a reference to this component.

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

GetHTTPReqResp returns this THTTPReqResp (see page 1214) instance (Delphi) or a pointer to this instance (C++).

19.1.2.4.11 THTTPReqResp.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 6585)

19.1.2.4.12 THTTPReqResp.HandleWinInetError Method

Pascal

```
function HandleWinInetError(LastError: DWord; Request: HINTERNET): DWord;
```

C++

```
__fastcall DWord HandleWinInetError(DWord LastError, HINTERNET Request);
```

Description

This is HandleWinInetError, a member of class THTTPReqResp.

19.1.2.4.13 THTTPReqResp.NewInstance Method

Allocates memory for each instance of THTTPReqResp (see page 1214).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

THTTPReqResp (see page 1214) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1218) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3391)

System::TObject::InstanceSize (see page 3389)

AfterConstruction (see page 1218)

19.1.2.4.14 THTTPReqResp.Receive Method

Receives the contents of an HTTP response to the message sent by the Send (see page 1224) message.

Pascal

```
procedure Receive(Context: Integer; Resp: TStream; IsGet: Boolean = False); virtual;
```

C++

```
virtual __fastcall Receive(int Context, TStream Resp, Boolean IsGet = False);
```

Description

Note: On Windows (when using WinInet), Receive fetches the HTTP response to the message sent by the SendMessage

method, and writes its contents to the specified stream.

Note: Receive is not available on Linux.

Context is the context ID that identifies the original request message that the Send (see page 1224) message sent. It is the return value from a previous call to the Send (see page 1224) method.

Resp is a stream to which Receive can write the contents of the response message.

IsGet indicates whether the response stream contains the results from a "Get (see page 1222)" request message.

See Also

Send (see page 1224)

Execute (see page 1221)

19.1.2.4.15 THTTPReqResp.Send Method

Sends the encoded method call as an HTTP-based SOAP message.

Pascal

```
function Send(const ASrc: TStream): Integer; virtual;
```

C++

```
virtual __fastcall int Send(const TStream ASrc);
```

Description

Note: On Windows (when using WinInet), Send sends an HTTP request message with the SoapAction (see page 1229) header specified by SoapAction (see page 1229) to the URL (see page 1230) specified by URL (see page 1230).

Note: Send is not available on Linux.

ASrc is a stream from which to read the contents of a SOAP message for the method call.

Send returns a context ID that identifies this request message. This context ID is passed to the Receive (see page 1223) method to retrieve the corresponding response message from the server.

See Also

Receive (see page 1223)

Execute (see page 1221)

SoapAction (see page 1229)

URL (see page 1230)

19.1.2.4.16 THTTPReqResp.SendGet Method

SendGets sends a "GET" request to the server.

Pascal

```
function SendGet: Integer; virtual;
```

C++

```
virtual __fastcall int SendGet();
```

Description

Note: On Windows (when using WinInet), the Get (see page 1222) method calls SendGet to send the GET request to the URL (see page 1230) specified by URL (see page 1230).

Note: SendGet is not available on Linux.

SendGet returns a context ID that identifies the GET request message. The Get (see page 1222) method uses this context ID to retrieve the corresponding response message from the server.

See Also

Get (see page 1222)

URL (see page 1230)

19.1.2.4.17 THTTPReqResp.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

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

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 6585)

19.1.2.5 THTTPReqResp Properties

The properties of the THTTPReqResp class are listed here.

Public Properties

	Name	Description
	ConnectTimeout (see page 1226)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	MaxSinglePostSize (see page 1227)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	ReceiveTimeout (see page 1228)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 1229)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 1229)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 1230)	Specifies the URL that identifies the target interface on the target Web Service application.

Published Properties

	Name	Description
	Agent (see page 1226)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.

	InvokeOptions (see page 1226)	Customizes the way THTTPReqResp (see page 1214) generates or handles headers in the messages it sends to the server.
	Password (see page 1227)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 1227)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1228)	Lists the host names of locally registered servers that do not require a proxy.
	UDDIBindingKey (see page 1229)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 1230)	Indicates the URL (see page 1230) of the UDDI registry to use for fail-over support.
	UserName (see page 1231)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 1231)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 1231)	Indicates the target interface method using a WSDL document.

19.1.2.5.1 THTTPReqResp.Agent Property

Provides the value of the User-Agent header for HTTP messages to the Web Service provider.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply the SOAP request message with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the request.

See Also

Title

19.1.2.5.2 THTTPReqResp.ConnectTimeout Property

Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.

Pascal

```
property ConnectTimeout: Integer;
```

C++

```
__property int ConnectTimeout;
```

Description

Use ConnectTimeout to indicate the number of milliseconds that can elapse before THTTPReqResp (see page 1214) abandons the attempt to form a connection.

See Also

SendTimeout (see page 1229)

ReceiveTimeout (see page 1228)

19.1.2.5.3 THTTPReqResp.InvokeOptions Property

Customizes the way THTTPReqResp (see page 1214) generates or handles headers in the messages it sends to the server.

Pascal

```
property InvokeOptions: TSOAPInvokeOptions;
```

C++

```
__property TSOAPInvokeOptions InvokeOptions;
```

Description

Use InvokeOptions to customize the way THTTPReqResp (see page 1214) generates SOAP messages to the invokable interface on the server. InvokeOptions is a set, allowing you to specify that THTTPReqResp (see page 1214) include none of the options.

See Also

SoapAction (see page 1229)

19.1.2.5.4 THTTPReqResp.MaxSinglePostSize Property

Pascal

```
property MaxSinglePostSize: Integer;
```

C++

```
__property int MaxSinglePostSize;
```

Description

This is MaxSinglePostSize, a member of class THTTPReqResp.

19.1.2.5.5 THTTPReqResp.Password Property

Specifies the password that allows the client application to log on when the Web Service provider requires authentication.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When THTTPReqResp (see page 1214) establishes a connection, it supplies the values of the UserName (see page 1231) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 1231)

Connect (see page 1220)

19.1.2.5.6 THTTPReqResp.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the client application can't resolve the value of host portion of URL (see page 1230) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 1228)

URL (see page 1230)

19.1.2.5.7 THTTPReqResp.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the client application can make a direct connection. When the URL (see page 1230) property addresses a server in this list, THTTPReqResp (see page 1214) makes a direct connection even if the Proxy (see page 1227) property is specified.

Note: ProxyByPass is only used on Windows, when the connection is formed using WinInet. On Linux, where the connection is formed using Indy, this property has no effect.

Note: When using WinInet, if ProxyByPass is an empty string, THTTPReqResp (see page 1214) checks the registry for any locally known servers and bypasses the proxy server if URL (see page 1230) refers to a registered local server.

See Also

Proxy (see page 1227)

URL (see page 1230)

19.1.2.5.8 THTTPReqResp.ReceiveTimeout Property

Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.

Pascal

```
property ReceiveTimeout: Integer;
```

C++

```
__property int ReceiveTimeout;
```

Description

Use ReceiveTimeout to indicate the number of milliseconds that can elapse after sending a request before THTTPReqResp (

see page 1214) cancels a request on the assumption that there will be no response forthcoming.

See Also

SendTimeout (see page 1229)

ConnectTimeout (see page 1226)

19.1.2.5.9 THTTPReqResp.SendTimeout Property

Indicates the number of milliseconds that can elapse before timing out the sending of a request.

Pascal

```
property SendTimeout: Integer;
```

C++

```
__property int SendTimeout;
```

Description

Use SendTimeout to indicate the number of milliseconds that can elapse when sending a request before THTTPReqResp (see page 1214) cancels the request.

See Also

ReceiveTimeout (see page 1228)

ConnectTimeout (see page 1226)

19.1.2.5.10 THTTPReqResp.SoapAction Property

Specifies the SoapAction header to include in the HTTP message that executes the method call.

Pascal

```
property SoapAction: string;
```

C++

```
__property UnicodeString SoapAction;
```

Description

SoapAction specifies the SoapAction header that appears in the HTTP message that is sent by the Send (see page 1224) method.The value of this header is deduced from information in a WSDL document, as indicated by the WSDLView (see page 1231) property.

See Also

URL (see page 1230)

WSDLView (see page 1231)

Send (see page 1224)

RegisterInterface

19.1.2.5.11 THTTPReqResp.UDDIBindingKey Property

Indicates the binding key of a UDDI entry to use for fail-over support.

Pascal

```
property UDDIBindingKey: WideString;
```

C++

```
__property BSTR UDDIBindingKey;
```

Description

UDDIBindingKey identifies the binding key for the Web Service TModel the defines the target interface. This key is used to look up the Web Service definition in the UDDI registry that the UDDIOperator (see page 1230) property identifies. UDDIBindingKey and UDDIOperator (see page 1230) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIOperator (see page 1230)

19.1.2.5.12 THTTPReqResp.UDDIOperator Property

Indicates the URL (see page 1230) of the UDDI registry to use for fail-over support.

Pascal

```
property UDDIOperator: String;
```

C++

```
__property UnicodeString UDDIOperator;
```

Description

UDDIOperator identifies the UDDI registry that was used to import the Web Service definition for the target interface. THTTPReqResp (see page 1214) uses this registry to look up the Web Service definition that the UDDIBindingKey (see page 1229) property identifies. UDDIOperator and UDDIBindingKey (see page 1229) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIBindingKey (see page 1229)

19.1.2.5.13 THTTPReqResp.URL Property

Specifies the URL that identifies the target interface on the target Web Service application.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

URL specifies the URL to which to send the message containing the SOAP request.

When communicating with a Web Service based on an imported WSDL document, the value of URL is deduced from information in a WSDL document, as indicated by the WSDLView (see page 1231) property.

Note: In Delphi, you can construct this URL as follows: The path portion must match the target path name of the THTTPSoapDispatcher component on the Web Service application and the protocol portion must specify http or https.

See Also

WSDLView (see page 1231)

WebDispatch (see page 5924)

SoapAction (see page 1229)

Send (see page 1224)

19.1.2.5.14 THTTPReqResp.UserName Property

Specifies the user name that allows the client application to log on to the server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When THTTPReqResp (see page 1214) establishes a connection, it supplies the values of the UserName and Password (see page 1227) properties so that it can log on to the host or proxy. The value of Password (see page 1227) can be an empty string if neither the host nor proxy requires authentication.

See Also

Password (see page 1227)

Connect (see page 1220)

19.1.2.5.15 THTTPReqResp.UseUTF8InHeader Property

Specifies whether the Content-Type header should specify UTF-8 as a character set.

Pascal

```
property UseUTF8InHeader: Boolean;
```

C++

```
__property Boolean UseUTF8InHeader;
```

Description

Set UseUTF8InHeader to indicate how THTTPReqResp (see page 1214) should generate the Content-Type header of the HTTP request message it sends to the server. When UseUTF8InHeader is true, the Content-Type header indicates that the content of the request is encoded in XML using the UTF-8 character set. Otherwise, the Content-Type header simply indicates that the content is encoded in XML.

See Also

SoapAction (see page 1229)

19.1.2.5.16 THTTPReqResp.WSDLView Property

Indicates the target interface method using a WSDL document.

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

WSDLView is a helper object that represents the WSDL file used to define the interfaces and protocols in the target Web Service application, as well as properties that indicate which is the target method call. It is used to generate the URL (see page 1230) and SoapAction (see page 1229) properties when importing definitions from a WSDL document.

See Also

URL (see page 1230)

SoapAction (see page 1229)

19.1.2.6 THTTPReqResp Events

The events of the THTTPReqResp class are listed here.

Published Events

	Name	Description
	OnBeforePost (see page 1232)	Occurs before THTTPReqResp (see page 1214) sends its message.
	OnPostingData (see page 1232)	Occurs before THTTPReqResp (see page 1214) sends each chunk of data when the content is large.
	OnReceivingData (see page 1233)	Occurs after THTTPReqResp (see page 1214) receives each chunk of data when the content is large.
	OnWinInetError (see page 1233)	This is OnWinInetError, a member of class THTTPReqResp.

19.1.2.6.1 THTTPReqResp.OnBeforePost Event

Occurs before THTTPReqResp (see page 1214) sends its message.

Pascal

```
property OnBeforePost: TBeforePostEvent;
```

C++

```
__property TBeforePostEvent OnBeforePost;
```

Description

Write (see page 3789) an OnBeforePost event handler to make final changes to the HTTP request before it is sent. The Send (see page 1224) method generates OnBeforePost after setting up the content and headers of a request message but before sending it.

See Also

OnPostingData (see page 1232)

Send (see page 1224)

19.1.2.6.2 THTTPReqResp.OnPostingData Event

Occurs before THTTPReqResp (see page 1214) sends each chunk of data when the content is large.

Pascal

```
property OnPostingData: TPostingDataEvent;
```

C++

```
__property TPostingDataEvent OnPostingData;
```

Description

Write (see page 3789) an OnPostingData event handler to provide the end-user with feedback of the progress when sending a request that contains a large amount of data. When the content of a request is very large (for example, when it contains large attachments), the Send (see page 1224) method breaks the message into blocks, which it sends separately. After sending each of these blocks, it generates an OnPostingData event. You can use the event handler to update a progress bar or similar control to indicate how much of the message has been sent.

See Also

OnBeforePost (see page 1232)

Send (see page 1224)

OnReceivingData (see page 1233)

19.1.2.6.3 THTTPReqResp.OnReceivingData Event

Occurs after THTTPReqResp (see page 1214) receives each chunk of data when the content is large.

Pascal

```
property OnReceivingData: TReceivingDataEvent;
```

C++

```
__property TReceivingDataEvent OnReceivingData;
```

Description

Write (see page 3789) an OnReceivingData event handler to provide the end-user with feedback of the progress when receiving a request that contains a large amount of data. When the content of a response is very large (for example, when it contains large attachments), it is broken into blocks, which are sent separately. After receiving each of these blocks, the Receive (see page 1223) method generates an OnReceivingData event. You can use the event handler to update a progress bar or similar control to indicate how much of the response has been received.

See Also

OnPostingData (see page 1232)

Receive (see page 1223)

19.1.2.6.4 THTTPReqResp.OnWinInetError Event

Pascal

```
property OnWinInetError: TWinInetErrorEvent;
```

C++

```
__property TWinInetErrorEvent OnWinInetError;
```

Description

This is OnWinInetError, a member of class THTTPReqResp.

19.2 Interfaces

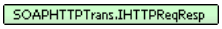
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHTTPReqResp (see page 1234)	This is class SOAPHTTPTrans.IHTTPReqResp.

19.2.1 IHTTPReqResp Interface

Class Hierarchy



Pascal

```
IHTTPReqResp = interface;
```

C++

```
__interface IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

This is class SOAPHTTPTrans.IHTTPReqResp.

19.2.1.1 IHTTPReqResp Members

The following tables list the members exposed by IHTTPReqResp.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 1235)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

19.2.1.2 IHTTPReqResp Methods

The methods of the IHTTPReqResp class are listed here.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 1235)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

19.2.1.2.1 IHTTPReqResp.GetHTTPReqResp Method

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

This is GetHTTPReqResp, a member of class IHTTPReqResp.

19.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPInvokeOptions (see page 1236)	This is record SOAPHTTPTrans.SOAPInvokeOptions.

19.3.1 SOAPHTTPTrans.SOAPInvokeOptions Enumeration

Pascal

```
SOAPInvokeOptions = (
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
);
```

C++

```
enum SOAPInvokeOptions {
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
};
```

File

SOAPHTTPTrans

Description

This is record SOAPHTTPTrans.SOAPInvokeOptions.

19.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePostEvent (see page 1237)	This is type SOAPHTTPTrans.TBeforePostEvent.
TPostingDataEvent (see page 1237)	This is type SOAPHTTPTrans.TPostingDataEvent.
TReceivingDataEvent (see page 1238)	This is type SOAPHTTPTrans.TReceivingDataEvent.
TSOAPInvokeOptions (see page 1238)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.
TWinInetErrorEvent (see page 1238)	This is type SOAPHTTPTrans.TWinInetErrorEvent.

19.4.1 SOAPHTTPTrans.TBeforePostEvent Type

Pascal

```
TBeforePostEvent = procedure (const HTTPReqResp: THTTPReqResp; Data: Pointer) of object;
```

C++

```
(const HTTPReqResp: THTTPReqResp; Data: Pointer) ( TBeforePostEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TBeforePostEvent.

19.4.2 SOAPHTTPTrans.TPostingDataEvent Type

Pascal

```
TPostingDataEvent = procedure (Sent: Integer; Total: Integer) of object;
```

C++

```
(Sent: Integer; Total: Integer) ( TPostingDataEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TPostingDataEvent.

19.4.3 SOAPHTTPTrans.TReceivingDataEvent Type

Pascal

```
TReceivingDataEvent = procedure (Read: Integer; Total: Integer) of object;
```

C++

```
(Read: Integer; Total: Integer) ( TReceivingDataEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TReceivingDataEvent.

19.4.4 SOAPHTTPTrans.TSOAPInvokeOptions Type

Pascal

```
TSOAPInvokeOptions = set of SOAPInvokeOptions;
```

C++

```
SOAPInvokeOptions TSOAPInvokeOptions;
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TSOAPInvokeOptions.

19.4.5 SOAPHTTPTrans.TWinInetErrorEvent Type

Pascal

```
TWinInetErrorEvent = function (LastError: DWord; Request: Pointer): DWord of object;
```

C++

```
(LastError: DWord; Request: Pointer): DWord ( TWinInetErrorEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TWinInetErrorEvent.

20 SOAPLinked Namespace

This is namespace SOAPLinked.

Topics

Name	Description
Classes (see page 1241)	The following table lists classes in this documentation.
Interfaces (see page 1268)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1272)	The following table lists structs, records, enums in this documentation.
Types (see page 1273)	The following table lists types in this documentation.

Classes

	Name	Description
	ECustomException (see page 1241)	This is class SOAPLinked.ECustomException.
	TFTCompT (see page 1247)	This is class SOAPLinked.TFTCompT.
	TFTCompTI (see page 1249)	This is class SOAPLinked.TFTCompTI.
	TLinkedRIO (see page 1252)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 1257)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 1262)	This is class SOAPLinked.TLogLinkedWebNode.
	TWinHandleUser (see page 1266)	This is class SOAPLinked.TWinHandleUser.

Interfaces

	Name	Description
	IIntegerTest (see page 1268)	This is class SOAPLinked.IIntegerTest.
	IObjectGetter (see page 1269)	This is class SOAPLinked.IObjectGetter.
	IStringTest (see page 1270)	This is class SOAPLinked.IStringTest.

Structs, Records, Enums

	Name	Description
	EventKind (see page 1272)	This is record SOAPLinked.EventKind.

Types

Name	Description
TCStringEvent (see page 1273)	This is type SOAPLinked.TCStringEvent.
TCustExCallback (see page 1273)	This is type SOAPLinked.TCustExCallback.
TStringEvent (see page 1274)	This is type SOAPLinked.TStringEvent.

TVStringEvent (see page 1274)	This is type SOAPLinked.TVStringEvent.

20.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECustomException (see page 1241)	This is class SOAPLinked.ECustomException.
	TFTCompT (see page 1247)	This is class SOAPLinked.TFTCompT.
	TFTCompTI (see page 1249)	This is class SOAPLinked.TFTCompTI.
	TLinkedRIO (see page 1252)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 1257)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 1262)	This is class SOAPLinked.TLogLinkedWebNode.
	TWinHandleUser (see page 1266)	This is class SOAPLinked.TWinHandleUser.

20.1.1 ECustomException Class

Class Hierarchy

Pascal

```
ECustomException = class(Exception);
```

C++

```
class ECustomException : public Exception;
```

File

SOAPLinked

Description

This is class SOAPLinked.ECustomException.

20.1.1.1 ECustomException Members

The following tables list the members exposed by ECustomException.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.

	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CatchException (see page 1246)	This is CatchException, a member of class ECustomException.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1243)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 1245)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceCount (see page 1246)	This is InstanceCount, a member of class ECustomException.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowExcept (see page 1246)	This is ThrowExcept, a member of class ECustomException.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

20.1.1.2 ECustomException.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const msg: string);
```

C++

```
__fastcall ECustomException(const UnicodeString msg);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
```

```
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  In addition to displaying the exception message, which
  happens by default, the following code shuts down the
  application when an exception is not caught and handled.
  AppException should be declared a method of TForm1.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnException := AppException;
  end;

  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    Application.ShowException(E);
    Application.Terminate;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    raise EPasswordInvalid.Create('Incorrect password entered');
  end;
```

See Also

CreateFmt (see page 4020)

CreateRes (see page 4022)

CreateResFmt (see page 4023)

Message (see page 4031)

20.1.1.3 ECustomException.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~ECustomException();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

20.1.1.4 ECustomException Methods

The methods of the ECustomException class are listed here.

Public Methods

	Name	Description
	CatchException (see page 1246)	This is CatchException, a member of class ECustomException.
	InstanceCount (see page 1246)	This is InstanceCount, a member of class ECustomException.
	ThrowExcept (see page 1246)	This is ThrowExcept, a member of class ECustomException.

20.1.1.4.1 ECustomException.CatchException Method

Pascal

```
class function CatchException(var ExceptTypeName: string; var Address: Pointer; TypedHandler:
boolean; Callback: TCustExCallback): boolean; static;
```

C++

```
static __fastcall boolean CatchException(UnicodeString ExceptTypeName, void * Address, boolean
TypedHandler, TCustExCallback Callback);
```

Description

This is CatchException, a member of class ECustomException.

20.1.1.4.2 ECustomException.InstanceCount Method

Pascal

```
class function InstanceCount: Integer; static;
```

C++

```
static __fastcall int InstanceCount();
```

Description

This is InstanceCount, a member of class ECustomException.

20.1.1.4.3 ECustomException.ThrowExcept Method

Pascal

```
class procedure ThrowExcept(const msg: string); static;
```

C++

```
static __fastcall ThrowExcept(const UnicodeString msg);
```

Description

This is ThrowExcept, a member of class ECustomException.

20.1.2 TFTCompT Class

Class Hierarchy

Pascal

```
TFTCompT = class(TComponent);
```

C++

```
class TFTCompT : public TComponent;
```

File

SOAPLinked

Description

This is class SOAPLinked.TFTCompT.

20.1.2.1 TFTCompT Members

The following tables list the members exposed by TFTCompT.

Public Methods

	Name	Description
	fireEvent (see page 1248)	This is fireEvent, a member of class TFTCompT.
	fireEventEx (see page 1248)	This is fireEventEx, a member of class TFTCompT.

Published Events

	Name	Description
	CStringEvent (see page 1248)	This is CStringEvent, a member of class TFTCompT.
	StringEvent (see page 1248)	This is StringEvent, a member of class TFTCompT.
	VStringEvent (see page 1249)	This is VStringEvent, a member of class TFTCompT.

20.1.2.2 TFTCompT Methods

The methods of the TFTCompT class are listed here.

Public Methods

	Name	Description
	fireEvent (see page 1248)	This is fireEvent, a member of class TFTCompT.
	fireEventEx (see page 1248)	This is fireEventEx, a member of class TFTCompT.

20.1.2.2.1 TFTCompT.fireEvent Method

Pascal

```
function fireEvent(S: string; ek: EventKind): string;
```

C++

```
__fastcall UnicodeString fireEvent(UnicodeString S, EventKind ek);
```

Description

This is fireEvent, a member of class TFTCompT.

20.1.2.2.2 TFTCompT.fireEventEx Method

Pascal

```
procedure fireEventEx(var S: string; ek: EventKind);
```

C++

```
__fastcall fireEventEx(UnicodeString S, EventKind ek);
```

Description

This is fireEventEx, a member of class TFTCompT.

20.1.2.3 TFTCompT Events

The events of the TFTCompT class are listed here.

Published Events

	Name	Description
	CStringEvent (see page 1248)	This is CStringEvent, a member of class TFTCompT.
	StringEvent (see page 1248)	This is StringEvent, a member of class TFTCompT.
	VStringEvent (see page 1249)	This is VStringEvent, a member of class TFTCompT.

20.1.2.3.1 TFTCompT.CStringEvent Event

Pascal

```
property CStringEvent: TCStringEvent;
```

C++

```
__property TCStringEvent CStringEvent;
```

Description

This is CStringEvent, a member of class TFTCompT.

20.1.2.3.2 TFTCompT.StringEvent Event

Pascal

```
property StringEvent: TStringEvent;
```

C++

```
__property TStringEvent StringEvent;
```

Description

This is StringEvent, a member of class TFTCompT.

20.1.2.3.3 TFTCompT.VStringEvent Event

Pascal

```
property VStringEvent: TVStringEvent;
```

C++

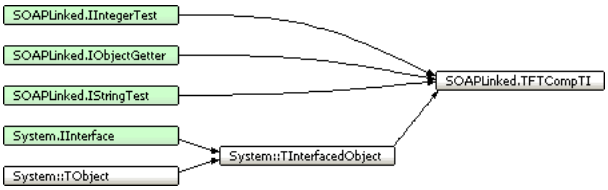
```
__property TVStringEvent VStringEvent;
```

Description

This is VStringEvent, a member of class TFTCompT.

20.1.3 TFTCompTI Class

Class Hierarchy



Pascal

```
TFTCompTI = class(TInterfacedObject, IStringTest, IIntegerTest, IObjectGetter);
```

C++

```
class TFTCompTI : public TInterfacedObject, public IStringTest, public IIntegerTest, public
IObjectGetter;
```

File

SOAPLinked

Description

This is class SOAPLinked.TFTCompTI.

20.1.3.1 TFTCompTI Members

The following tables list the members exposed by TFTCompTI.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.

	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	Add (see page 1251)	This is Add, a member of class TFTCompTI.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	Append (see page 1251)	This is Append, a member of class TFTCompTI.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Doubler (see page 1251)	This is Doubler, a member of class TFTCompTI.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAsObject (see page 1252)	This is GetAsObject, a member of class TFTCompTI.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	UpCase (see page 1252)	This is UpCase, a member of class TFTCompTI.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

20.1.3.2 TFTCompTI Methods

The methods of the TFTCompTI class are listed here.

Public Methods

	Name	Description
	Add (see page 1251)	This is Add, a member of class TFTCompTI.
	Append (see page 1251)	This is Append, a member of class TFTCompTI.
	Doubler (see page 1251)	This is Doubler, a member of class TFTCompTI.
	GetAsObject (see page 1252)	This is GetAsObject, a member of class TFTCompTI.
	UpCase (see page 1252)	This is UpCase, a member of class TFTCompTI.

20.1.3.2.1 TFTCompTI.Add Method

Pascal

```
function Add(const I1: Integer; const I2: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Add(const int I1, const int I2);
```

Description

This is Add, a member of class TFTCompTI.

20.1.3.2.2 TFTCompTI.Append Method

Pascal

```
function Append(const Str1: string; const Str2: string): string; cdecl;
```

C++

```
__cdecl __fastcall UnicodeString Append(const UnicodeString Str1, const UnicodeString Str2);
```

Description

This is Append, a member of class TFTCompTI.

20.1.3.2.3 TFTCompTI.Doubler Method

Pascal

```
procedure Doubler(var I: Integer); cdecl;
```

C++

```
__cdecl __fastcall Doubler(int I);
```

Description

This is Doubler, a member of class TFTCompTI.

20.1.3.2.4 TFTCompTI.GetAsObject Method

Pascal

```
function GetAsObject: TObject;
```

C++

```
__fastcall TObject * GetAsObject();
```

Description

This is GetAsObject, a member of class TFTCompTI.

20.1.3.2.5 TFTCompTI.UpCase Method

Pascal

```
procedure UpCase(var Str1: string); stdcall;
```

C++

```
__fastcall __stdcall UpCase(UnicodeString Str1);
```

Description

This is UpCase, a member of class TFTCompTI.

20.1.4 TLinkedRIO Class

TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.

Class Hierarchy

Pascal

```
TLinkedRIO = class(TRIO);
```

C++

```
class TLinkedRIO : public TRIO;
```

File

SOAPLinked

Description

Use TLinkedRIO to generate calls to registered invokable interfaces within a Web Service application. Unlike other remote interfaced objects (TRIO descendants), TLinkedRIO provides no mechanism for connecting to a remote server. Rather, it passes calls directly to a TLinkedWebNode (see page 1257) object, which uses an internal invoker component to look up the target interface and forward calls to the implementation object.

TLinkedRIO is intended primarily for debugging Web Service applications during development. Instead of directly calling the implementation objects in the application, TLinkedRIO marshals the call into its SOAP representation and uses the invocation registry to locate the implementation object and forward the encoded request. If you use the CreateFile (see page 1254) constructor (Delphi) or supply a pair of file names to the constructor (C++), TLinkedRIO saves the encoded value of every method call and the encoded results so that you can track the "messages" that your application receives and sends.

Before TLinkedRIO can call a method table on an invokable interface, the interface must be registered with the invocation

registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry

TSoapPascalInvoker (see page 1284)

TLinkedWebNode (see page 1257)

20.1.4.1 TLinkedRIO Members

The following tables list the members exposed by TLinkedRIO.

Protected Methods

	Name	Description
	GetResponseStream (see page 1255)	Returns a newly created stream that receives the results of executing a method call.

Public Methods

	Name	Description
	Create (see page 1253)	Creates an instance of TLinkedRIO (see page 1252).
	CreateFile (see page 1254)	Creates an instance of TLinkedRIO (see page 1252).
	Destroy (see page 1254)	Destroys the instance of a TLinkedRIO (see page 1252) object.
	QueryInterface (see page 1255)	Implements the IInterface (see page 3503) QueryInterface method.

Public Properties

	Name	Description
	WebNode (see page 1256)	Handles the execution of method calls on the invokable interface.

20.1.4.2 TLinkedRIO.Create Constructor

Creates an instance of TLinkedRIO (see page 1252).

Pascal

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

C++

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

Description

Call Create to instantiate a TLinkedRIO (see page 1252) object at runtime.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 1252) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 1252) instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

Tip: To save the encoded method calls that TLinkedRIO (see page 1252) generates and the encoded results that it receives, use the CreateFile (see page 1254) constructor instead.

See Also

RefCount

Owner

CreateFile (see page 1254)

20.1.4.3 TLinkedRIO.CreateFile Constructor

Creates an instance of TLinkedRIO (see page 1252).

Pascal

```
constructor CreateFile(AOwner: TComponent; ReqFile: string; RespFile: string); overload;
```

C++

```
__fastcall TLinkedRIO(TComponent * AOwner, UnicodeString ReqFile, UnicodeString RespFile);
```

Description

Call Create (see page 1253) to instantiate a TLinkedRIO (see page 1252) object when you want to save all of the encoded requests that it generates and the encoded results it receives to a file.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 1252) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 1252) instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

ReqFile is a file to which TLinkedRIO (see page 1252) writes every encoded request before passing it on to WebNode (see page 1256) for execution. If ReqFile is an empty string, TLinkedRIO (see page 1252) automatically generates a file with the same name as the invoked method. This lets you generate separate log files for each method of the interface.

RespFile is a file to which TLinkedRIO (see page 1252) writes every encoded response it receives from WebNode (see page 1256) after executing a method call. If RespFile is an empty string, TLinkedRIO (see page 1252) automatically generates a file with the name of the method and the string _resp appended. This lets you generate separate log files for responses to each method of the interface.

See Also

RefCount

Owner

Create (see page 1253)

WebNode (see page 1256)

20.1.4.4 TLinkedRIO.Destroy Destructor

Destroys the instance of a TLinkedRIO (see page 1252) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedRIO();
```

Description

There is no need to explicitly free a TLinkedRIO (see page 1252) instance.

If the TLinkedRIO (see page 1252) object is created with an Owner, then that Owner handles the freeing of the TLinkedRIO (see page 1252) instance. If the TLinkedRIO (see page 1252) object is created without an Owner, then it is automatically freed

when the reference count on its interface drops to zero.

See Also

Create (see page 1253)

Owner

RefCount

20.1.4.5 TLinkedRIO Methods

The methods of the TLinkedRIO class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 1255)	Returns a newly created stream that receives the results of executing a method call.

Public Methods

	Name	Description
	QueryInterface (see page 1255)	Implements the IInterface (see page 3503) QueryInterface method.

20.1.4.5.1 TLinkedRIO.GetResponseStream Method

Returns a newly created stream that receives the results of executing a method call.

Pascal

```
function GetResponseStream(BindingType: TWebServiceBindingType): TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream(TWebServiceBindingType BindingType);
```

Description

TRIO calls GetResponseStream internally when it executes a method call. It passes the returned stream as the Response parameter of the Execute method on the interface that the WebNode property specifies. After this stream receives the encoded response of a method call, the RIO component decodes the response and frees the stream.

BindingType indicates the expected coding of the response that the stream will hold. GetResponseStream can use this information to determine what type of stream to create.

See Also

Execute (see page 6584)

WebNode

20.1.4.5.2 TLinkedRIO.QueryInterface Method

Implements the IInterface (see page 3503) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface (see page 3503) interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts TLinkedRIO (see page 1252) to an invokable interface using the as operator.

TLinkedRIO (see page 1252) can only return a single type of interface pointer. The first time an application calls QueryInterface, TLinkedRIO (see page 1252) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this TLinkedRIO (see page 1252) instance. If the TLinkedRIO (see page 1252) object has already been typecast to an interface, then IID must identify that interface. If the TLinkedRIO (see page 1252) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless TLinkedRIO (see page 1252) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK (see page 3927)) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE (see page 3912) otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface (see page 3263) wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry

RegisterInterface

RefCount

20.1.4.6 TLinkedRIO Properties

The properties of the TLinkedRIO class are listed here.

Public Properties

	Name	Description
R	WebNode (see page 1256)	Handles the execution of method calls on the invokable interface.

20.1.4.6.1 TLinkedRIO.WebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property WebNode: TLinkedWebNode;
```

C++

```
__property TLinkedWebNode WebNode;
```

Description

TLinkedRIO (see page 1252) uses WebNode internally to execute method calls. It is the implementation of the IWebNode (see page 6582) interface that all TRIO descendants use to send method calls to the Web Service application. In the case of TLinkedRIO (see page 1252), this object can only execute calls on invokable interfaces that are implemented in the same application.

See Also

WebNode

IWebNode (see page 6582)

20.1.5 TLinkedWebNode Class

TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.

Class Hierarchy

Pascal

```
TLinkedWebNode = class(TComponent, IWebNode);
```

C++

```
class TLinkedWebNode : public TComponent, public IWebNode;
```

File

SOAPLinked

Description

TLinkedWebNode handles the communication between a TLinkedRIO (see page 1252) component and the implementation classes of registered invokable interfaces in the same application. To this end, it implements the IWebNode (see page 6582) interface that all TRIO descendants use for executing encoded method calls.

TLinkedRIO (see page 1252) uses an internal invoker (TSoapPascalInvoker (see page 1284)) to look up the invokable interface in the invocation registry and forward encoded requests for execution.

See Also

IWebNode (see page 6582)

TLinkedRio

TLinkedRIO (see page 1252)

TSoapPascalInvoker (see page 1284)

20.1.5.1 TLinkedWebNode Members

The following tables list the members exposed by TLinkedWebNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1260)	Returns the value of the MimeBoundary (see page 1262) property.
	GetResponseStream (see page 1261)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1261)	Sets the value of the MimeBoundary (see page 1262) property.

Public Methods

	Name	Description
	BeforeExecute (see page 1259)	Caches the RTTI for the method that is about to be executed.
	Create (see page 1258)	Creates a new TLinkedWebNode (see page 1257) object.
	Destroy (see page 1258)	Frees the memory associated with the TLinkedWebNode (see page 1257) object.
	Execute (see page 1260)	This is the overview for the Execute method overload.
	Execute (see page 6584)	This is the overview for the Execute method overload.

Public Properties

	Name	Description
	Invoker (see page 1262)	Executes encoded method calls received from the associated TLinkedRIO (see page 1252) object.
	MimeBoundary (see page 1262)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

20.1.5.2 TLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode (see page 1257) object.

Pascal

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

C++

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

Description

Most applications do not create instances of TLinkedWebNode (see page 1257). It is created and used by the TLinkedRio class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode (see page 1257) instance.

See Also

Owner

TLinkedRio

20.1.5.3 TLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode (see page 1257) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3382)

20.1.5.4 TLinkedWebNode Methods

The methods of the TLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1260)	Returns the value of the MimeBoundary (see page 1262) property.
	GetResponseStream (see page 1261)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1261)	Sets the value of the MimeBoundary (see page 1262) property.

Public Methods

	Name	Description
	BeforeExecute (see page 1259)	Caches the RTTI for the method that is about to be executed.
	Execute (see page 1260)	This is the overview for the Execute method overload.

20.1.5.4.1 TLinkedWebNode.BeforeExecute Method

Caches the RTTI for the method that is about to be executed.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler); virtual;
```

C++

```
virtual __fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode (see page 6582) method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 1260) method to send a request to the Web Service application. BeforeExecute allows an IWebNode (see page 6582) implementation class to perform any transport-specific tasks prior to establishing a connection. Because TLinkedWebNode (see page 1257) only works with interfaces implemented in the same executable, there is no transport protocol at all, and no transport-specific tasks to perform. BeforeExecute merely caches the RTTI for the method that is about to be executed.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 1260)

OnBeforeExecute

BeforeExecute (see page 6583)

20.1.5.4.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TLinkedWebNode.Execute (String, TStream) (see page 1260)	Executes a SOAP method call and returns the encoded results.
	TLinkedWebNode.Execute (TStream) (see page 1260)	Executes a SOAP method call and returns the encoded results.
	TLinkedWebNode.Execute (TStream, TStream) (see page 1260)	Executes a SOAP method call and returns the encoded results.

20.1.5.4.2.1 TLinkedWebNode.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
virtual __fastcall Execute(const UnicodeString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode (see page 6582) method of the same name. TLinkedRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Uses the invocation registry to look up the implementation class for the invokable interface.
2. Uses its internal invoker (the Invoker (see page 1262) property) to unmarshal the request specified by Request or DataMsg and pass it to the implementation class for execution.
3. Returns the encoded results from the invoker in the stream specified by Response or, using the third syntax, in a stream that is the return value of the method. When using the third syntax, the caller is responsible for freeing the returned stream.

See Also

Execute (see page 6584)

TLinkedRio

Invoker (see page 1262)

20.1.5.4.3 TLinkedWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 1262) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read implementation of the MimeBoundary (see page 1262) property.

See Also

MimeBoundary (see page 1262)

20.1.5.4.4 TLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream

20.1.5.4.5 TLinkedWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 1262) property.

Pascal

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

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 1262) property.

Value is the new MIME boundary string.

See Also

MimeBoundary (see page 1262)

20.1.5.5 TLinkedWebNode Properties

The properties of the TLinkedWebNode class are listed here.

Public Properties

	Name	Description
	Invoker (see page 1262)	Executes encoded method calls received from the associated TLinkedRIO (see page 1252) object.

	MimeBoundary (see page 1262)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

20.1.5.5.1 TLinkedWebNode.Invoker Property

Executes encoded method calls received from the associated TLinkedRIO (see page 1252) object.

Pascal

```
property Invoker: TSoapPascalInvoker;
```

C++

```
__property TSoapPascalInvoker Invoker;
```

Description

Invoker is the internal invoker component that looks up method calls in the invocation registry, forwards encoded requests to the registered implementation class, and returns the encoded results for the associated TLinkedRIO (see page 1252) object.

See Also

TLinkedRIO (see page 1252)

20.1.5.5.2 TLinkedWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property UnicodeString MimeBoundary;
```

Description

When the encoded method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

MimeBoundary (see page 6585)

20.1.6 TLogLinkedWebNode Class

Class Hierarchy

Pascal

```
TLogLinkedWebNode = class(TLinkedWebNode);
```

C++

```
class TLogLinkedWebNode : public TLinkedWebNode;
```

File

SOAPLinked

Description

This is class SOAPLinked.TLogLinkedWebNode.

20.1.6.1 TLogLinkedWebNode Members

The following tables list the members exposed by TLogLinkedWebNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1260)	Returns the value of the MimeBoundary (see page 1262) property.
V	GetResponseStream (see page 1265)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1261)	Sets the value of the MimeBoundary (see page 1262) property.

Public Methods

	Name	Description
V	BeforeExecute (see page 1259)	Caches the RTTI for the method that is about to be executed.
	BeforeExecute (see page 6583)	Performs any transport-specific setup before executing an encoded method call.
V	Create (see page 1263)	Creates a new TLinkedWebNode object.
V	Destroy (see page 1264)	Frees the memory associated with the TLinkedWebNode object.
V	Execute (see page 1264)	This is Execute, a member of class TLogLinkedWebNode.
V	Execute (see page 1260)	This is the overview for the Execute method overload.
	Execute (see page 6584)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6585)	Returns the value of the MimeBoundary (see page 6585) property.
	SetMimeBoundary (see page 6585)	Sets the value of the MimeBoundary (see page 6585) property.

Public Properties

	Name	Description
R	Invoker (see page 1262)	Executes encoded method calls received from the associated TLinkedRIO (see page 1252) object.
	MimeBoundary (see page 1262)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
	MimeBoundary (see page 6585)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Published Properties

	Name	Description
	ReqFile (see page 1265)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 1265)	This is RespFile, a member of class TLogLinkedWebNode.

20.1.6.2 TLogLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode object.

Pascal

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

C++

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

Description

Most applications do not create instances of TLinkedWebNode. It is created and used by the TLinkedRio class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode instance.

See Also

Owner

TLinkedRio

20.1.6.3 TLogLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLogLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3382)

20.1.6.4 TLogLinkedWebNode Methods

The methods of the TLogLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 1265)	Returns a stream to which the application can write the response to the method call.

Public Methods

	Name	Description
	Execute (see page 1264)	This is Execute, a member of class TLogLinkedWebNode.

20.1.6.4.1 TLogLinkedWebNode.Execute Method

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); override;
```

C++

```
virtual __fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TLogLinkedWebNode.

20.1.6.4.2 TLogLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream

20.1.6.5 TLogLinkedWebNode Properties

The properties of the TLogLinkedWebNode class are listed here.

Published Properties

	Name	Description
	ReqFile (see page 1265)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 1265)	This is RespFile, a member of class TLogLinkedWebNode.

20.1.6.5.1 TLogLinkedWebNode.ReqFile Property

Pascal

```
property ReqFile: string;
```

C++

```
__property UnicodeString ReqFile;
```

Description

This is ReqFile, a member of class TLogLinkedWebNode.

20.1.6.5.2 TLogLinkedWebNode.RespFile Property

Pascal

```
property RespFile: string;
```

C++

```
__property UnicodeString RespFile;
```

Description

This is RespFile, a member of class TLogLinkedWebNode.

20.1.7 TWinHandleUser Class

Class Hierarchy

Pascal

```
TWinHandleUser = class(TObject);
```

C++

```
class TWinHandleUser : public TObject;
```

File

SOAPLinked

Description

This is class SOAPLinked.TWinHandleUser.

20.1.7.1 TWinHandleUser Members

The following tables list the members exposed by TWinHandleUser.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
V	UseHandle1 (see page 1267)	This is UseHandle1, a member of class TWinHandleUser.
V	UseHandle2 (see page 1267)	This is UseHandle2, a member of class TWinHandleUser.

20.1.7.2 TWinHandleUser Methods

The methods of the TWinHandleUser class are listed here.

Public Methods

	Name	Description
V	UseHandle1 (see page 1267)	This is UseHandle1, a member of class TWinHandleUser.
V	UseHandle2 (see page 1267)	This is UseHandle2, a member of class TWinHandleUser.

20.1.7.2.1 TWinHandleUser.UseHandle1 Method

Pascal

```
procedure UseHandle1(ahdesk: HDESK; aenhMf: HENHMETAFILE; amf: HMETAFILE; apalette: HPALETTE;
apen: HPEN; adc: HDC; abmp: HBITMAP; awnd: HWND; ahkl: HKL; aicon: HICON; afont: HFONT);
virtual;
```

C++

```
virtual __fastcall UseHandle1(HDESK ahdesk, HENHMETAFILE aenhMf, HMETAFILE amf, HPALETTE
apalette, HPEN apen, HDC adc, HBITMAP abmp, HWND awnd, HKL ahkl, HICON aicon, HFONT afont);
```

Description

This is UseHandle1, a member of class TWinHandleUser.

20.1.7.2.2 TWinHandleUser.UseHandle2 Method

Pascal

```
procedure UseHandle2(abrush: HBRUSH; amenu: HMENU; akey: HKEY; atask: HTASK; ainst: HINST;
ahook: HHOOK; agdiobj: HGDIOBJ; aAccel: HACCEL; aClrSpc: HCOLORSPACE; aglrc: HGLRC; acrsr:
HCURSOR; ahstr: HSTR; ahwinsta: HWINSTA); virtual;
```

C++

```
virtual __fastcall UseHandle2(HBRUSH abrush, HMENU amenu, HKEY akey, HTASK atask, HINST ainst,
HHOOK ahook, HGDIOBJ agdiobj, HACCEL aAccel, HCOLORSPACE aClrSpc, HGLRC aglrc, HCURSOR acrsr,
HSTR ahstr, HWINSTA ahwinsta);
```

Description

This is UseHandle2, a member of class TWinHandleUser.

20.2 Interfaces

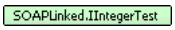
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IIntegerTest (see page 1268)	This is class SOAPLinked.IIntegerTest.
	IObjectGetter (see page 1269)	This is class SOAPLinked.IObjectGetter.
	IStringTest (see page 1270)	This is class SOAPLinked.IStringTest.

20.2.1 IIntegerTest Interface

Class Hierarchy



Pascal

```
IIntegerTest = interface;
```

C++

```
__interface IIntegerTest;
```

File

SOAPLinked

Description

This is class SOAPLinked.IIntegerTest.

20.2.1.1 IIntegerTest Members

The following tables list the members exposed by IIntegerTest.

Public Methods

	Name	Description
	Add (see page 1269)	This is Add, a member of class IIntegerTest.
	Doubler (see page 1269)	This is Doubler, a member of class IIntegerTest.

20.2.1.2 IIntegerTest Methods

The methods of the IIntegerTest class are listed here.

Public Methods

	Name	Description
	Add (see page 1269)	This is Add, a member of class IIntegerTest.
	Doubler (see page 1269)	This is Doubler, a member of class IIntegerTest.

20.2.1.2.1 IIntegerTest.Add Method

Pascal

```
function Add(const I1: Integer; const I2: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Add(const int I1, const int I2);
```

Description

This is Add, a member of class IIntegerTest.

20.2.1.2.2 IIntegerTest.Doubler Method

Pascal

```
procedure Doubler(var I: Integer); cdecl;
```

C++

```
__cdecl __fastcall Doubler(int I);
```

Description

This is Doubler, a member of class IIntegerTest.

20.2.2 IObjectGetter Interface

Class Hierarchy

SOAPLinked.IObjectGetter

Pascal

```
IObjectGetter = interface;
```

C++

```
__interface IObjectGetter;
```

File

SOAPLinked

Description

This is class SOAPLinked.IObjectGetter.

20.2.2.1 IObjectGetter Members

The following tables list the members exposed by IObjectGetter.

Public Methods

	Name	Description
	GetAsObject (see page 1270)	This is GetAsObject, a member of class IObjectGetter.

20.2.2.2 IObjectGetter Methods

The methods of the IObjectGetter class are listed here.

Public Methods

	Name	Description
	GetAsObject (see page 1270)	This is GetAsObject, a member of class IObjectGetter.

20.2.2.2.1 IObjectGetter.GetAsObject Method

Pascal

```
function GetAsObject: TObject;
```

C++

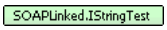
```
__fastcall TObject * GetAsObject();
```

Description

This is GetAsObject, a member of class IObjectGetter.

20.2.3 IStringTest Interface

Class Hierarchy



Pascal

```
IStringTest = interface;
```

C++

```
__interface IStringTest;
```

File

SOAPLinked

Description

This is class SOAPLinked.IStringTest.

20.2.3.1 IStringTest Members

The following tables list the members exposed by IStringTest.

Public Methods

	Name	Description
	Append (see page 1271)	This is Append, a member of class IStringTest.
	UpCase (see page 1271)	This is UpCase, a member of class IStringTest.

20.2.3.2 IStringTest Methods

The methods of the IStringTest class are listed here.

Public Methods

	Name	Description
	Append (see page 1271)	This is Append, a member of class IStringTest.
	UpCase (see page 1271)	This is UpCase, a member of class IStringTest.

20.2.3.2.1 IStringTest.Append Method

Pascal

```
function Append(const Str1: string; const Str2: string): string; cdecl;
```

C++

```
__cdecl __fastcall UnicodeString Append(const UnicodeString Str1, const UnicodeString Str2);
```

Description

This is Append, a member of class IStringTest.

20.2.3.2.2 IStringTest.UpCase Method

Pascal

```
procedure UpCase(var Str1: string); stdcall;
```

C++

```
__fastcall __stdcall UpCase(UnicodeString Str1);
```

Description

This is UpCase, a member of class IStringTest.

20.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	EventKind (see page 1272)	This is record SOAPLinked.EventKind.

20.3.1 SOAPLinked.EventKind Enumeration

Pascal

```
EventKind = (
  evNormal,
  evConst,
  evVar
);
```

C++

```
enum EventKind {
  evNormal,
  evConst,
  evVar
};
```

File

SOAPLinked

Description

This is record SOAPLinked.EventKind.

20.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCStringEvent (see page 1273)	This is type SOAPLinked.TCStringEvent.
TCustExCallback (see page 1273)	This is type SOAPLinked.TCustExCallback.
TStringEvent (see page 1274)	This is type SOAPLinked.TStringEvent.
TVStringEvent (see page 1274)	This is type SOAPLinked.TVStringEvent.

20.4.1 SOAPLinked.TCStringEvent Type

Pascal

```
TCStringEvent = procedure (Sender: TObject; const S: string) of object;
```

C++

```
(Sender: TObject; const S: string) ( TCStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TCStringEvent.

20.4.2 SOAPLinked.TCustExCallback Type

Pascal

```
TCustExCallback = procedure;
```

C++

```
( TCustExCallback)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TCustExCallback.

20.4.3 SOAPLinked.TStringEvent Type

Pascal

```
TStringEvent = procedure (Sender: TObject; S: string) of object;
```

C++

```
(Sender: TObject; S: string) ( TStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TStringEvent.

20.4.4 SOAPLinked.TVStringEvent Type

Pascal

```
TVStringEvent = procedure (Sender: TObject; var S: string) of object;
```

C++

```
(Sender: TObject; var S: string) ( TVStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TVStringEvent.

21 SOAPMidas Namespace

This is namespace SOAPMidas.

Topics

Name	Description
Interfaces (see page 1276)	The following table lists interfaces in this documentation.
Functions (see page 1281)	The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 1281)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Interfaces

	Name	Description
	IAppServerSOAP (see page 1276)	IAppServerSOAP is the interface for communication with a SOAP data module.

21.1 Interfaces

The following table lists interfaces in this documentation.

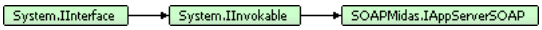
Interfaces

	Name	Description
	IAppServerSOAP (see page 1276)	IAppServerSOAP is the interface for communication with a SOAP data module.

21.1.1 IAppServerSOAP Interface

IAppServerSOAP is the interface for communication with a SOAP data module.

Class Hierarchy



Pascal

```
IAppServerSOAP = interface(IInvokable);
```

C++

```
__interface IAppServerSOAP : public IInvokable;
```

File

SOAPMidas

Description

TSoapConnection and TSoapDataModule use the IAppServerSOAP interface to communicate information in a SOAP-based multi-tiered database application. This is different from other DataSnap components, which use the IAppServer interface. IAppServerSOAP differs from IAppServer primarily in that it uses the stdcall calling convention rather than safecall. This makes it easier for marshalling interface calls when the client or server is written in C++.

Note: C++ method declarations that use IAppServerSOAP use the _di_IAppServerSOAP type instead. This type is a DelphiInterface (see page 3263) wrapper around the IAppServerSOAP interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IAppServerSOAP >  _di_IAppServerSOAP;
```

See Also

TSOAPConnection

TSoapDataModule (see page 1146)

IAppServer

21.1.1.1 IAppServerSOAP Members

The following tables list the members exposed by IAppServerSOAP.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SAS_ApplyUpdates (see page 1277)	This is SAS_ApplyUpdates, a member of class IAppServerSOAP.
	SAS_DataRequest (see page 1277)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 1278)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 1279)	This is SAS_GetParams, a member of class IAppServerSOAP.
	SAS_GetProviderNames (see page 1279)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 1279)	This is SAS_GetRecords, a member of class IAppServerSOAP.
	SAS_RowRequest (see page 1280)	This is SAS_RowRequest, a member of class IAppServerSOAP.

21.1.1.2 IAppServerSOAP Methods

The methods of the IAppServerSOAP class are listed here.

Public Methods

	Name	Description
	SAS_ApplyUpdates (see page 1277)	This is SAS_ApplyUpdates, a member of class IAppServerSOAP.
	SAS_DataRequest (see page 1277)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 1278)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 1279)	This is SAS_GetParams, a member of class IAppServerSOAP.
	SAS_GetProviderNames (see page 1279)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 1279)	This is SAS_GetRecords, a member of class IAppServerSOAP.
	SAS_RowRequest (see page 1280)	This is SAS_RowRequest, a member of class IAppServerSOAP.

21.1.1.2.1 IAppServerSOAP.SAS_ApplyUpdates Method

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta,
int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is SAS_ApplyUpdates, a member of class IAppServerSOAP.

21.1.1.2.2 IAppServerSOAP.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This

method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

See Also

OnDataRequest

DataRequest

21.1.1.2.3 IAppServerSOAP.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); stdcall;
```

C++

```
__fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant
Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls the SAS_Execute method of the specified provider after assigning any parameters. Output (see page 3901) parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's SAS_Execute method generates a BeforeSAS_Execute event, passes the command on to its associated dataset, and generates an AfterSAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method. To decode Params into a TParams object, use the Unpackparams routine.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

See Also

DataSet

beforeexecute

AfterExecute

Params

Execute

Execute

BeforeExecute

AfterExecute

Options

PackageParams

UnpackParams

21.1.1.2.4 IAppServerSOAP.SAS_GetParams Method

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

This is SAS_GetParams, a member of class IAppServerSOAP.

21.1.1.2.5 IAppServerSOAP.SAS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP (see page 1276) interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

ProviderName

Providers (see page 1168)

21.1.1.2.6 IAppServerSOAP.SAS_GetRecords Method

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is SAS_GetRecords, a member of class IAppServerSOAP.

21.1.1.2.7 IAppServerSOAP.SAS_RowRequest Method

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant Row, int
RequestType, OleVariant OwnerData);
```

Description

This is SAS_RowRequest, a member of class IAppServerSOAP.

21.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 1281)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

21.2.1 SOAPMidas.RegDefIAppServerInvClass Function

Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Pascal

```
procedure RegDefIAppServerInvClass;
```

C++

```
RegDefIAppServerInvClass();
```

File

SOAPMidas

Description

RegDefIAppServerInvClass registers TSoapDataModule as an invokable class that implements IAppServer as a Web Service. C++ applications must add a call to this procedure after the registration of IAppServerSOAP (see page 1276) in the program startup code if they are to be used by clients written using Kylix 2 or Delphi 5 prior to update patch 2. This call is not necessary for other clients of a Web Service-based application server, because they look for the IAppServerSOAP (see page 1276) interface.

Note: Delphi applications do not need to call this procedure.

The following example shows where to place the call to RegDefIAppServerInvClass:

```
void static RegTypes()
{
  InvRegistry()->RegisterInterface(__interfaceTypeinfo(IMyAppServerService));
  InvRegistry()->RegisterInvokableClass(__classid(TMyAppServerService),
MyAppServerServiceFactory);
  RegDefIAppServerInvClass();
}
#pragma statrtup RegTypes 32
```

See Also

TSoapDataModule

22 SOAPPasInv Namespace

This is namespace SOAPPasInv.

Topics

Name	Description
Classes (see page 1284)	The following table lists classes in this documentation.
Interfaces (see page 1293)	The following table lists interfaces in this documentation.
Functions (see page 1296)	The following table lists functions in this documentation.
Types (see page 1297)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapPascalInvoker (see page 1284)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Functions

	Name	Description
	GetSOAPHeaders (see page 1296)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Interfaces

	Name	Description
	IAdapterConverter (see page 1293)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 1294)	This is class SOAPPasInv.IInvokerConverter.

Types

Name	Description
TAfterDispatchEvent (see page 1297)	This is type SOAPPasInv.TAfterDispatchEvent.
TBeforeDispatchEvent (see page 1297)	This is type SOAPPasInv.TBeforeDispatchEvent.
TBeforeDispatchEvent2 (see page 1298)	This is type SOAPPasInv.TBeforeDispatchEvent2.
TOnExceptionEvent (see page 1298)	This is type SOAPPasInv.TOnExceptionEvent.

22.1 Classes

The following table lists classes in this documentation.

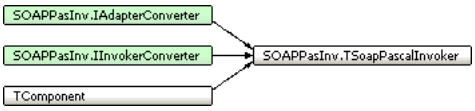
Classes

	Name	Description
	TSoapPascalInvoker (see page 1284)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

22.1.1 TSoapPascalInvoker Class

TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Class Hierarchy



Pascal

```
TSoapPascalInvoker = class(TComponent, IAdapterConverter, IInvokerConverter);
```

C++

```
class TSoapPascalInvoker : public TComponent, public IAdapterConverter, public
IInvokerConverter;
```

File

SOAPPasInv

Description

Use TSoapPascalInvoker as a base class for components that interpret SOAP requests. TSoapPascalInvoker implements the logic that parses a SOAP request, given its target invokable interface, executes the interface call, and encodes the results of that code.

Each TSoapPascalInvoker descendant receives SOAP requests that arrive via a specific transport protocol. The TSoapPascalInvoker descendant must implement the logic to take the incoming SOAP request, identify the target invokable interface, and look up its implementation class and runtime type information (RTTI).

See Also

IInvokable (see page 3505)

THTTPSoapPascalInvoker (see page 1205)

22.1.1.1 TSoapPascalInvoker Members

The following tables list the members exposed by TSoapPascalInvoker.

Protected Fields

	Name	Description
	FConverter (see page 1286)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1286)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1287)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1287)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1287)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	GetDomConverter (see page 1288)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1288)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 1289)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	Create (see page 1285)	Creates a new TSoapPascalInvoker (see page 1284) object.
V	Destroy (see page 1286)	Frees the memory associated with the TSoapPascalInvoker (see page 1284) object.
V	Invoke (see page 1288)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1290)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1291)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 1289)	Specifies the interface of the component that handles the marshaling of SOAP messages.

22.1.1.2 TSoapPascalInvoker.Create Constructor

Creates a new TSoapPascalInvoker (see page 1284) object.

Pascal

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

C++

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

Description

Call Create to instantiate an invoker at runtime. Invoker components added to Web modules at design time (including the one that the Soap Server wizard adds automatically) are created automatically.

The AOwner parameter specifies the component that is responsible for freeing the TSoapPascalInvoker (see page 1284) instance.

See Also

Owner

22.1.1.3 TSoapPascalInvoker.Destroy Destructor

Frees the memory associated with the TSoapPascalInvoker (see page 1284) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapPascalInvoker();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the invoker component is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

22.1.1.4 TSoapPascalInvoker Fields

The fields of the TSoapPascalInvoker class are listed here.

Protected Fields

	Name	Description
	FConverter (see page 1286)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1286)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1287)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1287)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1287)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

22.1.1.4.1 TSoapPascalInvoker.FConverter Field

Pascal

```
FConverter: IOPConvert;
```

C++

```
IOPConvert FConverter;
```

Description

This is FConverter, a member of class TSoapPascalInvoker.

22.1.1.4.2 TSoapPascalInvoker.FDomConverter Field

Pascal

```
FDomConverter: TOPToSoapDomConvert;
```

C++

```
TOPToSoapDomConvert FDomConverter;
```

Description

This is FDomConverter, a member of class TSoapPascalInvoker.

22.1.1.4.3 TSoapPascalInvoker.FHeadersIn Field

Pascal

```
FHeadersIn: THeaderList;
```

C++

```
THeaderList FHeadersIn;
```

Description

This is FHeadersIn, a member of class TSoapPascalInvoker.

22.1.1.4.4 TSoapPascalInvoker.FHeadersOut Field

Pascal

```
FHeadersOut: THeaderList;
```

C++

```
THeaderList FHeadersOut;
```

Description

This is FHeadersOut, a member of class TSoapPascalInvoker.

22.1.1.4.5 TSoapPascalInvoker.FSOAPHeaders Field

Pascal

```
FSOAPHeaders: TSOAPHeaders;
```

C++

```
TSOAPHeaders FSOAPHeaders;
```

Description

This is FSOAPHeaders, a member of class TSoapPascalInvoker.

22.1.1.5 TSoapPascalInvoker Methods

The methods of the TSoapPascalInvoker class are listed here.

Protected Methods

	Name	Description
	GetDomConverter (see page 1288)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1288)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 1289)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Invoke (see page 1288)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

22.1.1.5.1 TSoapPascalInvoker.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class TSoapPascalInvoker.

22.1.1.5.2 TSoapPascalInvoker.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class TSoapPascalInvoker.

22.1.1.5.3 TSoapPascalInvoker.Invoke Method

Interprets a SOAP request message, calls the target interface, and encodes any return values.

Pascal

```
procedure Invoke(AClass: TClass; IntfInfo: PTypeInfo; MethName: string; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall Invoke(TClass AClass, PTypeInfo IntfInfo, UnicodeString MethName, const
TStream Request, TStream Response, TWebServiceBindingType BindingType);
```

Description

Call Invoke to convert a SOAP request to an interface call on an invokable interface.

AClass is the class that implements the invokable interface. The invoker passes this to the invocation registry so that it can obtain an instance of the implementation class, which it uses to execute the SOAP request.

IntfInfo is the runtime type information (RTTI) from the invokable interface that is the target of the SOAP request. The invoker uses this to interpret the SOAP request.

MethName is the name of the method to call on the invokable interface.

Request is a stream from which Invoke can read the SOAP request. It contains the SOAP encoding of an interface call.

Response is a stream to which Invoke writes the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a

multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. Invoke changes BindingType to the format of the response message.

See Also

IInvokable (see page 3505)

InvRegistry

GetInvokableObjectFromClass

22.1.1.5.4 TSoapPascalInvoker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

22.1.1.6 TSoapPascalInvoker Properties

The properties of the TSoapPascalInvoker class are listed here.

Published Properties

	Name	Description
	Converter (see page 1289)	Specifies the interface of the component that handles the marshaling of SOAP messages.

22.1.1.6.1 TSoapPascalInvoker.Converter Property

Specifies the interface of the component that handles the marshaling of SOAP messages.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

Converter is the component that marshals and unmarshals method calls for the invoker. The invoker uses this object to interpret incoming method calls and to generate the results it returns.

See Also

TOPToSoapDomConvert

22.1.1.7 TSoapPascalInvoker Events

The events of the TSoapPascalInvoker class are listed here.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1290)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1291)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1291)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1292)	Occurs when TSoapPascalInvoker (see page 1284) sends each attachment included in the response to the current request.

22.1.1.7.1 TSoapPascalInvoker.AfterDispatchEvent Event

Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.

Pascal

```
property AfterDispatchEvent: TAfterDispatchEvent;
```

C++

```
__property TAfterDispatchEvent AfterDispatchEvent;
```

Description

Write (see page 3789) a AfterDispatchEvent event handler to examine a SOAP response after the invoker executes a request by calling the target invokable interface. The Invoke (see page 1288) method generates this event after it has executed a requested method and generated a response packet.

Note: AfterDispatchEvent only occurs if the method call is successful. If the invoker encounters an exception when trying to execute the specified method, it generates an OnExceptionEvent (see page 1291) instead.

See Also

BeforeDispatchEvent (see page 1291)

Invoke (see page 1288)

OnExceptionEvent (see page 1291)

22.1.1.7.2 TSoapPascalInvoker.BeforeDispatchEvent Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent: TBeforeDispatchEvent;
```

C++

```
__property TBeforeDispatchEvent BeforeDispatchEvent;
```

Description

Write (see page 3789) a BeforeDispatchEvent event handler to examine a SOAP request before the invoker calls on the target invokable interface to execute it. The Invoke (see page 1288) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 1290)

Invoke (see page 1288)

BeforeDispatchEvent2 (see page 1291)

22.1.1.7.3 TSoapPascalInvoker.BeforeDispatchEvent2 Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent2: TBeforeDispatchEvent2;
```

C++

```
__property TBeforeDispatchEvent2 BeforeDispatchEvent2;
```

Description

Write (see page 3789) a BeforeDispatchEvent2 event handler to override the default handling of an incoming request. The Invoke (see page 1288) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 1290)

Invoke (see page 1288)

BeforeDispatchEvent (see page 1291)

22.1.1.7.4 TSoapPascalInvoker.OnExceptionEvent Event

Occurs when the invoker encounters an exception while executing a SOAP request.

Pascal

```
property OnExceptionEvent: TOnExceptionEvent;
```

C++

```
__property TOnExceptionEvent OnExceptionEvent;
```

Description

Write (see page 3789) a OnExceptionEvent event handler to examine a SOAP fault packet after the invoker encounters an exception when executing a SOAP request. The Invoke (see page 1288) method generates this event if it encounters an exception, after it generates the SOAP fault packet that describes the problem.

See Also

AfterDispatchEvent (see page 1290)

22.1.1.7.5 TSoapPascalInvoker.OnGetAttachment Event

Occurs when TSoapPascalInvoker (see page 1284) reads each attachment included in an incoming request.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

Write (see page 3789) an OnGetAttachment event handler to examine the encoded attachments included with an incoming request or to redirect them so that they are saved to a specific file. OnGetAttachment occurs once for each attachment received.

Tip: Use the FileName property of the attachment object to redirect the attachment to a different file.

See Also

OnSendAttachment (see page 1292)

BeforeDispatchEvent (see page 1291)

22.1.1.7.6 TSoapPascalInvoker.OnSendAttachment Event

Occurs when TSoapPascalInvoker (see page 1284) sends each attachment included in the response to the current request.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

Write (see page 3789) an OnSendAttachment event handler to examine or alter the attachments that are to be included in a message that represents the results of a method call. OnSendAttachment occurs once for each attachment sent with the response message.

See Also

AfterDispatchEvent (see page 1290)

OnGetAttachment (see page 1292)

22.2 Interfaces

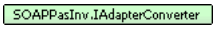
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterConverter (see page 1293)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 1294)	This is class SOAPPasInv.IInvokerConverter.

22.2.1 IAdapterConverter Interface

Class Hierarchy



Pascal

```
IAdapterConverter = interface;
```

C++

```
__interface IAdapterConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IAdapterConverter.

22.2.1.1 IAdapterConverter Members

The following tables list the members exposed by IAdapterConverter.

Public Methods

	Name	Description
	GetDomConverter (see page 1294)	This is GetDomConverter, a member of class IAdapterConverter.

22.2.1.2 IAdapterConverter Methods

The methods of the IAdapterConverter class are listed here.

Public Methods

	Name	Description
	GetDomConverter (see page 1294)	This is GetDomConverter, a member of class IAdapterConverter.

22.2.1.2.1 IAdapterConverter.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class IAdapterConverter.

22.2.2 IInvokerConverter Interface

Class Hierarchy

SOAPPasInv.IInvokerConverter

Pascal

```
IInvokerConverter = interface;
```

C++

```
__interface IInvokerConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IInvokerConverter.

22.2.2.1 IInvokerConverter Members

The following tables list the members exposed by IInvokerConverter.

Public Methods

	Name	Description
	GetInvoker (see page 1295)	This is GetInvoker, a member of class IInvokerConverter.

22.2.2.2 IInvokerConverter Methods

The methods of the IInvokerConverter class are listed here.

Public Methods

	Name	Description
	GetInvoker (see page 1295)	This is GetInvoker, a member of class IInvokerConverter.

22.2.2.2.1 IInvokerConverter.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class IInvokerConverter.

22.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPHeaders (see page 1296)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

22.3.1 SOAPPasInv.GetSOAPHeaders Function

Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Pascal

```
function GetSOAPHeaders: ISOAPHeaders;
```

C++

```
ISOAPHeaders GetSOAPHeaders();
```

File

SOAPPasInv

Description

Call GetSOAPHeaders from an application that implements a Web Service when you do not use TInvokableClass as a base class for the implementation of an invokable interface. GetSOAPHeaders returns an ISOAPHeaders interface, which lets your application examine the SOAP headers in an incoming request message and to add SOAP headers to be sent with the outgoing response.

See Also

TInvokableClass

Defining and Using SOAP Headers

22.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterDispatchEvent (see page 1297)	This is type SOAPPasInv.TAfterDispatchEvent.
TBeforeDispatchEvent (see page 1297)	This is type SOAPPasInv.TBeforeDispatchEvent.
TBeforeDispatchEvent2 (see page 1298)	This is type SOAPPasInv.TBeforeDispatchEvent2.
TOnExceptionEvent (see page 1298)	This is type SOAPPasInv.TOnExceptionEvent.

22.4.1 SOAPPasInv.TAfterDispatchEvent Type

Pascal

```
TAfterDispatchEvent = procedure (const MethodName: string; SOAPResponse: TStream) of object;
```

C++

```
(const MethodName: string; SOAPResponse: TStream) ( TAfterDispatchEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TAfterDispatchEvent.

22.4.2 SOAPPasInv.TBeforeDispatchEvent Type

Pascal

```
TBeforeDispatchEvent = procedure (const MethodName: string; const Request: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream) ( TBeforeDispatchEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TBeforeDispatchEvent.

22.4.3 SOAPPasInv.TBeforeDispatchEvent2 Type

Pascal

```
TBeforeDispatchEvent2 = procedure (const MethodName: string; const Request: TStream; Response:
TStream; var BindingType: TWebServiceBindingType; var Handled: Boolean) of object;
```

C++

```
(const MethodName: string; const Request: TStream; Response: TStream; var BindingType:
TWebServiceBindingType; var Handled: Boolean) ( TBeforeDispatchEvent2)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TBeforeDispatchEvent2.

22.4.4 SOAPPasInv.TOnExceptionEvent Type

Pascal

```
TOnExceptionEvent = procedure (const MethodName: string; const Request: TStream; const
Response: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream; const Response: TStream) (
TOnExceptionEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TOnExceptionEvent.

23 Sockets Namespace

This is namespace Sockets.

Topics

Name	Description
Classes (see page 1301)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 1403)	The following table lists structs, records, enums in this documentation.
Types (see page 1408)	The following table lists types in this documentation.
Constants (see page 1412)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 1301)	Exception (see page 4015) class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 1302)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 1322)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 1326)	Encapsulates the client network functions.
	TCustomTcpServer (see page 1332)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 1341)	Encapsulates the network support functions.
	TRawSocket (see page 1353)	Encapsulates a raw socket.
	TServerSocketThread (see page 1356)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 1362)	TCP client component.
	TTcpServer (see page 1375)	TTcpServer is the TCP server component.
	TUdpSocket (see page 1387)	UDP component.

Constants

Name	Description
CRLF (see page 1412)	This is constant Sockets.CRLF.

Structs, Records, Enums

	Name	Description
	TIPHeader (see page 1403)	This is record Sockets.TIPHeader.
	TServerSocketBlockMode (see page 1404)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 1404)	Set of possible protocol family types used by a socket.
	TSocketType (see page 1406)	The TSocketType type defines the set of possible socket types.

Types

Name	Description
TGetThreadEvent (see page 1408)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 1408)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 1409)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 1409)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 1409)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 1410)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 1410)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 1410)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 1411)	This is type Sockets.TSocketProtocol.

23.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 1301)	Exception (see page 4015) class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 1302)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 1322)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 1326)	Encapsulates the client network functions.
	TCustomTcpServer (see page 1332)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 1341)	Encapsulates the network support functions.
	TRawSocket (see page 1353)	Encapsulates a raw socket.
	TServerSocketThread (see page 1356)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 1362)	TCP client component.
	TTcpServer (see page 1375)	TTcpServer is the TCP server component.
	TUdpSocket (see page 1387)	UDP component.

23.1.1 ESocketError Class

Exception (see page 4015) class for failures that occur when trying to create, use, or shut down Windows socket objects.

Class Hierarchy

Pascal

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

C++

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

File

Sockets

Description

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

ESocketError is thrown if:

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

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

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

A socket cannot be properly closed.

There are two ESocketError classes defined, one in the ScktComp (see page 1) unit and one in the Sockets (see page 1299) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

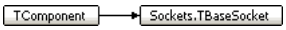
See Also

TCustomWinSocket (see page 54)

23.1.2 TBaseSocket Class

TBaseSocket encapsulates local operations on file descriptors.

Class Hierarchy



Pascal

```
TBaseSocket = class(TComponent);
```

C++

```
class TBaseSocket : public TComponent;
```

File

Sockets

Description

The TBaseSocket component encapsulates local operations on files descriptors, which are objects that a program uses to read from or write to an open file, or open a network socket or a variety of other devices.

Do not create instances of TBaseSocket. Use TBaseSocket as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TBaseSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TIpSocket (see page 1341)

23.1.2.1 TBaseSocket Members

The following tables list the members exposed by TBaseSocket.

Protected Methods

	Name	Description
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
V	Close (see page 1305)	Shuts down the socket connection.
V	Create (see page 1303)	Creates an instance of TBaseSocket (see page 1302).
V	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
V	Open (see page 1307)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1315)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	SockType (see page 1317)	Sets or returns the socket type.

23.1.2.2 TBaseSocket.Create Constructor

Creates an instance of TBaseSocket (see page 1302).

Pascal

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

C++

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

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TBaseSocket (see page 1302) object.

See Also

TBaseSocket (see page 1302)

23.1.2.3 TBaseSocket.Destroy Destructor

Frees the memory associated with the TBaseSocket (see page 1302) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseSocket();
```

Description

For C++, do not call ~TBaseSocket (see page 1302) directly. Instead, use the **delete** keyword on the object, which causes ~TBaseSocket (see page 1302) to be invoked automatically.

See Also

delete

TBaseSocket (see page 1302)

23.1.2.4 TBaseSocket Methods

The methods of the TBaseSocket class are listed here.

Protected Methods

	Name	Description
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
	Close (see page 1305)	Shuts down the socket connection.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
	Open (see page 1307)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

23.1.2.4.1 TBaseSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 1314) property to **false**.

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

See Also

Active (see page 1314)

Open (see page 1307)

23.1.2.4.2 TBaseSocket.DoCreateHandle Method

Pascal

```
procedure DoCreateHandle; dynamic;
```

C++

```
__fastcall DoCreateHandle();
```

Description

This is DoCreateHandle, a member of class TBaseSocket.

23.1.2.4.3 TBaseSocket.DoDestroyHandle Method

Pascal

```
procedure DoDestroyHandle; dynamic;
```

C++

```
__fastcall DoDestroyHandle();
```

Description

This is DoDestroyHandle, a member of class TBaseSocket.

23.1.2.4.4 TBaseSocket.DoHandleError Method

Pascal

```
procedure DoHandleError; dynamic;
```

C++

```
__fastcall DoHandleError();
```

Description

This is DoHandleError, a member of class TBaseSocket.

23.1.2.4.5 TBaseSocket.DoReceive Method

Pascal

```
procedure DoReceive(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoReceive(const char * Buf, int DataLen);
```

Description

This is DoReceive, a member of class TBaseSocket.

23.1.2.4.6 TBaseSocket.DoSend Method

Pascal

```
procedure DoSend(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoSend(const char * Buf, int DataLen);
```

Description

This is DoSend, a member of class TBaseSocket.

23.1.2.4.7 TBaseSocket.ErrorCheck Method

Pascal

```
function ErrorCheck(rc: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ErrorCheck(int rc);
```

Description

This is ErrorCheck, a member of class TBaseSocket.

23.1.2.4.8 TBaseSocket.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling

pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

23.1.2.4.9 TBaseSocket.MapDomain Method

Maps the domain.

Pascal

```
function MapDomain(sd: TSocketDomain): Integer;
```

C++

```
__fastcall int MapDomain(TSocketDomain sd);
```

Description

MapDomain maps the socket domain to an integer based on the value supplied by the Domain (see page 1316) property.

See Also

Domain (see page 1316)

MapSockType (see page 1307)

23.1.2.4.10 TBaseSocket.MapSockType Method

Maps the socket type.

Pascal

```
function MapSockType(st: TSocketType): Integer;
```

C++

```
__fastcall int MapSockType(TSocketType st);
```

Description

MapSockType maps the socket type to an integer based on the value supplied by the SockType (see page 1317) property.

See Also

SockType (see page 1317)

23.1.2.4.11 TBaseSocket.Open Method

Creates the socket connection.

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create the socket connection. Descendants of TBaseSocket (see page 1302) use Open to initiate the socket connection.

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

See Also

TBaseSocket (see page 1302)

Active (see page 1314)

Close (see page 1305)

23.1.2.4.12 TBaseSocket.PeekBuf Method

Reads the buffer.

Pascal

```
function PeekBuf(var Buf; BufSize: Integer): Integer;
```

C++

```
__fastcall int PeekBuf( Buf, int BufSize);
```

Description

Use PeekBuf to make a copy of data waiting in the socket buffer. Use the **Buf** parameter to assign a variable to the buffer, and use **Bufsize** to set the size of the buffer to be copied. PeekBuf only copies, but does not otherwise affect the contents of the buffer.

See Also

ReceiveBuf (see page 1308)

SendBuf (see page 1310)

23.1.2.4.13 TBaseSocket.ReceiveBuf Method

Reads data buffers of to the socket.

Pascal

```
function ReceiveBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int BufSize, int Flags = 0);
```

Description

ReceiveBuf reads data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be read. The **flags** argument can be any of the flags used by the Linux **recv** function. Refer to recv(2) in the Linux manual pages for more information.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
```

```
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

PeekBuf (see page 1308)

SendBuf (see page 1310)

23.1.2.4.14 TBaseSocket.ReceiveLn Method

Reads delimited lines of data from the socket.

Pascal

```
function Receiveln(const eol: string = CRLF): string;
```

C++

```
__fastcall UnicodeString Receiveln(const UnicodeString eol = CRLF);
```

Description

Receiveln reads a delimited chunk of data from the socket. The **delimeter** cannot be part of the character set. Receiveln fills the buffer until it reaches the **delimeter**.

See Also

Sendln (see page 1311)

OnReceive (see page 1320)

23.1.2.4.15 TBaseSocket.Select Method

Tests for socket activity.

Pascal

```
function Select(ReadReady: PBoolean; WriteReady: PBoolean; ExceptFlag: PBoolean; TimeOut:
Integer = 0): Boolean;
```

C++

```
__fastcall Boolean Select(PBoolean ReadReady, PBoolean WriteReady, PBoolean ExceptFlag, int
TimeOut = 0);
```

Description

Use Select to test for socket activity. Select indicates whether there is any data on the socket waiting to be sent or whether an error condition exists.

The parameter **TimeOut** is the time-out value. The socket waits for activity up to **TimeOut** milliseconds before returning. If **TimeOut** is not explicitly specified, the default value is zero. If **TimeOut** is any negative number, the socket is blocked. It waits for activity indefinitely, which can be useful for servers.

See Also

WaitForData (see page 1313)

23.1.2.4.16 TBaseSocket.SendBuf Method

Writes data buffers of size bufsize from the socket.

Pascal

```
function SendBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int BufSize, int Flags = 0);
```

Description

SendBuf sends data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be sent. The flags argument can be any of the flags used by the Linux **send** function. Refer to the send Linux manual page for more information.

See Also

PeekBuf (see page 1308)

ReceiveBuf (see page 1308)

23.1.2.4.17 TBaseSocket.Sendln Method

Writes delimited lines of data to the socket.

Pascal

```
function Sendln(s: string; const eol: string = CRLF): Integer;
```

C++

```
__fastcall int Sendln(UnicodeString s, const UnicodeString eol = CRLF);
```

Description

Sendln sends a delimited chunk of data to the socket. The **delimeter** cannot be part of the character set. Sendln fills the buffer until it reaches the **delimeter**.

See Also

Receiveln (see page 1310)

OnReceive (see page 1320)

23.1.2.4.18 TBaseSocket.SendStream Method

Writes a stream of data to the socket.

Pascal

```
function SendStream(AStream: TStream): Integer;
```

C++

```
__fastcall int SendStream(TStream AStream);
```

Description

SendStream sends a stream of data to the socket.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;
```

```
procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

OnReceive (see page 1320)

23.1.2.4.19 TBaseSocket.SetBytesReceived Method

Pascal

```
procedure SetBytesReceived(Value: Cardinal);
```

C++

```
__fastcall SetBytesReceived(unsigned Value);
```

Description

This is SetBytesReceived, a member of class TBaseSocket.

23.1.2.4.20 TBaseSocket.SetBytesSent Method

Pascal

```
procedure SetBytesSent(Value: Cardinal);
```

C++

```
__fastcall SetBytesSent(unsigned Value);
```

Description

This is SetBytesSent, a member of class TBaseSocket.

23.1.2.4.21 TBaseSocket.WaitForData Method

Waits for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(TimeOut: Integer = 0): Boolean;
```

C++

```
__fastcall Boolean WaitForData(int TimeOut = 0);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns **true** if the socket connection is ready. WaitForData returns **false** if the socket connection is not ready after **TimeOut** milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, ReceiveBuf (see page 1308), ReceiveIn (see page 1310), SendBuf (see page 1310), and SendIn (see page 1311) method calls may time out before any data is transferred.

See Also

ReceiveBuf (see page 1308)

ReceiveIn (see page 1310)

SendBuf (see page 1310)

SendIn (see page 1311)

23.1.2.5 TBaseSocket Properties

The properties of the TBaseSocket class are listed here.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1315)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	SockType (see page 1317)	Sets or returns the socket type.

23.1.2.5.1 TBaseSocket.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open (see page 1307) or Close (see page 1305) method to open or close the connection.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
```

```
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

Close (see page 1305)

Open (see page 1307)

23.1.2.5.2 TBaseSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode (see page 1409) type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1409)

OnReceive (see page 1320)

OnSend (see page 1321)

23.1.2.5.3 TBaseSocket.BytesReceived Property

Returns the number of bytes received by the socket.

Pascal

```
property BytesReceived: Cardinal;
```

C++

```
__property unsigned BytesReceived;
```

Description

BytesReceived returns the number of bytes received by the socket.

See Also

BytesSent (see page 1315)

23.1.2.5.4 TBaseSocket.BytesSent Property

Returns the number of bytes sent from the socket.

Pascal

```
property BytesSent: Cardinal;
```

C++

```
__property unsigned BytesSent;
```

Description

BytesSent returns the number of bytes sent from the socket.

See Also

BytesReceived (see page 1315)

23.1.2.5.5 TBaseSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain (see page 1404) type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 1307)

TSocketDomain (see page 1404)

23.1.2.5.6 TBaseSocket.Handle Property

Provides access to the unique-4 byte identifier that represents the instance of the socket object.

Pascal

```
property Handle: TSocket;
```

C++

```
__property TSocket Handle;
```

Description

Use Handle to return a typed pointer to the instance of the socket class. This Handle can be used when making low level function calls into the QT shared libraries, where the function call requires an instance pointer to the object.

23.1.2.5.7 TBaseSocket.Protocol Property

Sets or returns the protocol used by the socket.

Pascal

```
property Protocol: TSocketProtocol;
```

C++

```
__property TSocketProtocol Protocol;
```

Description

Protocol sets or returns the protocol used by the socket. The socket protocol can be IP, TCP, UDP, or other network protocols.

See Also

TTcpServer (see page 1375)

23.1.2.5.8 TBaseSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 1406) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 1406)

23.1.2.6 TBaseSocket Events

The events of the TBaseSocket class are listed here.

Public Events

	Name	Description
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

23.1.2.6.1 TBaseSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write (see page 3789) an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}
```

```
const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1314)

Handle (see page 1316)

23.1.2.6.2 TBaseSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write (see page 3789) an OnDestroyHandle event handler to take specific action when the socket is deactivated.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin
```

```
    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

23.1.2.6.3 TBaseSocket.OnError Event

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

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write (see page 3789) an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 1301) exception from being raised.

See Also

ESocketError (see page 1301)

23.1.2.6.4 TBaseSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write (see page 3789) an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 127)

BytesReceived (see page 1315)

ReceiveBuf (see page 1308)

ReceiveIn (see page 1310)

23.1.2.6.5 TBaseSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write (see page 3789) an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
```

```
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TWinSocketStream (see page 127)

SendBuf (see page 1310)

Sendln (see page 1311)

23.1.3 TClientSocketThread Class

Enables creation of separate threads of execution for the client socket in an application.

Class Hierarchy

Pascal

```
TClientSocketThread = class(TThread);
```

C++

```
class TClientSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 1323) a TClientSocketThread (or a descendant of the class) to represent an execution thread for the client socket in a multi-threaded application. Each new instance of TClientSocketThread, a TThread descendant, is a new thread of execution. Using a separate TClientSocketThread allows the client socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TServerSocketThread (see page 1356)

TThread

23.1.3.1 TClientSocketThread Members

The following tables list the members exposed by TClientSocketThread.

Protected Methods

	Name	Description
	SyncProc (see page 1325)	Method where data synchronization is actually performed.

Public Methods

	Name	Description
	Create (see page 1323)	Creates an instance of a TClientSocketThread (see page 1322) object.
V	Destroy (see page 1323)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 1324)	Executes when the thread is run.
	ExecuteSyncProc (see page 1324)	Synchronizes data between threads.

Public Properties

	Name	Description
R	ClientSocket (see page 1325)	Refers to the client socket handling the connection in that thread.
R	ServerSocketThread (see page 1326)	Specifies the server socket's thread.

23.1.3.2 TClientSocketThread.Create Constructor

Creates an instance of a TClientSocketThread (see page 1322) object.

Pascal

```
constructor Create(ServerSocketThread: TServerSocketThread);
```

C++

```
__fastcall TClientSocketThread(TServerSocketThread ServerSocketThread);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application. If the **CreateSuspended** parameter is false, Execute (see page 1324) is called immediately. If **CreateSuspended** is true, Execute (see page 1324) is not called until after Resume is called.

See Also

TClientSocketThread (see page 1322)

Destroy (see page 1323)

Execute (see page 1324)

Resume

23.1.3.3 TClientSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

See Also

delete

Create

FreeOnTerminate

23.1.3.4 TClientSocketThread Methods

The methods of the TClientSocketThread class are listed here.

Protected Methods

	Name	Description
	SyncProc (see page 1325)	Method where data synchronization is actually performed.

Public Methods

	Name	Description
	Execute (see page 1324)	Executes when the thread is run.
	ExecuteSyncProc (see page 1324)	Synchronizes data between threads.

23.1.3.4.1 TClientSocketThread.Execute Method

Executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated

Synchronize

23.1.3.4.2 TClientSocketThread.ExecuteSyncProc Method

Synchronizes data between threads.

Pascal

```
procedure ExecuteSyncProc;
```

C++

```
__fastcall ExecuteSyncProc();
```

Description

Objects in the client socket thread should not directly manipulate data in the main thread. Attempting to do so will cause synchronization problems and, most likely, the application will crash. To avoid synchronization errors, override the SyncProc (see page 1325) method to include instructions that must be executed by the main thread.

See Also

Synchronize

SyncProc (see page 1325)

23.1.3.4.3 TClientSocketThread.SyncProc Method

Method where data synchronization is actually performed.

Pascal

```
procedure SyncProc; virtual;
```

C++

```
virtual __fastcall SyncProc();
```

Description

In subclasses of TClientSocketThread (see page 1322), override SyncProc to perform synchronization between threads. Put code in it to be executed by the main thread.

Warning: Do not call this protected method directly; instead, call ExecuteSyncProc (see page 1324).

See Also

TClientSocketThread (see page 1322)

ExecuteSyncProc (see page 1324)

Synchronize

23.1.3.5 TClientSocketThread Properties

The properties of the TClientSocketThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 1325)	Refers to the client socket handling the connection in that thread.
R	ServerSocketThread (see page 1326)	Specifies the server socket's thread.

23.1.3.5.1 TClientSocketThread.ClientSocket Property

Refers to the client socket handling the connection in that thread.

Pascal

```
property ClientSocket: TCustomIpClient;
```

C++

```
__property TCustomIpClient ClientSocket;
```

Description

Use ClientSocket to obtain access to the client socket object handling the connection in the current thread of execution.

See Also

TCustomIpClient (see page 1326)

23.1.3.5.2 TClientSocketThread.ServerSocketThread Property

Specifies the server socket's thread.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

Use ServerSocketThread get access to the thread object of the server socket.

See Also

TServerSocketThread (see page 1356)

23.1.4 TCustomIpClient Class

Encapsulates the client network functions.

Class Hierarchy

Pascal

```
TCustomIpClient = class(TIpSocket);
```

C++

```
class TCustomIpClient : public TIpSocket;
```

File

Sockets

Description

TCustomIpClient encapsulates the client network functions, linking sockets to network protocols such as TCP/IP and UDP, as well as raw sockets and custom protocols. Do not create instances of TCustomIpClient. Use TCustomIpClient as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomIpClient provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 1302)

23.1.4.1 TCustomIpClient Members

The following tables list the members exposed by TCustomIpClient.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.

	DoConnect (see page 1330)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoDisconnect (see page 1330)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnConnect (see page 1332)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1332)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Close (see page 1329)	Shuts down the socket connection.
	Connect (see page 1329)	Connects the local socket to the remote socket.
	Create (see page 1328)	Creates an instance of TCustomIpClient (see page 1326).
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	Disconnect (see page 1329)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	GetThreadObject (see page 1330)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
	Open (see page 1330)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1315)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
R	Connected (see page 1331)	Indicates whether the socket connection is open and available for communication with other machines.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.
	SockType (see page 1317)	Sets or returns the socket type.

23.1.4.2 TCustomIpClient.Create Constructor

Creates an instance of TCustomIpClient (see page 1326).

Pascal

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

C++

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

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomIpClient (see page 1326) object.

See Also

Destroy

23.1.4.3 TCustomIpClient Methods

The methods of the TCustomIpClient class are listed here.

Protected Methods

	Name	Description
V	DoConnect (see page 1330)	This is DoConnect, a member of class TCustomIpClient.
V	DoDisconnect (see page 1330)	This is DoDisconnect, a member of class TCustomIpClient.

Public Methods

	Name	Description
V	Close (see page 1329)	Shuts down the socket connection.
	Connect (see page 1329)	Connects the local socket to the remote socket.
	Disconnect (see page 1329)	Disconnects the local socket from the remote socket.
	GetThreadObject (see page 1330)	Returns a reference to the TThread object for the currently executing thread.
V	Open (see page 1330)	Creates the socket connection.

23.1.4.3.1 TCustomIpClient.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. This method performs the same functions as Disconnect (see page 1329).

See Also

Disconnect (see page 1329)

Open (see page 1330)

Active (see page 1314)

23.1.4.3.2 TCustomIpClient.Connect Method

Connects the local socket to the remote socket.

Pascal

```
function Connect: Boolean;
```

C++

```
__fastcall Boolean Connect();
```

Description

Call Connect to connect the local socket to the remote socket.

See Also

Connected (see page 1331)

Close (see page 1329)

Disconnect (see page 1329)

23.1.4.3.3 TCustomIpClient.Disconnect Method

Disconnects the local socket from the remote socket.

Pascal

```
procedure Disconnect;
```

C++

```
__fastcall Disconnect();
```

Description

Call Disconnect to break the connection between the local and remote sockets. This method performs the same function as Close (see page 1329).

See Also

Connected (see page 1331)

Close (see page 1329)

Connect (see page 1329)

23.1.4.3.4 TCustomIpClient.DoConnect Method

Pascal

```
procedure DoConnect; virtual;
```

C++

```
virtual __fastcall DoConnect();
```

Description

This is DoConnect, a member of class TCustomIpClient.

23.1.4.3.5 TCustomIpClient.DoDisconnect Method

Pascal

```
procedure DoDisconnect; virtual;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

This is DoDisconnect, a member of class TCustomIpClient.

23.1.4.3.6 TCustomIpClient.GetThreadObject Method

Returns a reference to the TThread object for the currently executing thread.

Pascal

```
function GetThreadObject: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread GetThreadObject();
```

Description

Call GetThreadObject to obtain a reference to the currently executing thread. GetThreadObject is only useful when BlockMode (see page 1315) is set to **bmBlockingMode**, which means that the client is executing in its own thread. If this method is called from the main application thread, the value NULL is returned.

See Also

TThread

BlockMode (see page 1315)

23.1.4.3.7 TCustomIpClient.Open Method

Creates the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create and initiate the socket connection.

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

See Also

Active (see page 1314)

Close (see page 1329)

23.1.4.4 TCustomIpClient Properties

The properties of the TCustomIpClient class are listed here.

Public Properties

	Name	Description
	Connected (see page 1331)	Indicates whether the socket connection is open and available for communication with other machines.

23.1.4.4.1 TCustomIpClient.Connected Property

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

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read (see page 3727) Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open (see page 1330) method. To close a socket, use the Closemethod.

See Also

Close (see page 1329)

Open (see page 1330)

OnConnect (see page 1332)

23.1.4.5 TCustomIpClient Events

The events of the TCustomIpClient class are listed here.

Public Events

	Name	Description
	OnConnect (see page 1332)	Occurs on client sockets just after the connection to the server is opened.
	OnDisconnect (see page 1332)	Occurs on client sockets just after the connection to the server is closed.

23.1.4.5.1 TCustomIpClient.OnConnect Event

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

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write (see page 3789) an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 1331)

OnDisconnect (see page 1332)

23.1.4.5.2 TCustomIpClient.OnDisconnect Event

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

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write (see page 3789) an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

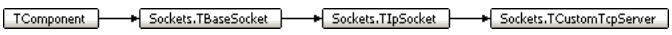
Connected (see page 1331)

OnConnect (see page 1332)

23.1.5 TCustomTcpServer Class

Ancestor object of all the server socket descendants.

Class Hierarchy



Pascal

```
TCustomTcpServer = class(TIpSocket);
```

C++

```
class TCustomTcpServer : public TIpSocket;
```

File

Sockets

Description

Do not create instances of TCustomTcpServer. Use TCustomTcpServer as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomTcpServer provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

23.1.5.1 TCustomTcpServer Members

The following tables list the members exposed by TCustomTcpServer.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoAccept (see page 1336)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Listen (see page 1337)	This is Listen, a member of class TCustomTcpServer.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnAccept (see page 1339)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetThread (see page 1340)	A wrapper for the TServerSocketThread (see page 1356) event of the same name.
	OnListening (see page 1341)	Occurs when a server socket begins listening for connections.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Accept (see page 1335)	This is the overview for the Accept method overload.
	Close (see page 1336)	Shuts down the socket connection.
	Create (see page 1334)	Creates an instance of TCustomTcpServer (see page 1332).
	Destroy (see page 1335)	Frees the memory associated with the TCustomTcpServer (see page 1332) object.

	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
V	Open (see page 1337)	Opens the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForConnection (see page 1337)	Listens for TCP connections.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1338)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 1338)	Sets the server application to listen for client requests.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.
	ServerSocketThread (see page 1338)	Specifies the TServerSocketThread (see page 1356) in use.
	SockType (see page 1317)	Sets or returns the socket type.

23.1.5.2 TCustomTcpServer.Create Constructor

Creates an instance of TCustomTcpServer (see page 1332).

Pascal

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

C++

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

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomTcpServer (see page 1332) object.

See Also

TCustomTcpServer (see page 1332)

Destroy (see page 1335)

23.1.5.3 TCustomTcpServer.Destroy Destructor

Frees the memory associated with the TCustomTcpServer (see page 1332) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTcpServer();
```

Description

For C++, do not call Destroy directly. Instead, use the **delete** keyword on the object, which causes Destroy to be invoked automatically.

See Also

delete

23.1.5.4 TCustomTcpServer Methods

The methods of the TCustomTcpServer class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 1336)	This is DoAccept, a member of class TCustomTcpServer.
	Listen (see page 1337)	This is Listen, a member of class TCustomTcpServer.

Public Methods

	Name	Description
	Accept (see page 1335)	This is the overview for the Accept method overload.
	Close (see page 1336)	Shuts down the socket connection.
	Open (see page 1337)	Opens the socket connection.
	WaitForConnection (see page 1337)	Listens for TCP connections.

23.1.5.4.1 Accept Method

This is the overview for the Accept method overload.

Overload List

	Name	Description
	TCustomTcpServer.Accept () (see page 1336)	Accepts a connection to a client socket.
	TCustomTcpServer.Accept (TCustomIpClient) (see page 1336)	Accepts a connection to a client socket.

23.1.5.4.1.1 TCustomTcpServer.Accept Method ()

Accepts a connection to a client socket.

Pascal

```
function Accept: Boolean; overload;
function Accept(var ClientSocket: TCustomIpClient): Boolean; overload;
```

C++

```
__fastcall Boolean Accept();
__fastcall Boolean Accept(TCustomIpClient ClientSocket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection.

The Accept method for TCustomTcpServerdoes nothing with the notification. Descendants of TCustomTcpServer (see page 1332) that represent listening server sockets override this method to accept the connection to the client socket. In the second declaration of the method, a TCustomIpClient (see page 1326) object is created (as the **ClientSocket** parameter) to handle the connection.

See Also

TCustomIpClient (see page 1326)

TCustomTcpServer (see page 1332)

OnAccept (see page 1339)

23.1.5.4.2 TCustomTcpServer.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 1314) property to false.

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

See Also

Active (see page 1314)

Open (see page 1337)

23.1.5.4.3 TCustomTcpServer.DoAccept Method

Pascal

```
procedure DoAccept(ClientSocket: TCustomIpClient); virtual;
```

C++

```
virtual __fastcall DoAccept(TCustomIpClient ClientSocket);
```

Description

This is DoAccept, a member of class TCustomTcpServer.

23.1.5.4.4 TCustomTcpServer.Listen Method

Pascal

```
function Listen(backlog: Integer = SOMAXCONN): Boolean;
```

C++

```
__fastcall Boolean Listen(int backlog = SOMAXCONN);
```

Description

This is Listen, a member of class TCustomTcpServer.

23.1.5.4.5 TCustomTcpServer.Open Method

Opens the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 1314) property to true.

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

See Also

Active (see page 1314)

Close (see page 1336)

23.1.5.4.6 TCustomTcpServer.WaitForConnection Method

Listens for TCP connections.

Pascal

```
function WaitForConnection: Boolean;
```

C++

```
__fastcall Boolean WaitForConnection();
```

Description

Call WaitForConnection to listen for TCP connections on the port specified by the LocalPort (see page 1349) property. The thread will block until a connection request is received.

See Also

LocalPort (see page 1349)

Accept (see page 1336)

23.1.5.5 TCustomTcpServer Properties

The properties of the TCustomTcpServer class are listed here.

Public Properties

	Name	Description
	BlockMode (see page 1338)	Sets or returns the blocking mode of the socket.
R	Listening (see page 1338)	Sets the server application to listen for client requests.
	ServerSocketThread (see page 1338)	Specifies the TServerSocketThread (see page 1356) in use.

23.1.5.5.1 TCustomTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TServerSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode (see page 1404) type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 1320)

OnSend (see page 1321)

TServerSocketBlockMode (see page 1404)

23.1.5.5.2 TCustomTcpServer.Listening Property

Sets the server application to listen for client requests.

Pascal

```
property Listening: Boolean;
```

C++

```
__property Boolean Listening;
```

Description

Listening sets the server application to listen for client requests.

23.1.5.5.3 TCustomTcpServer.ServerSocketThread Property

Specifies the TServerSocketThread (see page 1356) in use.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

ServerSocketThread is a reference to the server socket thread object.

See Also

TServerSocketThread (see page 1356)

Listening (see page 1338)

23.1.5.6 TCustomTcpServer Events

The events of the TCustomTcpServer class are listed here.

Public Events

	Name	Description
	OnAccept (see page 1339)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnGetThread (see page 1340)	A wrapper for the TServerSocketThread (see page 1356) event of the same name.
	OnListening (see page 1341)	Occurs when a server socket begins listening for connections.

23.1.5.6.1 TCustomTcpServer.OnAccept Event

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

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

Write (see page 3789) an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that is used in the server endpoint of the connection. The port number, and possibly the IP address, differs from those of the listening socket.

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

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
```

```
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active

OnConnect (see page 1332)

OnListening

23.1.5.6.2 TCustomTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread (see page 1356) event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread (see page 1356) object referenced in ServerSocketThread (see page 1338).

See Also

TServerSocketThread (see page 1356)

ServerSocketThread (see page 1326)

OnGetThread (see page 1361)

23.1.5.6.3 TCustomTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write (see page 3789) an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 1338)

TNotifyEvent

23.1.6 TIpSocket Class

Encapsulates the network support functions.

Class Hierarchy

Pascal

```
TIpSocket = class(TBaseSocket);
```

C++

```
class TIpSocket : public TBaseSocket;
```

File

Sockets

Description

TIpSocket encapsulates the network support functions, linking sockets to network protocols such as TCP/IP and UDP, as well as

raw sockets and custom protocols.

Do not create instances of TIpSocket. Use TIpSocket as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TIpSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 1302)

23.1.6.1 TIpSocket Members

The following tables list the members exposed by TIpSocket.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Close (see page 1305)	Shuts down the socket connection.
	Create (see page 1343)	Creates an instance of TIpSocket (see page 1341).
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
	Open (see page 1307)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.

	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1315)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.
	SockType (see page 1317)	Sets or returns the socket type.

23.1.6.2 TIpSocket.Create Constructor

Creates an instance of TIpSocket (see page 1341).

Pascal

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

C++

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

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TIpSocket (see page 1341) object.

See Also

Destroy

23.1.6.3 TIpSocket Methods

The methods of the TIpSocket class are listed here.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.

Public Methods

	Name	Description
	GetSocketAddr (see page 1344)	Builds a valid network socket address.

	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	SendTo (see page 1347)	Sends a packet of information.

23.1.6.3.1 TIpSocket.Bind Method

Pascal

```
function Bind: Boolean;
```

C++

```
__fastcall Boolean Bind();
```

Description

This is Bind, a member of class TIpSocket.

23.1.6.3.2 TIpSocket.GetSocketAddr Method

Builds a valid network socket address.

Pascal

```
function GetSocketAddr(h: TSocketHost; p: TSocketPort): TSockAddr;
```

C++

```
__fastcall TSockAddr GetSocketAddr(TSocketHost h, TSocketPort p);
```

Description

Use GetSocketAddr to build a valid socket address based on the host name and port number. The **TSocketHost (see page 1410)** and **TSocketPort (see page 1410)** arguments are used to return a TSockAddr structure, which can then be passed to Connect or Bind (see page 1344).

See Also

Bind (see page 1344)

Connect (see page 1329)

LocalHostAddr (see page 1345)

LookupHostAddr (see page 1345)

RemoteHost (see page 1350)

RemotePort (see page 1352)

23.1.6.3.3 TIpSocket.LocalDomainName Method

Returns the domain name for the local host.

Pascal

```
function LocalDomainName: string;
```

C++

```
__fastcall UnicodeString LocalDomainName();
```

Description

Use LocalDomainName name to return the domain name for the local host.

See Also

LocalHostAddr (see page 1345)

LocalHostName (see page 1345)

23.1.6.3.4 TIpSocket.LocalHostAddr Method

Returns the local IP address for the host machine.

Pascal

```
function LocalHostAddr: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostAddr();
```

Description

Use LocalHostAddr to return the local IP address in string format.

See Also

LocalHost (see page 1348)

LocalDomainName (see page 1344)

LocalHostName (see page 1345)

23.1.6.3.5 TIpSocket.LocalHostName Method

Returns the local host name.

Pascal

```
function LocalHostName: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostName();
```

Description

Use LocalHostName to return the host name for the local host. For example:

```
ShowMessage(TcpClient1.LocalHostName);
```

See Also

LocalHost (see page 1348)

LocalDomainName (see page 1344)

LocalHostAddr (see page 1345)

23.1.6.3.6 TIpSocket.LookupHostAddr Method

Looks up the host address based on the host name.

Pascal

```
function LookupHostAddr(const hn: AnsiString): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostAddr(const AnsiString hn);
```

Description

Use LookupHostAddr to look up the host address based on the machine's host name. For example:

```
StrToInt(TcpClient1.LookupHostAddr("borland.com"));
```

See Also

RemoteHost (see page 1350)

LookupHostName (see page 1346)

LookupPort (see page 1346)

23.1.6.3.7 TIpSocket.LookupHostName Method

Looks up the host name based on the IP address.

Pascal

```
function LookupHostName(const ipaddr: AnsiString): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostName(const AnsiString ipaddr);
```

Description

Use LookupHostName to look up the host name based on the machine's IP address. For example:

```
TcpClient1.LookupHostName("123.60.192.10");
```

See Also

RemoteHost (see page 1350)

LookupHostAddr (see page 1345)

LookupPort (see page 1346)

23.1.6.3.8 TIpSocket.LookupPort Method

Returns the port number.

Pascal

```
function LookupPort(const sn: AnsiString; pn: PAnsiChar = nil): word;
```

C++

```
__fastcall word LookupPort(const AnsiString sn, PAnsiChar pn = nil);
```

Description

Use LookupPort to return the port number of the IP socket. The port number is related to the service name and protocol number.

For more information, refer to the home page of the Internet Assigned (see page 3628) Numbers Authority (www.iana.org), which maintains lists of port numbers and their associated protocols. Also, refer to the documentation for the application's target operating system for information on how port numbers are stored and changed in that OS.

See Also

RemotePort (see page 1352)

LookupProtocol (see page 1347)

23.1.6.3.9 TIpSocket.LookupProtocol Method

Returns the protocol number.

Pascal

```
function LookupProtocol(const pn: AnsiString): TSocketProtocol;
```

C++

```
__fastcall TSocketProtocol LookupProtocol(const AnsiString pn);
```

Description

Use LookupProtocol to return the protocol number being used by the IP socket. The protocol number is based on the protocol name.

For more information, refer to the home page of the Internet Assigned (see page 3628) Numbers Authority (www.iana.org), which maintains lists of protocol numbers and their associated protocols.

See Also

RemotePort (see page 1352)

LookupPort (see page 1346)

23.1.6.3.10 TIpSocket.ReceiveFrom Method

Receives a packet of information.

Pascal

```
function ReceiveFrom(var buf; bufsize: Integer; ToAddr: TSockAddr; var len: Integer; flags:
Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveFrom( buf, int bufsize, TSockAddr ToAddr, int len, int flags = 0);
```

Description

Use ReceiveFrom to receive a single packet of information. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

SendTo (see page 1347)

OnReceive (see page 1320)

23.1.6.3.11 TIpSocket.SendTo Method

Sends a packet of information.

Pascal

```
function SendTo(var buf; bufsize: Integer; ToAddr: TSockAddr; flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendTo( buf, int bufsize, TSockAddr ToAddr, int flags = 0);
```

Description

Use SendTo to send a single packet of information over the network. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

ReceiveFrom (see page 1347)

OnSend (see page 1321)

23.1.6.4 TIpSocket Properties

The properties of the TIpSocket class are listed here.

Public Properties

	Name	Description
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.

23.1.6.4.1 TIpSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 1307)

TSocketDomain (see page 1404)

23.1.6.4.2 TIpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr (see page 1345) function, to host multiple connections simultaneously. Each combination of LocalPort (see page 1349) and LocalHostAddr (see page 1345) can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr (see page 1345) function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 1349)

RemoteHost (see page 1350)

LocalHostAddr (see page 1345)

LookupHostName (see page 1346)

LookupPort (see page 1346)

23.1.6.4.3 TIpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
```

```
    //disconnecting the server
    TcpServer1.Active := false;
  end;

  procedure TForm3.Button3Click(Sender: TObject);
  begin
    //clearing the memo
    Memo1.Lines.Clear;
  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Memo1.Lines.Clear;

    //server initialization
    TcpServer1 := TTcpServer.Create(Self);
    TcpServer1.OnAccept := TCpServer1Accept;
    TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
    TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
    TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
  end;

  procedure TForm3.TcpServer1Accept(Sender: TObject;
    ClientSocket: TCustomIpClient);
  var
    a : array[0..9] of byte;
    I : Cardinal;
  begin

    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

LocalHost (see page 1348)

RemotePort (see page 1352)

LookupPort (see page 1346)

23.1.6.4.4 TIpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemotePort (see page 1352)

LocalHostAddr (see page 1345)

LocalHostName (see page 1345)

LookupHostAddr (see page 1345)

LookupHostName (see page 1346)

23.1.6.4.5 TIpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost (see page 1350) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost (see page 1350) can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol (see page 1347) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
```

```
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemoteHost (see page 1350)

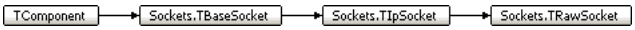
LookupPort (see page 1346)

LookupProtocol (see page 1347)

23.1.7 TRawSocket Class

Encapsulates a raw socket.

Class Hierarchy



Pascal

```
TRawSocket = class(TIpSocket);
```

C++

```
class TRawSocket : public TIpSocket;
```

File

Sockets

Description

TRawSocket encapsulates a raw socket.

See Also

TBaseSocket (see page 1302)

23.1.7.1 TRawSocket Members

The following tables list the members exposed by TRawSocket.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1317)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1319)	Occurs when a socket is deactivated.
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Close (see page 1305)	Shuts down the socket connection.
	Create (see page 1355)	Creates an instance of a TRawSocket (see page 1353) object.
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
	Open (see page 1307)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1314)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1315)	Sets or returns the blocking mode of the socket.

R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.
	SockType (see page 1355)	Sets or returns the socket type.

23.1.7.2 TRawSocket.Create Constructor

Creates an instance of a TRawSocket (see page 1353) object.

Pascal

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

C++

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

Description

Call Create indirectly, using the **new** keyword, to instantiate a TRawSocket (see page 1353) object.

See Also

TRawSocket (see page 1353)

Destroy

23.1.7.3 TRawSocket Properties

The properties of the TRawSocket class are listed here.

Public Properties

	Name	Description
	SockType (see page 1355)	Sets or returns the socket type.

23.1.7.3.1 TRawSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 1406) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 1406)

23.1.8 TServerSocketThread Class

Enables creation of separate threads of execution for the server socket in an application.

Class Hierarchy

Pascal

```
TServerSocketThread = class(TThread);
```

C++

```
class TServerSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 1357) TServerSocketThread (or a descendant of the class) to represent an execution thread for the server socket in a multi-threaded application. Each new instance of a TServerSocketThread, a TThread descendant, is a new thread of execution. Using a separate TServerSocketThread allows the server socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TClientSocketThread (see page 1322)

TThread

23.1.8.1 TServerSocketThread Members

The following tables list the members exposed by TServerSocketThread.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 1358)	This is AddClientSocketThread, a member of class TServerSocketThread.
	CreateThread (see page 1359)	This is CreateThread, a member of class TServerSocketThread.
	FetchClientSocketThread (see page 1359)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 1359)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Public Events

	Name	Description
	OnGetThread (see page 1361)	Creates and initializes a client socket thread when a connection is requested.

Public Methods

	Name	Description
	ClearThreadPool (see page 1358)	Terminates all of the threads in the thread pool.
	Create (see page 1357)	Creates an instance of a TServerSocketThread (see page 1356) object.

	Destroy (see page 1357)	Destroys the thread object and releases the memory allocated to it.
	Execute (see page 1359)	Provides the code which executes when the thread is run.

Public Properties

	Name	Description
	ServerSocket (see page 1360)	Specifies the TCustomTcpServer (see page 1332) object that manages all the TClientSocketThread (see page 1322) objects for its client connections.
	ThreadCacheSize (see page 1360)	Indicates the number of TServerSocketThread (see page 1356) objects cached for reuse by new connections to client sockets.
	ThreadPool (see page 1361)	A list of all the client socket threads created by the TServerSocketThread (see page 1356) object.

23.1.8.2 TServerSocketThread.Create Constructor

Creates an instance of a TServerSocketThread (see page 1356) object.

Pascal

```
constructor Create(AServerSocket: TCustomTcpServer);
```

C++

```
__fastcall TServerSocketThread(TCustomTcpServer AServerSocket);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application.

See Also

Destroy (see page 1357)

Execute (see page 1359)

23.1.8.3 TServerSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

Destroy signals the thread to terminate and then waits for the thread to return before calling the parent class's destructor.

See Also

delete

FreeOnTerminate

Create (see page 1357)

23.1.8.4 TServerSocketThread Methods

The methods of the TServerSocketThread class are listed here.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 1358)	This is AddClientSocketThread, a member of class TServerSocketThread.
	CreateThread (see page 1359)	This is CreateThread, a member of class TServerSocketThread.
	FetchClientSocketThread (see page 1359)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 1359)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Public Methods

	Name	Description
	ClearThreadPool (see page 1358)	Terminates all of the threads in the thread pool.
	Execute (see page 1359)	Provides the code which executes when the thread is run.

23.1.8.4.1 TServerSocketThread.AddClientSocketThread Method

Pascal

```
function AddClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread AddClientSocketThread();
```

Description

This is AddClientSocketThread, a member of class TServerSocketThread.

23.1.8.4.2 TServerSocketThread.ClearThreadPool Method

Terminates all of the threads in the thread pool.

Pascal

```
procedure ClearThreadPool;
```

C++

```
__fastcall ClearThreadPool();
```

Description

The thread pool is a list of client socket threads that are available to handle connection requests. This list is kept to allow reuse of old threads and improve performance. It is sometimes useful to destroy all of the thread object instances in this list.

Use ClearThreadPool to terminate all of the threads in the thread pool list. Any suspended thread in the pool is resumed so that termination can proceed.

See Also

ThreadPool (see page 1361)

Terminate

23.1.8.4.3 TServerSocketThread.CreateThread Method

Pascal

```
function CreateThread: TClientSocketThread; virtual;
```

C++

```
virtual __fastcall TClientSocketThread CreateThread();
```

Description

This is CreateThread, a member of class TServerSocketThread.

23.1.8.4.4 TServerSocketThread.Execute Method

Provides the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated

Synchronize

Create (see page 1357)

23.1.8.4.5 TServerSocketThread.FetchClientSocketThread Method

Pascal

```
function FetchClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread FetchClientSocketThread();
```

Description

This is FetchClientSocketThread, a member of class TServerSocketThread.

23.1.8.4.6 TServerSocketThread.RemoveClientSocketThread Method

Pascal

```
procedure RemoveClientSocketThread(ClientSocketThread: TClientSocketThread);
```

C++

```
__fastcall RemoveClientSocketThread(TClientSocketThread ClientSocketThread);
```

Description

This is RemoveClientSocketThread, a member of class TServerSocketThread.

23.1.8.5 TServerSocketThread Properties

The properties of the TServerSocketThread class are listed here.

Public Properties

	Name	Description
	ServerSocket (see page 1360)	Specifies the TCustomTcpServer (see page 1332) object that manages all the TClientSocketThread (see page 1322) objects for its client connections.
	ThreadCacheSize (see page 1360)	Indicates the number of TServerSocketThread (see page 1356) objects cached for reuse by new connections to client sockets.
	ThreadPool (see page 1361)	A list of all the client socket threads created by the TServerSocketThread (see page 1356) object.

23.1.8.5.1 TServerSocketThread.ServerSocket Property

Specifies the TCustomTcpServer (see page 1332) object that manages all the TClientSocketThread (see page 1322) objects for its client connections.

Pascal

```
property ServerSocket: TCustomTcpServer;
```

C++

```
__property TCustomTcpServer ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening socket object that spawned the thread for a client connection it accepted. TServerSocketThread (see page 1356) passes events and errors to the appropriate event handlers assigned to ServerSocket. TServerSocketThread (see page 1356) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

TClientSocketThread (see page 1322)

TServerSocketThread (see page 1356)

23.1.8.5.2 TServerSocketThread.ThreadCacheSize Property

Indicates the number of TServerSocketThread (see page 1356) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends

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

23.1.8.5.3 TServerSocketThread.ThreadPool Property

A list of all the client socket threads created by the TServerSocketThread (see page 1356) object.

Pascal

```
property ThreadPool: TList;
```

C++

```
__property TList ThreadPool;
```

Description

TServerSocketThread (see page 1356) objects listen for requests on the network and delegate them to client socket threads. Each client socket thread is added to the list ThreadPool after it is created. Creating and destroying client socket threads can be quite time consuming, so whenever one completes its work on a given connection it becomes dormant. When a new connection request is received, the ThreadPool is checked for dormant client socket threads. If one is available, it is assigned the connection. If not, a new client socket thread created. By reusing previously-created threads, the application's performance is improved.

See Also

TClientSocketThread (see page 1322)

ClearThreadPool (see page 1358)

23.1.8.6 TServerSocketThread Events

The events of the TServerSocketThread class are listed here.

Public Events

	Name	Description
	OnGetThread (see page 1361)	Creates and initializes a client socket thread when a connection is requested.

23.1.8.6.1 TServerSocketThread.OnGetThread Event

Creates and initializes a client socket thread when a connection is requested.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

An OnGetThread event occurs when a connection is requested. The event handler creates an instance of TClientSocketThread (see page 1322) and initializes it. Write (see page 3789) an event handler on TCustomTcpServer (see page 1332) to create and initialize instances of your own client socket thread class.

See Also

TClientSocketThread (see page 1322)

TCustomTcpServer (see page 1332)

23.1.9 TTcpClient Class

TCP client component.

Class Hierarchy

Pascal

```
TTcpClient = class(TCustomIpClient);
```

C++

```
class TTcpClient : public TCustomIpClient;
```

File

Sockets

Description

Use TTcpClient to create TCP client applications.

TTcpClient publishes properties and events from its ancestor socket components.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;
```

```
procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TCustomIpClient (see page 1326)

TTcpServer (see page 1375)

23.1.9.1 TTcpClient Members

The following tables list the members exposed by TTcpClient.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoConnect (see page 1330)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoDisconnect (see page 1330)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
	Close (see page 1305)	Shuts down the socket connection.
	Close (see page 1329)	Shuts down the socket connection.
	Connect (see page 1329)	Connects the local socket to the remote socket.
	Create (see page 1303)	Creates an instance of TBaseSocket (see page 1302).
	Create (see page 1328)	Creates an instance of TCustomIpClient (see page 1326).
	Create (see page 1343)	Creates an instance of TIpSocket (see page 1341).
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	Disconnect (see page 1329)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	GetThreadObject (see page 1330)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.

	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
V	Open (see page 1307)	Creates the socket connection.
V	Open (see page 1330)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1348)	Specifies the name of the local system.
	LocalPort (see page 1349)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	SockType (see page 1317)	Sets or returns the socket type.

Published Events

	Name	Description
	OnConnect (see page 1370)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1370)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1371)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1373)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1373)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1373)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1374)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 1365)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1366)	Sets or returns the blocking mode of the socket.
	Connected (see page 1366)	Indicates whether the socket connection is open and available for communication with other machines.
	RemoteHost (see page 1367)	Specifies the name of the remote system.
	RemotePort (see page 1368)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

23.1.9.2 TTcpClient Properties

The properties of the TTcpClient class are listed here.

Published Properties

	Name	Description
	Active (see page 1365)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1366)	Sets or returns the blocking mode of the socket.
	Connected (see page 1366)	Indicates whether the socket connection is open and available for communication with other machines.
	RemoteHost (see page 1367)	Specifies the name of the remote system.
	RemotePort (see page 1368)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

23.1.9.2.1 TTcpClient.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);
```

```
    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

Close (see page 1305)

Open (see page 1307)

23.1.9.2.2 TTcpClient.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1409)

OnReceive (see page 1320)

OnSend (see page 1321)

23.1.9.2.3 TTcpClient.Connected Property

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

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open method. To close a socket, use the Closemethod.

See Also

Close (see page 1329)

Open (see page 1330)

OnConnect (see page 1332)

23.1.9.2.4 TTcpClient.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
```

```
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemotePort (see page 1352)

LocalHostAddr (see page 1345)

LocalHostName (see page 1345)

LookupHostAddr (see page 1345)

LookupHostName (see page 1346)

23.1.9.2.5 TTcpClient.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
```

```
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemoteHost (see page 1350)

LookupPort (see page 1346)

LookupProtocol (see page 1347)

23.1.9.3 TTcpClient Events

The events of the TTcpClient class are listed here.

Published Events

	Name	Description
	OnConnect (see page 1370)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1370)	Occurs when the handle to a socket becomes active.

	OnDestroyHandle (see page 1371)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1373)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1373)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1373)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1374)	Occurs just before a client socket writes information to the socket connection.

23.1.9.3.1 TTcpClient.OnConnect Event

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

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

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

See Also

Connected (see page 1331)

OnDisconnect (see page 1332)

23.1.9.3.2 TTcpClient.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
```

```
    //disconnecting the server
    TcpServer1.Active := false;
  end;

  procedure TForm3.Button3Click(Sender: TObject);
  begin
    //clearing the memo
    Memo1.Lines.Clear;
  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Memo1.Lines.Clear;

    //server initialization
    TcpServer1 := TTcpServer.Create(Self);
    TcpServer1.OnAccept := TCpServer1Accept;
    TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
    TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
    TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
  end;

  procedure TForm3.TcpServer1Accept(Sender: TObject;
    ClientSocket: TCustomIpClient);
  var
    a : array[0..9] of byte;
    I : Cardinal;
  begin

    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

Active (see page 1314)

Handle (see page 1316)

23.1.9.3.3 TTcpClient.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;
```

```
procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

23.1.9.3.4 TTcpClient.OnDisconnect Event

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

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 1331)

OnConnect (see page 1332)

23.1.9.3.5 TTcpClient.OnError Event

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

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

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

See Also

ESocketError (see page 1301)

23.1.9.3.6 TTcpClient.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 127)

BytesReceived (see page 1315)

ReceiveBuf (see page 1308)

ReceiveIn (see page 1310)

23.1.9.3.7 TTcpClient.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;
```

```
    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

TWinSocketStream (see page 127)

SendBuf (see page 1310)

SendLn (see page 1311)

23.1.10 TTcpServer Class

TTcpServer is the TCP server component.

Class Hierarchy

Pascal

```
TTcpServer = class(TCustomTcpServer);
```

C++

```
class TTcpServer : public TCustomTcpServer;
```

File

Sockets

Description

Use TTcpServer to create TCP server applications.

TTcpServer publishes properties and events from its ancestor socket components.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;
```

```
{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

TCustomTcpServer

TTcpClient

23.1.10.1 TTcpServer Members

The following tables list the members exposed by TTcpServer.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoAccept (see page 1336)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Listen (see page 1337)	This is Listen, a member of class TCustomTcpServer.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnError (see page 1320)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1320)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1321)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Accept (see page 1335)	This is the overview for the Accept method overload.
	Close (see page 1305)	Shuts down the socket connection.
	Close (see page 1336)	Shuts down the socket connection.
	Create (see page 1303)	Creates an instance of TBaseSocket (see page 1302).
	Create (see page 1334)	Creates an instance of TCustomTcpServer (see page 1332).
	Create (see page 1343)	Creates an instance of TIpSocket (see page 1341).
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	Destroy (see page 1335)	Frees the memory associated with the TCustomTcpServer (see page 1332) object.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.
	Open (see page 1307)	Creates the socket connection.
	Open (see page 1337)	Opens the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	ReceiveIn (see page 1310)	Reads delimited lines of data from the socket.

	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	SendLn (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForConnection (see page 1337)	Listens for TCP connections.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 1338)	Sets the server application to listen for client requests.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1350)	Specifies the name of the remote system.
	RemotePort (see page 1352)	Specifies the ID number of the port used by the RemoteHost (see page 1350) to form the socket connection.
	ServerSocketThread (see page 1338)	Specifies the TServerSocketThread (see page 1356) in use.
	SockType (see page 1317)	Sets or returns the socket type.

Published Events

	Name	Description
	OnAccept (see page 1382)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1384)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1385)	Occurs when a socket is deactivated.
	OnGetThread (see page 1387)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 1387)	Occurs when a server socket begins listening for connections.

Published Properties

	Name	Description
	Active (see page 1379)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1380)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1380)	Specifies the name of the local system.
	LocalPort (see page 1381)	Specifies the ID number of the port used by the socket connection.

23.1.10.2 TTcpServer Properties

The properties of the TTcpServer class are listed here.

Published Properties

	Name	Description
	Active (see page 1379)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1380)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1380)	Specifies the name of the local system.
	LocalPort (see page 1381)	Specifies the ID number of the port used by the socket connection.

23.1.10.2.1 TTcpServer.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
```

```
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

Close (see page 1305)

Open (see page 1307)

23.1.10.2.2 TTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 1320)

OnSend (see page 1321)

TServerSocketBlockMode (see page 1404)

23.1.10.2.3 TTcpServer.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 1349)

RemoteHost (see page 1350)

LocalHostAddr (see page 1345)

LookupHostName (see page 1346)

LookupPort (see page 1346)

23.1.10.2.4 TTcpServer.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
```

```
    TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
    TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
  end;

  procedure TForm3.TcpServer1Accept(Sender: TObject;
    ClientSocket: TCustomIpClient);
  var
    a : array[0..9] of byte;
    I : Cardinal;
  begin

    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

LocalHost (see page 1348)

RemotePort (see page 1352)

LookupPort (see page 1346)

23.1.10.3 TTcpServer Events

The events of the TTcpServer class are listed here.

Published Events

	Name	Description
	OnAccept (see page 1382)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1384)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1385)	Occurs when a socket is deactivated.
	OnGetThread (see page 1387)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 1387)	Occurs when a server socket begins listening for connections.

23.1.10.3.1 TTcpServer.OnAccept Event

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

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

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

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

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
```

```
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

Active

OnConnect (see page 1332)

OnListening

23.1.10.3.2 TTcpServer.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;
```

```
procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1314)

Handle (see page 1316)

23.1.10.3.3 TTcpServer.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
```

```
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

23.1.10.3.4 TTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread object referenced in ServerSocketThread.

See Also

TServerSocketThread (see page 1356)

ServerSocketThread (see page 1326)

OnGetThread (see page 1361)

23.1.10.3.5 TTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 1338)

TNotifyEvent

23.1.11 TUdpSocket Class

UDP component.

Class Hierarchy

Pascal

```
TUdpSocket = class(TCustomIpClient);
```

C++

```
class TUdpSocket : public TCustomIpClient;
```

File

Sockets

Description

Use TUdpSocket to create UDP-based applications. Add a TUdpSocket object to a form or data module to turn an application into a UDP/IP client and server. TUdpSocket specifies a desired connection to a UDP/IP server, manages the connection when it is open, and terminates the connection when the application is through. It also listens for requests for UDP/IP connections from other machines and establishes connections when requests are received.

23.1.11.1 TUdpSocket Members

The following tables list the members exposed by TUdpSocket.

Protected Methods

	Name	Description
	Bind (see page 1344)	This is Bind, a member of class TIpSocket.
	DoConnect (see page 1330)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1305)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1305)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoDisconnect (see page 1330)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 1305)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1306)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1306)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1306)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1306)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1312)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1313)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
	Close (see page 1305)	Shuts down the socket connection.
	Close (see page 1329)	Shuts down the socket connection.
	Connect (see page 1329)	Connects the local socket to the remote socket.
	Create (see page 1389)	Creates an instance of TUdpSocket (see page 1387).
	Destroy (see page 1304)	Frees the memory associated with the TBaseSocket (see page 1302) object.
	Disconnect (see page 1329)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1344)	Builds a valid network socket address.
	GetThreadObject (see page 1330)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1344)	Returns the domain name for the local host.
	LocalHostAddr (see page 1345)	Returns the local IP address for the host machine.
	LocalHostName (see page 1345)	Returns the local host name.
	LookupHostAddr (see page 1345)	Looks up the host address based on the host name.
	LookupHostName (see page 1346)	Looks up the host name based on the IP address.
	LookupPort (see page 1346)	Returns the port number.
	LookupProtocol (see page 1347)	Returns the protocol number.
	MapDomain (see page 1307)	Maps the domain.
	MapSockType (see page 1307)	Maps the socket type.

	Open (see page 1307)	Creates the socket connection.
	Open (see page 1330)	Creates the socket connection.
	PeekBuf (see page 1308)	Reads the buffer.
	ReceiveBuf (see page 1308)	Reads data buffers of to the socket.
	ReceiveFrom (see page 1347)	Receives a packet of information.
	Receiveln (see page 1310)	Reads delimited lines of data from the socket.
	Select (see page 1310)	Tests for socket activity.
	SendBuf (see page 1310)	Writes data buffers of size bufsize from the socket.
	Sendln (see page 1311)	Writes delimited lines of data to the socket.
	SendStream (see page 1311)	Writes a stream of data to the socket.
	SendTo (see page 1347)	Sends a packet of information.
	WaitForData (see page 1313)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	BytesReceived (see page 1315)	Returns the number of bytes received by the socket.
	BytesSent (see page 1315)	Returns the number of bytes sent from the socket.
	Connected (see page 1331)	Indicates whether the socket connection is open and available for communication with other machines.
	Domain (see page 1316)	Sets or returns the protocol family used by the socket.
	Domain (see page 1348)	Sets or returns the protocol family used by the socket.
	Handle (see page 1316)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1316)	Sets or returns the protocol used by the socket.
	SockType (see page 1317)	Sets or returns the socket type.

Published Events

	Name	Description
	OnConnect (see page 1396)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1397)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1398)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1399)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1400)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1400)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1401)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 1390)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1391)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1392)	Specifies the name of the local system.
	LocalPort (see page 1392)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1394)	Specifies the name of the remote system.
	RemotePort (see page 1395)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

23.1.11.2 TUdpSocket.Create Constructor

Creates an instance of TUdpSocket (see page 1387).

Pascal

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

C++

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

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TUdpSocket (see page 1387) object.

23.1.11.3 TUdpSocket Properties

The properties of the TUdpSocket class are listed here.

Published Properties

	Name	Description
	Active (see page 1390)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1391)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1392)	Specifies the name of the local system.
	LocalPort (see page 1392)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1394)	Specifies the name of the remote system.
	RemotePort (see page 1395)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

23.1.11.3.1 TUdpSocket.Active Property

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

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

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

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
```

```
      myStream.WriteBuffer(I,1);
    end;
    userValue := StrToIntDef(Edit1.Text, 0);
    myStream.WriteBuffer(userValue,1);

    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

Close (see page 1305)

Open (see page 1307)

23.1.11.3.2 TUdpSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1409)

OnReceive (see page 1320)

OnSend (see page 1321)

23.1.11.3.3 TUdpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 1349)

RemoteHost (see page 1350)

LocalHostAddr (see page 1345)

LookupHostName (see page 1346)

LookupPort (see page 1346)

23.1.11.3.4 TUdpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;
```

```
{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

LocalHost (see page 1348)

RemotePort (see page 1352)

LookupPort (see page 1346)

23.1.11.3.5 TUdpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
```

```
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemotePort (see page 1352)

LocalHostAddr (see page 1345)

LocalHostName (see page 1345)

LookupHostAddr (see page 1345)

LookupHostName (see page 1346)

23.1.11.3.6 TUdpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);
```

```
    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemoteHost (see page 1350)

LookupPort (see page 1346)

LookupProtocol (see page 1347)

23.1.11.4 TUdpSocket Events

The events of the TUdpSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 1396)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1397)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1398)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1399)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1400)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1400)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1401)	Occurs just before a client socket writes information to the socket connection.

23.1.11.4.1 TUdpSocket.OnConnect Event

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

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

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

See Also

Connected (see page 1331)

OnDisconnect (see page 1332)

23.1.11.4.2 TUdpSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
```

```
    TcpServer1.OnAccept := TCpServer1Accept;
    TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
    TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
    TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
  end;

  procedure TForm3.TcpServer1Accept(Sender: TObject;
    ClientSocket: TCustomIpClient);
  var
    a : array[0..9] of byte;
    I : Cardinal;
  begin

    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

Active (see page 1314)

Handle (see page 1316)

23.1.11.4.3 TUdpSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;
```

```
{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

23.1.11.4.4 TUdpSocket.OnDisconnect Event

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

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 1331)

OnConnect (see page 1332)

23.1.11.4.5 TUdpSocket.OnError Event

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

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

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

See Also

ESocketError (see page 1301)

23.1.11.4.6 TUdpSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 127)

BytesReceived (see page 1315)

ReceiveBuf (see page 1308)

ReceiveLn (see page 1310)

23.1.11.4.7 TUdpSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
```

```
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TWinSocketStream (see page 127)

SendBuf (see page 1310)

Sendln (see page 1311)

23.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TServerSocketBlockMode (see page 1404)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 1404)	Set of possible protocol family types used by a socket.
	TSocketType (see page 1406)	The TSocketType type defines the set of possible socket types.

Records

	Name	Description
	TIPHeader (see page 1403)	This is record Sockets.TIPHeader.

23.2.1 Sockets.TIPHeader Record

Pascal

```
TIPHeader = packed record
  iph_verlen: byte;
  iph_tos: byte;
  iph_length: word;
  iph_id: word;
  iph_offset: word;
  iph_ttl: byte;
  iph_protocol: byte;
  iph_xsum: word;
  iph_src: longword;
  iph_dest: longword;
end;
```

C++

```
struct TIPHeader {
  byte iph_verlen;
  byte iph_tos;
  word iph_length;
  word iph_id;
  word iph_offset;
  byte iph_ttl;
  byte iph_protocol;
  word iph_xsum;
  longword iph_src;
  longword iph_dest;
};
```

File

Sockets

Description

This is record Sockets.TIPHeader.

23.2.2 Sockets.TServerSocketBlockMode Enumeration

Defines the set of possible socket flags for block mode.

Pascal

```
TServerSocketBlockMode = (
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
);
```

C++

```
enum TServerSocketBlockMode {
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
};
```

File

Sockets

Description

The following table lists the possible values for the TServerSocketBlockMode type and the meaning corresponding to each value:

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.
bmThreadBlocking	A separate thread is created so that listening can be performed in the background without blocking the entire server application.

23.2.3 Sockets.TSocketDomain Enumeration

Set of possible protocol family types used by a socket.

Pascal

```
TSocketDomain = (
  pfUnspec,
  pfUnix,
  pfInet,
  pfImpLink,
  pfPup,
  pfChaos,
  pfIpx,
  pfNs,
  pfIso,
  pfOsi,
  pfEcma,
  pfDataKit,
  pfCcitt,
  pfSna,
  pfDecNet,
```

```
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  );
```

C++

```
  enum TSocketDomain {
    pfUnspec,
    pfUnix,
    pfInet,
    pfImpLink,
    pfPup,
    pfChaos,
    pfIpx,
    pfNs,
    pfIso,
    pfOsi,
    pfEcma,
    pfDataKit,
    pfCcitt,
    pfSna,
    pfDecNet,
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  };
```

File

Sockets

Description

The following table lists the possible values for the TSocketDomain type and the meaning corresponding to each value:

Value	Description
pfAppleTalk	AppleTalk DDP
pfBan	Banyan Vines
pfCcitt	CCIT (International Telegraph and Telephone Consultative)
pfChaos	
pfDataKit	DataKit
pfDecNet	Reserved for DECnet project
pfDli	
pfEcma	EcmaScript
pfFireFox	FireFox
pfHylink	Hyperlink
pfImpLink	Imported link

pfInet	IP protocol family
pfIpx	Novell Interner Protocol
pfIso	International Standards Organization (ISO)
pfLat	
pfMax	Maximum protocol family number
pfNs	Network Services Protocol (used with DECnet)
pfOsi	Open Systems Interconnection (OSI)
pfPup	PUP protocol
pfSna	Linux SNA project
pfUnix	Local to host (old BSD name for local)
pfUnknown1	Unknown
pfUnspec	Unspecified
pfVoiceView	VoiceView

23.2.4 Sockets.TSocketType Enumeration

The TSocketType type defines the set of possible socket types.

Pascal

```
TSocketType = (
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
);
```

C++

```
enum TSocketType {
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
};
```

File

Sockets

Description

The following table lists the possible values for the TSocketType type and the meaning corresponding to each value:

Value	Description
stDgram	Single datagram packets, not used by IP. Non-reliable.
stRaw	The socket sends and receives raw datagram packets. Used by IP.
stRdm	Reliably delivered message.
stSeqPacket	Sequenced packet stream sends and receives any type of IP packet.

stStream	Reliable streaming data packets. Used by TCP.

23.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetThreadEvent (see page 1408)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 1408)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 1409)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 1409)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 1409)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 1410)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 1410)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 1410)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 1411)	This is type Sockets.TSocketProtocol.

23.3.1 Sockets.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; var ClientSocketThread: TClientSocketThread) of object;
```

C++

```
(Sender: TObject; var ClientSocketThread: TClientSocketThread) ( TGetThreadEvent)();
```

File

Sockets

Description

This is type Sockets.TGetThreadEvent.

23.3.2 Sockets.TSocketAcceptEvent Type

Pascal

```
TSocketAcceptEvent = procedure (Sender: TObject; ClientSocket: TCustomIpClient) of object;
```

C++

```
(Sender: TObject; ClientSocket: TCustomIpClient) ( TSocketAcceptEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketAcceptEvent.

23.3.3 Sockets.TSocketBlockMode Type

Defines the set of possible flags for blocking mode of the socket.

Pascal

```
TSocketBlockMode = bmBlocking..bmNonBlocking;
```

C++

```
bmBlocking..bmNonBlocking TSocketBlockMode;
```

File

Sockets

Description

The following table describes TSocketBlockMode values.

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.

23.3.4 Sockets.TSocketDataEvent Type

Pascal

```
TSocketDataEvent = procedure (Sender: TObject; Buf: pchar; var DataLen: Integer) of object;
```

C++

```
(Sender: TObject; Buf: pchar; var DataLen: Integer) ( TSocketDataEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketDataEvent.

23.3.5 Sockets.TSocketErrorEvent Type

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; SocketError: Integer) of object;
```

C++

```
(Sender: TObject; SocketError: Integer) ( TSocketErrorEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketErrorEvent.

23.3.6 Sockets.TSocketHost Type

Pascal

```
TSocketHost = type string;
```

C++

```
type string TSocketHost;
```

File

Sockets

Description

This is type Sockets.TSocketHost.

23.3.7 Sockets.TSocketNotifyEvent Type

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject) of object;
```

C++

```
(Sender: TObject) ( TSocketNotifyEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketNotifyEvent.

23.3.8 Sockets.TSocketPort Type

Pascal

```
TSocketPort = type string;
```

C++

```
type string TSocketPort;
```

File

Sockets

Description

This is type Sockets.TSocketPort.

23.3.9 Sockets.TSocketProtocol Type

Pascal

```
TSocketProtocol = Word;
```

C++

```
Word TSocketProtocol;
```

File

Sockets

Description

This is type Sockets.TSocketProtocol.

23.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CRLF (see page 1412)	This is constant Sockets.CRLF.

23.4.1 Sockets.CRLF Constant

Pascal

```
CRLF = #13#10;
```

C++

```
CRLF = #13#10;
```

File

Sockets

Description

This is constant Sockets.CRLF.

24 SqlExpr Namespace

This is namespace SqlExpr.

Topics

Name	Description
Classes (see page 1416)	The following table lists classes in this documentation.
Functions (see page 1620)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1624)	The following table lists structs, records, enums in this documentation.
Types (see page 1633)	The following table lists types in this documentation.
Variables (see page 1643)	The following table lists variables in this documentation.
Constants (see page 1644)	The following table lists constants in this documentation.

Classes

	Name	Description
	TConnectionData (see page 1416)	This is class SqlExpr.TConnectionData.
	TCustomSQLDataSet (see page 1421)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 1471)	This is class SqlExpr.TFLDDesc.
	TISQLCommand (see page 1481)	This is class SqlExpr.TISQLCommand.
	TISQLConnection (see page 1486)	This is class SqlExpr.TISQLConnection.
	TISQLCursor (see page 1492)	This is class SqlExpr.TISQLCursor.
	TISQLMetaData (see page 1502)	This is class SqlExpr.TISQLMetaData.
	TSQLBlobStream (see page 1507)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 1510)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 1553)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 1556)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 1567)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 1577)	TSQLQuery represents a query that is executed using dbExpress.
	TSqlServerMethod (see page 1589)	TDataSet used to call DataSnap server methods.
	TSQLStoredProc (see page 1594)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 1604)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 1618)	This is class SqlExpr.TTransactionItem.

Constants

Name	Description
BlobTypeMap (see page 1644)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 1645)	This is constant SqlExpr.DataTypeMap.
DefaultCursor (see page 1645)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 1646)	This is constant SqlExpr.DefaultMaxBlobSize.
DefaultRowsetSize (see page 1646)	This is constant SqlExpr.DefaultRowsetSize.
eSQLAutoIncr (see page 1646)	This is constant SqlExpr.eSQLAutoIncr.
eSQLDefault (see page 1647)	This is constant SqlExpr.eSQLDefault.
eSQLFunction (see page 1647)	This is constant SqlExpr.eSQLFunction.
eSQLLocal (see page 1647)	This is constant SqlExpr.eSQLLocal.
eSQLNonUnique (see page 1648)	This is constant SqlExpr.eSQLNonUnique.
eSQLPackage (see page 1648)	This is constant SqlExpr.eSQLPackage.
eSQLPrimaryKey (see page 1648)	This is constant SqlExpr.eSQLPrimaryKey.
eSQLProcedure (see page 1649)	This is constant SqlExpr.eSQLProcedure.
eSQLRowId (see page 1649)	This is constant SqlExpr.eSQLRowId.
eSQLRowVersion (see page 1649)	This is constant SqlExpr.eSQLRowVersion.
eSQLSynonym (see page 1650)	This is constant SqlExpr.eSQLSynonym.
eSQLSysProcedure (see page 1650)	This is constant SqlExpr.eSQLSysProcedure.
eSQLSystemTable (see page 1650)	This is constant SqlExpr.eSQLSystemTable.
eSQLTable (see page 1651)	This is constant SqlExpr.eSQLTable.
eSQLTempTable (see page 1651)	This is constant SqlExpr.eSQLTempTable.
eSQLUnique (see page 1651)	This is constant SqlExpr.eSQLUnique.
eSQLView (see page 1652)	This is constant SqlExpr.eSQLView.
FldSubTypeMap (see page 1652)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 1652)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 1653)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 1653)	This is constant SqlExpr.SAnd.
SOrderBy (see page 1653)	This is constant SqlExpr.SOrderBy.
SParam (see page 1654)	This is constant SqlExpr.SParam.
SSelect (see page 1654)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 1654)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 1655)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 1655)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 1655)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 1656)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 1656)	This is constant SqlExpr.TErrorMessageSize.

Functions

	Name	Description
	FreeProcParams (see page 1620)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 1620)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 1621)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDelegateDriverNames (see page 1621)	This is function SqlExpr.GetDelegateDriverNames.
	GetDriverNames (see page 1621)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 1622)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 1622)	Fills a TParams object with information from a list of parameter descriptors.
	RegisterDbXpressLib (see page 1623)	This is function SqlExpr.RegisterDbXpressLib.

Structs, Records, Enums

	Name	Description
	EConnectFlag (see page 1624)	This is record SqlExpr.EConnectFlag.
	SPParamDesc (see page 1625)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.

	SQLTRACEDesc25 (see page 1626)	This is record SqlExpr.SQLTRACEDesc25.
	SQLTRACEDesc30 (see page 1626)	This is record SqlExpr.SQLTRACEDesc30.
	TConnectionState (see page 1627)	TConnectionState describes the state of a TSQLConnection (see page 1510) component.
	TConnectionUserType (see page 1627)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 1628)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 1629)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TSQLSchemaInfo (see page 1630)	This is record SqlExpr.TSQLSchemaInfo.
	TTableScope (see page 1630)	TTableScopes (see page 1640) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransactionDesc (see page 1631)	This is record SqlExpr.TTransactionDesc.
	TTransIsolationLevel (see page 1631)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Types

Name	Description
PSPParamDesc (see page 1633)	This is type SqlExpr.PSPParamDesc.
pSQLTRACEDesc (see page 1634)	This is type SqlExpr.pSQLTRACEDesc.
pSQLTRACEDesc25 (see page 1634)	This is type SqlExpr.pSQLTRACEDesc25.
pSQLTRACEDesc30 (see page 1634)	This is type SqlExpr.pSQLTRACEDesc30.
SQLResult (see page 1635)	SQLResult is the return type for dbExpress function calls.
SQLSPParamDesc (see page 1636)	This is type SqlExpr.SQLSPParamDesc.
SQLTRACEDesc (see page 1636)	This is type SqlExpr.SQLTRACEDesc.
TConnectChangeEvent (see page 1636)	This is type SqlExpr.TConnectChangeEvent.
TFieldDescList (see page 1637)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 1637)	This is type SqlExpr.TFieldList.
TFLDDescRef (see page 1637)	This is type SqlExpr.TFLDDescRef.
TGetDriverFunc (see page 1638)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 1638)	This is type SqlExpr.TLocale.
TLocaleCode (see page 1638)	This is type SqlExpr.TLocaleCode.
TParseInsertSqlEvent (see page 1639)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 1639)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 1639)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 1640)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 1640)	This is type SqlExpr.TTraceEvent.
TTraceEvent25 (see page 1640)	This is type SqlExpr.TTraceEvent25.
TTraceEvent30 (see page 1641)	This is type SqlExpr.TTraceEvent30.
TTraceLogEvent (see page 1641)	This is type SqlExpr.TTraceLogEvent.
TTraceLogEvent25 (see page 1641)	This is type SqlExpr.TTraceLogEvent25.
TTraceLogEvent30 (see page 1642)	This is type SqlExpr.TTraceLogEvent30.

Variables

Name	Description
DllGetClassObject (see page 1643)	This is variable SqlExpr.DllGetClassObject.

24.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TConnectionData (see page 1416)	This is class SqlExpr.TConnectionData.
	TCustomSQLDataSet (see page 1421)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 1471)	This is class SqlExpr.TFLDDesc.
	TISQLCommand (see page 1481)	This is class SqlExpr.TISQLCommand.
	TISQLConnection (see page 1486)	This is class SqlExpr.TISQLConnection.
	TISQLCursor (see page 1492)	This is class SqlExpr.TISQLCursor.
	TISQLMetaData (see page 1502)	This is class SqlExpr.TISQLMetaData.
	TSQLBlobStream (see page 1507)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 1510)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 1553)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 1556)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 1567)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 1577)	TSQLQuery represents a query that is executed using dbExpress.
	TSqlServerMethod (see page 1589)	TDataSet used to call DataSnap server methods.
	TSQLStoredProc (see page 1594)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 1604)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 1618)	This is class SqlExpr.TTransactionItem.

24.1.1 TConnectionData Class

Class Hierarchy

Pascal

```
TConnectionData = class(TPersistent);
```

C++

```
class TConnectionData : public TPersistent;
```

File

SqlExpr

Description

This is class SqlExpr.TConnectionData.

24.1.1.1 TConnectionData Members

The following tables list the members exposed by TConnectionData.

Protected Methods

	Name	Description
	GetFullParams (see page 1419)	This is GetFullParams, a member of class TConnectionData.

Public Methods

	Name	Description
	AddProperties (see page 1418)	This is AddProperties, a member of class TConnectionData.
	BeginUpdate (see page 1419)	This is BeginUpdate, a member of class TConnectionData.
	Create (see page 1417)	This is the overview for the Create constructor overload.
V	Destroy (see page 1418)	Destroys the TPersistent instance and frees its memory.
	EndUpdate (see page 1419)	This is EndUpdate, a member of class TConnectionData.
	RefreshProperties (see page 1419)	This is RefreshProperties, a member of class TConnectionData.
	ReloadProperties (see page 1420)	This is ReloadProperties, a member of class TConnectionData.
	UpdateProperties (see page 1420)	This is UpdateProperties, a member of class TConnectionData.

Published Properties

	Name	Description
	ConnectionName (see page 1420)	This is ConnectionName, a member of class TConnectionData.
	DelegateConnection (see page 1420)	This is DelegateConnection, a member of class TConnectionData.
	DriverName (see page 1421)	This is DriverName, a member of class TConnectionData.
R	IsModified (see page 1421)	This is IsModified, a member of class TConnectionData.
	Properties (see page 1421)	This is Properties, a member of class TConnectionData.

24.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TConnectionData.Create (TConnectionData) (see page 1417)	This is Create, a member of class TConnectionData.
	TConnectionData.Create (TSQLConnection) (see page 1418)	This is Create, a member of class TConnectionData.

24.1.1.2.1 TConnectionData.Create Constructor (TConnectionData)

Pascal

```
constructor Create(AParentData: TConnectionData); overload;
```

C++

```
__fastcall TConnectionData(TConnectionData AParentData);
```

Description

This is Create, a member of class TConnectionData.

24.1.1.2.2 TConnectionData.Create Constructor (TSQLConnection)

Pascal

```
constructor Create(AConnection: TSQLConnection); overload;
```

C++

```
__fastcall TConnectionData(TSQLConnection AConnection);
```

Description

This is Create, a member of class TConnectionData.

24.1.1.3 TConnectionData.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConnectionData();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

System::TObject::Destroy (see page 3366)

24.1.1.4 TConnectionData Methods

The methods of the TConnectionData class are listed here.

Protected Methods

	Name	Description
	GetFullParams (see page 1419)	This is GetFullParams, a member of class TConnectionData.

Public Methods

	Name	Description
	AddProperties (see page 1418)	This is AddProperties, a member of class TConnectionData.
	BeginUpdate (see page 1419)	This is BeginUpdate, a member of class TConnectionData.
	EndUpdate (see page 1419)	This is EndUpdate, a member of class TConnectionData.
	RefreshProperties (see page 1419)	This is RefreshProperties, a member of class TConnectionData.
	ReloadProperties (see page 1420)	This is ReloadProperties, a member of class TConnectionData.
	UpdateProperties (see page 1420)	This is UpdateProperties, a member of class TConnectionData.

24.1.1.4.1 TConnectionData.AddProperties Method

Pascal

```
procedure AddProperties(NewProperties: TStrings);
```

C++

```
__fastcall AddProperties(TStrings NewProperties);
```

Description

This is AddProperties, a member of class TConnectionData.

24.1.1.4.2 TConnectionData.BeginUpdate Method

Pascal

```
function BeginUpdate: Boolean;
```

C++

```
__fastcall Boolean BeginUpdate();
```

Description

This is BeginUpdate, a member of class TConnectionData.

24.1.1.4.3 TConnectionData.EndUpdate Method

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

This is EndUpdate, a member of class TConnectionData.

24.1.1.4.4 TConnectionData.GetFullParams Method

Pascal

```
procedure GetFullParams(Params: TStrings);
```

C++

```
__fastcall GetFullParams(TStrings Params);
```

Description

This is GetFullParams, a member of class TConnectionData.

24.1.1.4.5 TConnectionData.RefreshProperties Method

Pascal

```
procedure RefreshProperties;
```

C++

```
__fastcall RefreshProperties();
```

Description

This is RefreshProperties, a member of class TConnectionData.

24.1.1.4.6 TConnectionData.ReloadProperties Method

Pascal

```
procedure ReloadProperties;
```

C++

```
__fastcall ReloadProperties();
```

Description

This is ReloadProperties, a member of class TConnectionData.

24.1.1.4.7 TConnectionData.UpdateProperties Method

Pascal

```
procedure UpdateProperties(NewProperties: TStrings);
```

C++

```
__fastcall UpdateProperties(TStrings NewProperties);
```

Description

This is UpdateProperties, a member of class TConnectionData.

24.1.1.5 TConnectionData Properties

The properties of the TConnectionData class are listed here.

Published Properties

	Name	Description
	ConnectionName (see page 1420)	This is ConnectionName, a member of class TConnectionData.
	DelegateConnection (see page 1420)	This is DelegateConnection, a member of class TConnectionData.
	DriverName (see page 1421)	This is DriverName, a member of class TConnectionData.
R	IsModified (see page 1421)	This is IsModified, a member of class TConnectionData.
	Properties (see page 1421)	This is Properties, a member of class TConnectionData.

24.1.1.5.1 TConnectionData.ConnectionName Property

Pascal

```
property ConnectionName: string;
```

C++

```
__property UnicodeString ConnectionName;
```

Description

This is ConnectionName, a member of class TConnectionData.

24.1.1.5.2 TConnectionData.DelegateConnection Property

Pascal

```
property DelegateConnection: TConnectionData;
```

C++

```
__property TConnectionData DelegateConnection;
```

Description

This is DelegateConnection, a member of class TConnectionData.

24.1.1.5.3 TConnectionData.DriverName Property

Pascal

```
property DriverName: string;
```

C++

```
__property UnicodeString DriverName;
```

Description

This is DriverName, a member of class TConnectionData.

24.1.1.5.4 TConnectionData.IsModified Property

Pascal

```
property IsModified: Boolean;
```

C++

```
__property Boolean IsModified;
```

Description

This is IsModified, a member of class TConnectionData.

24.1.1.5.5 TConnectionData.Properties Property

Pascal

```
property Properties: TDBXProperties;
```

C++

```
__property TDBXProperties Properties;
```

Description

This is Properties, a member of class TConnectionData.

24.1.2 TCustomSQLDataSet Class

TCustomSQLDataSet is the base class for all dbExpress dataset components.

Class Hierarchy

Pascal

```
TCustomSQLDataSet = class(TWideDataSet);
```

C++

```
class TCustomSQLDataSet : public TWideDataSet;
```

File

SqlExpr

Description

TCustomSQLDataSet encapsulates the properties, events, and methods for working with data accessed through dbExpress. Applications should not instantiate TCustomSQLDataSet objects. Instead, instantiate one of the descendant classes: TSQLDataSet (see page 1556), TSQLTable (see page 1604), TSQLQuery (see page 1577), or TSQLStoredProc (see page 1594).

TCustomSQLDataSet and its descendants represent unidirectional datasets. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, they have the following limitations:

There is no searching or navigation support except for moving to the first record or moving to the next record.

There is no built-in editing support. The data underlying an SQL dataset can only be edited by explicitly creating an SQL UPDATE command or by connecting the SQL dataset to a client dataset using a provider.

There is no support for filtering records or for lookup fields.

See Also

TSQLDataSet (see page 1556)

TSQLQuery (see page 1577)

TSQLStoredProc (see page 1594)

TSQLTable (see page 1604)

TSQLConnection (see page 1510)

TClientDataSEt

TDataSEtProvider

24.1.2.1 TCustomSQLDataSet Members

The following tables list the members exposed by TCustomSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.

	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.

	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1457)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	MaxBlobSize (see page 1460)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1462)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	Params (see page 1463)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1425)	Creates an instance of a TCustomSQLDataSet (see page 1421) component.
V	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	Destroy (see page 1426)	Destroys the TCustomSQLDataSet (see page 1421) instance.
V	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
V	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
V	PrepareStatement (see page 1440)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1467)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.

	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1466)	Specifies the name of the database schema.

24.1.2.2 TCustomSQLDataSet.Create Constructor

Creates an instance of a TCustomSQLDataSet (see page 1421) component.

Pascal

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

C++

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

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

TCustomSQLDataSet (see page 1421) is rarely used directly in applications because many of its properties are protected. Instead, applications instantiate TCustomSQLDataSet (see page 1421) descendants such as TSQLDataSet (see page 1556), TSQLTable (see page 1604), TSQLQuery (see page 1577), or TSQLStoredProc (see page 1594).

AOwner is the component that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner

Create

TSQLDataSet (see page 1556)

TSQLStoredProc (see page 1594)

TSQLTable (see page 1604)

TSQLQuery (see page 1577)

24.1.2.3 TCustomSQLDataSet.Destroy Destructor

Destroys the TCustomSQLDataSet (see page 1421) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomSQLDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1425)

System::TObject::Free (see page 3382)

24.1.2.4 TCustomSQLDataSet Methods

The methods of the TCustomSQLDataSet class are listed here.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.

	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Public Methods

	Name	Description
	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.

	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
	PrepareStatement (see page 1440)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

24.1.2.4.1 TCustomSQLDataSet.AddFieldDesc Method

Pascal

```
procedure AddFieldDesc(FieldDescs: TFieldDescList; DescNo: Integer; var FieldID: Integer;
RequiredFields: TBits; FieldDefs: TFieldDefs);
```

C++

```
__fastcall AddFieldDesc(TFieldDescList FieldDescs, int DescNo, int FieldID, TBits
RequiredFields, TFieldDefs FieldDefs);
```

Description

This is AddFieldDesc, a member of class TCustomSQLDataSet.

24.1.2.4.2 TCustomSQLDataSet.AddIndexDefs Method

Pascal

```
procedure AddIndexDefs(SourceDS: TCustomSQLDataSet; IndexName: string = '');
```

C++

```
__fastcall AddIndexDefs(TCustomSQLDataSet SourceDS, UnicodeString IndexName = '');
```

Description

This is AddIndexDefs, a member of class TCustomSQLDataSet.

24.1.2.4.3 TCustomSQLDataSet.Check Method

Pascal

```
function Check(status: SQLResult; eType: TSQLExceptionType): SQLResult;
```

C++

```
__fastcall SQLResult Check(SQLResult status, TSQLExceptionType eType);
```

Description

This is Check, a member of class TCustomSQLDataSet.

24.1.2.4.4 TCustomSQLDataSet.CheckPrepareError Method

Pascal

```
procedure CheckPrepareError;
```

C++

```
__fastcall CheckPrepareError();
```

Description

This is CheckPrepareError, a member of class TCustomSQLDataSet.

24.1.2.4.5 TCustomSQLDataSet.ClearIndexDefs Method

Pascal

```
procedure ClearIndexDefs;
```

C++

```
__fastcall ClearIndexDefs();
```

Description

This is ClearIndexDefs, a member of class TCustomSQLDataSet.

24.1.2.4.6 TCustomSQLDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TCustomSQLDataSet.

24.1.2.4.7 TCustomSQLDataSet.CloseStatement Method

Pascal

```
procedure CloseStatement;
```

C++

```
__fastcall CloseStatement();
```

Description

This is CloseStatement, a member of class TCustomSQLDataSet.

24.1.2.4.8 TCustomSQLDataSet.CreateBlobStream Method

Creates a blob stream for a Binary large object (BLOB) field in the dataset.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading and writing the value of the field specified by the Field parameter.

Field is a string containing the name of the BLOB field in the dataset. It must be an instance of TBlobField or one of its descendants.

Mode indicates whether the stream will be used for reading the field's value (bmRead), writing the field's value (bmWrite), or both (bmReadWrite).

CreateBlobStream returns the stream you can use for reading or writing data.

See Also

TSQLBlobStream (see page 1507)

TBlobField

24.1.2.4.9 TCustomSQLDataSet.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor

DefineBinaryProperty

DefineProperties

DefineProperty

24.1.2.4.10 TCustomSQLDataSet.ExecSQL Method

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; virtual;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

This is ExecSQL, a member of class TCustomSQLDataSet.

24.1.2.4.11 TCustomSQLDataSet.ExecuteStatement Method

Pascal

```
procedure ExecuteStatement;
```

C++

```
__fastcall ExecuteStatement();
```

Description

This is ExecuteStatement, a member of class TCustomSQLDataSet.

24.1.2.4.12 TCustomSQLDataSet.FreeBuffers Method

Pascal

```
procedure FreeBuffers;
```

C++

```
__fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TCustomSQLDataSet.

24.1.2.4.13 TCustomSQLDataSet.FreeCursor Method

Pascal

```
procedure FreeCursor;
```

C++

```
__fastcall FreeCursor();
```

Description

This is FreeCursor, a member of class TCustomSQLDataSet.

24.1.2.4.14 TCustomSQLDataSet.FreeStatement Method

Pascal

```
procedure FreeStatement;
```

C++

```
__fastcall FreeStatement();
```

Description

This is FreeStatement, a member of class TCustomSQLDataSet.

24.1.2.4.15 TCustomSQLDataSet.GetBlobFieldData Method

Retrieves the current value of a BLOB field into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

This method is used internally to read BLOB data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

GetBlobFieldData is intended primarily for internal use. Applications should use the BLOB stream returned by the CreateBlobStream (see page 1429) method to access BLOB data.

See Also

GetFieldData (see page 1433)

IsBlob

CreateBlobStream (see page 1429)

GetBlobSize

24.1.2.4.16 TCustomSQLDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TCustomSQLDataSet.

24.1.2.4.17 TCustomSQLDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TCustomSQLDataSet.

24.1.2.4.18 TCustomSQLDataSet.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLInkFIelds fills the two lists with field components that define a master-detail relationship between this dataset and another (master) dataset. The MasterFields list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from this dataset.

Applications seldom call GetDetailLinkFields. It is used internally by the Field Link designer when setting up master/detail relationships.

See Also

DataSource (see page 1457)

MasterFields (see page 1615)

MasterSource (see page 1615)

24.1.2.4.19 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TCustomSQLDataSet.GetFieldData (Integer, Pointer) (see page 1433)	Retrieves the current value of a field into a buffer.
	TCustomSQLDataSet.GetFieldData (TField, Pointer) (see page 1433)	Retrieves the current value of a field into a buffer.

24.1.2.4.19.1 TCustomSQLDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number.

Buffer is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). GetFieldData returns a value that indicates whether the data was successfully fetched.

GetFieldData returns a value that indicates whether the data was successfully fetched.

See Also

GetBlobFieldData (see page 1431)

IsBlob

GetData

24.1.2.4.20 GetKeyFieldNames Method

This is the overview for the GetKeyFieldNames method overload.

Overload List

	Name	Description
	TCustomSQLDataSet.GetKeyFieldNames (TStrings) (see page 1434)	Fills a list with the names of all fields in all indexes defined for the dataset.
	TCustomSQLDataSet.GetKeyFieldNames (TWideStrings) (see page 1434)	Fills a list with the names of all fields in all indexes defined for the dataset.

24.1.2.4.20.1 TCustomSQLDataSet.GetKeyFieldNames Method (TStrings)

Fills a list with the names of all fields in all indexes defined for the dataset.

Pascal

```
function GetKeyFieldNames(List: TStrings): Integer; overload;
function GetKeyFieldNames(List: TWideStrings): Integer; overload;
```

C++

```
__fastcall int GetKeyFieldNames(TStrings List);
__fastcall int GetKeyFieldNames(TWideStrings List);
```

Description

Call GetKeyFieldNames to fill a string list with information about the indexes that are available on the database server for this dataset.

For every index defined on the server for the dataset, GetKeyFieldNames adds a string to the string list specified by List. Each string is a semicolon-delimited set of field names that lists the fields in a single index.

TCustomSQLDataSet (see page 1421) does not explicitly use these indexes. GetKeyFieldNames can be useful for composing SQL commands dynamically that need to use an index to access fields.

See Also

IndexDefs (see page 1459)

24.1.2.4.21 TCustomSQLDataSet.GetObjectTypeNames Method

Pascal

```
procedure GetObjectTypeNames(Fields: TFields);
```

C++

```
__fastcall GetObjectTypeNames(TFields Fields);
```

Description

This is GetObjectTypeNames, a member of class TCustomSQLDataSet.

24.1.2.4.22 TCustomSQLDataSet.GetOutputParams Method

Pascal

```
procedure GetOutputParams(AProcParams: TList = nil);
```

C++

```
__fastcall GetOutputParams(TList AProcParams = nil);
```

Description

This is GetOutputParams, a member of class TCustomSQLDataSet.

24.1.2.4.23 TCustomSQLDataSet.GetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function GetQuoteChar: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetQuoteChar();
```

Description

GetQuoteChar returns the character or characters that delimit quoted strings in automatically generated SQL statements. This string appears both before and after quoted strings.

24.1.2.4.24 TCustomSQLDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; override;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TCustomSQLDataSet.

24.1.2.4.25 TCustomSQLDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TCustomSQLDataSet.

24.1.2.4.26 TCustomSQLDataSet.GetSortFieldNames Method

Pascal

```
function GetSortFieldNames: WideString;
```

C++

```
__fastcall BSTR GetSortFieldNames();
```

Description

This is GetSortFieldNames, a member of class TCustomSQLDataSet.

24.1.2.4.27 TCustomSQLDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InitRecord(const char * Buffer);
```

Description

This is InitRecord, a member of class TCustomSQLDataSet.

24.1.2.4.28 TCustomSQLDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TCustomSQLDataSet.

24.1.2.4.29 TCustomSQLDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TCustomSQLDataSet.

24.1.2.4.30 TCustomSQLDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.

24.1.2.4.31 TCustomSQLDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TCustomSQLDataSet.

24.1.2.4.32 TCustomSQLDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TCustomSQLDataSet.

24.1.2.4.33 TCustomSQLDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TCustomSQLDataSet.

24.1.2.4.34 TCustomSQLDataSet.IsSequenced Method

Indicates whether the dataset can use record numbers to indicate the order of records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Datasets use IsSequenced to indicate whether the RecNo property corresponds to the order of records. When IsSequenced returns true, applications can safely use the RecNo property to navigate to records in the dataset.

Note: For TCustomSQLDataSet (see page 1421), IsSequenced always returns false because TCustomSQLDataSet (see page 1421) descendants do not support record numbers at all.

See Also

RecNo

RecordCount (see page 1465)

24.1.2.4.35 TCustomSQLDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the

components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

24.1.2.4.36 TCustomSQLDataSet.Locate Method

Implements a virtual method for searching a dataset for a specified record and making it the active record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns false, indicating that a matching record was not found and the active record was not changed.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the semicolon-separated list of fields in KeyFields have the values specified by the Variant (see page 3430) or Variant (see page 3430) array KeyValues. Options indicates whether the search is case insensitive and whether partial matches are supported. Locate returns true if a record is found that matches the specified criteria and that record is now active.

See Also

FindField

Lookup

24.1.2.4.37 TCustomSQLDataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TCustomSQLDataSet.

24.1.2.4.38 TCustomSQLDataSet.Lookup Method

Implements a virtual method to retrieve field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const UnicodeString KeyFields, const Variant KeyValues,
const UnicodeString ResultFields);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns a Variant (see page 3430) with the value false, indicating that a matching record was not found.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the comma-delimited string KeyFields have the values specified by the Variant (see page 3430) or Variant (see page 3430) array KeyValues. In classes that implement Lookup, it returns a Variant (see page 3430) or Variant (see page 3430) array that contains the value or values of the fields specified by the comma-delimited string ResultFields on the specified record.

See Also

Locate (see page 1438)

24.1.2.4.39 TCustomSQLDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TCustomSQLDataSet.

24.1.2.4.40 TCustomSQLDataSet.OpenSchema Method

Pascal

```
procedure OpenSchema; virtual;
```

C++

```
virtual __fastcall OpenSchema();
```

Description

This is OpenSchema, a member of class TCustomSQLDataSet.

24.1.2.4.41 TCustomSQLDataSet.ParamByName Method

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const UnicodeString Value);
```

Description

This is ParamByName, a member of class TCustomSQLDataSet.

24.1.2.4.42 TCustomSQLDataSet.PrepareStatement Method

Performs a prepare on the SQL statement.

Pascal

```
procedure PrepareStatement; virtual;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

The prepare operation is a type of pre-compile that checks the performance of the SQL statement. Some databases support a prepare statement that creates an execution plan that speeds processing.

24.1.2.4.43 TCustomSQLDataSet.PropertyChanged Method

Pascal

```
procedure PropertyChanged;
```

C++

```
__fastcall PropertyChanged();
```

Description

This is PropertyChanged, a member of class TCustomSQLDataSet.

24.1.2.4.44 TCustomSQLDataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction

PSInTransaction

applyupdates

24.1.2.4.45 TCustomSQLDataSet.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText

Execute

Options

24.1.2.4.46 TCustomSQLDataSet.PSExecuteStatement Method

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: Pointer =
nil): Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, void * ResultSet =
nil);
```

Description

If the value of its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord

applyupdates

ResolveToDataSet

24.1.2.4.47 TCustomSQLDataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); override;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes

24.1.2.4.48 TCustomSQLDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText

PSExecute

PSSetCommandText

GetRecords

Execute

Options

24.1.2.4.49 TCustomSQLDataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText

PSExecute

PSGetCommandText

Execute

Options

24.1.2.4.50 TCustomSQLDataSet.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket

24.1.2.4.51 TCustomSQLDataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options

PSGetKeyFields

PSGetDefaultOrder

24.1.2.4.52 TCustomSQLDataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.

24.1.2.4.53 TCustomSQLDataSet.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams

24.1.2.4.54 TCustomSQLDataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.

24.1.2.4.55 TCustomSQLDataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TCustomSQLDataSet.

24.1.2.4.56 TCustomSQLDataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create

24.1.2.4.57 TCustomSQLDataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction

PSEndTransaction

24.1.2.4.58 TCustomSQLDataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported

PSExecuteStatement

24.1.2.4.59 TCustomSQLDataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset

that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased

PSExecuteStatement

ResolveToDataSet

24.1.2.4.60 TCustomSQLDataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords

AS_GetRecords

24.1.2.4.61 TCustomSQLDataSet.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const ACommandText: WideString); override;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR ACommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText

PSExecute

GetRecords

Execute

Options

24.1.2.4.62 TCustomSQLDataSet.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params

PSGetParams

24.1.2.4.63 TCustomSQLDataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction

PSInTransaction

applyupdates

24.1.2.4.64 TCustomSQLDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in

some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord

TUpdateSQL

applyupdates

ResolveToDataSet

PSExecuteStatement

24.1.2.4.65 TCustomSQLDataSet.SetBufListSize Method

Pascal

```
procedure SetBufListSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetBufListSize(int Value);
```

Description

This is SetBufListSize, a member of class TCustomSQLDataSet.

24.1.2.4.66 TCustomSQLDataSet.SetCommandText Method

Pascal

```
procedure SetCommandText(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const BSTR Value);
```

Description

This is SetCommandText, a member of class TCustomSQLDataSet.

24.1.2.4.67 TCustomSQLDataSet.SetFCommandText Method

Pascal

```
procedure SetFCommandText(const Value: string);
```

C++

```
__fastcall SetFCommandText(const UnicodeString Value);
```

Description

This is SetFCommandText, a member of class TCustomSQLDataSet.

24.1.2.4.68 TCustomSQLDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TCustomSQLDataSet.

24.1.2.4.69 TCustomSQLDataSet.SetMaxBlobSize Method

Pascal

```
procedure SetMaxBlobSize(MaxSize: Integer);
```

C++

```
__fastcall SetMaxBlobSize(int MaxSize);
```

Description

This is SetMaxBlobSize, a member of class TCustomSQLDataSet.

24.1.2.4.70 TCustomSQLDataSet.SetParamsFromCursor Method

Pascal

```
procedure SetParamsFromCursor;
```

C++

```
__fastcall SetParamsFromCursor();
```

Description

This is SetParamsFromCursor, a member of class TCustomSQLDataSet.

24.1.2.4.71 TCustomSQLDataSet.SetSchemaInfo Method

Indicates whether the dataset represents metadata from the server, and if so, what kind.

Pascal

```
procedure SetSchemaInfo(SchemaType: TSchemaType; SchemaObjectName: WideString; SchemaPattern:
WideString; PackageName: WideString = '');
```

C++

```
__fastcall SetSchemaInfo(TSchemaType SchemaType, BSTR SchemaObjectName, BSTR SchemaPattern,
BSTR PackageName = '');
```

Description

Call SetSchemaInfo to indicate whether the SQL dataset represents the results of an SQL command (specified by CommandText (see page 1455)) or whether it represents a table of schema information (metadata).

The SchemaType parameter identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. The SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.
stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.

SchemaObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When SchemaType is stColumns or stIndexes, SchemaObjectName is the name of a table. When SchemaType is stProcedureParams, SchemaObjectName is the name of a stored procedure. When SchemaType is stNoSchema, stTables, stSysTables, or stProcedures, or stPackages, SchemaObjectName is ignored.

SchemaPattern is an SQL pattern mask that filters the resulting dataset. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. SchemaPattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package in which to find the stored procedure when SchemaType is stProcedures or stProcedureParams. If the server is not an Oracle server, or if the stored procedure is not defined in a package, PackageName should be an empty string.

After a call to SetSchemaInfo, opening the dataset results in a dataset populated with the specified information. If SchemaType is stNoSchema, the dataset reverts to the command specified by CommandText (see page 1455).

Note: When you change the CommandText (see page 1455) property, the dataset automatically reverts to a SchemaType of stNoSchema.

See Also

Open

CommandText (see page 1455)

TableScope (see page 1548)

Fetching Metadata into a Unidirectional Dataset

Accessing Schema Information

24.1.2.4.72 TCustomSQLDataSet.SetSortFieldNames Method

Pascal

```
procedure SetSortFieldNames(Value: WideString);
```

C++

```
__fastcall SetSortFieldNames(BSTR Value);
```

Description

This is SetSortFieldNames, a member of class TCustomSQLDataSet.

24.1.2.4.73 TCustomSQLDataSet.SQLError Method

Pascal

```
procedure SQLError(OpStatus: SQLResult; eType: TSQLExceptionType);
```

C++

```
__fastcall SQLError(SQLResult OpStatus, TSQLExceptionType eType);
```

Description

This is SQLError, a member of class TCustomSQLDataSet.

24.1.2.4.74 TCustomSQLDataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

24.1.2.5 TCustomSQLDataSet Properties

The properties of the TCustomSQLDataSet class are listed here.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1457)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	MaxBlobSize (see page 1460)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1462)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	Params (see page 1463)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1467)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1466)	Specifies the name of the database schema.

24.1.2.5.1 TCustomSQLDataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      // This path depends on the Demos directory of your RAD Studio installation.
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
end;
```

See Also

DataSource

State

AfterOpen

BeforeOpen

Close

Open

CanModify

BeforeClose

AfterClose

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

24.1.2.5.2 TCustomSQLDataSet.BlobBuffer Property

Reserves a buffer for storing BLOB values.

Pascal

```
property BlobBuffer: TBlobByteData;
```

C++

```
__property TBlobByteData BlobBuffer;
```

Description

SQL datasets use the BlobBuffer property to allocate a special buffer for holding BLOB data. Applications do not access this protected property.

See Also

MaxBlobSize (see page 1460)

GetBlobFieldData

24.1.2.5.3 TCustomSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 1456) property:

When CommandType (see page 1456) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL (see page 1430) method is called.

When CommandType (see page 1456) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL (see page 1430) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 1456) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 1466) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 1463) property to supply parameter values.

See Also

CommandType (see page 1456)

Params (see page 1463)

Open

ExecSQL (see page 1560)

SortFieldNames (see page 1466)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

24.1.2.5.4 TCustomSQLDataSet.CommandType Property

Indicates the meaning of the CommandText (see page 1455) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText (see page 1455) property. TCustomSQLDataSet (see page 1421) descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 1455)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 1455)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

24.1.2.5.5 TCustomSQLDataSet.CurrentBlobSize Property

Indicates the size of the most recently read BLOB field.

Pascal

```
property CurrentBlobSize: LongWord;
```

C++

```
__property LongWord CurrentBlobSize;
```

Description

Applications can't use this protected property. TCustomSQLDataSet (see page 1421) uses CurrentBlobSize internally to determine the size needed for BlobBuffer (see page 1454). When the dataset reads a BLOB field, it sets CurrentBlobSize to the number of bytes in the BLOB field, as reported by the database driver.

See Also

MaxBlobSize (see page 1460)

BlobBuffer (see page 1454)

24.1.2.5.6 TCustomSQLDataSet.DataLink Property

Identifies the datalink that manages communication between the SQL dataset and another, master dataset.

Pascal

```
property DataLink: TDataLink;
```

C++

```
__property TDataLink DataLink;
```

Description

The protected DataLink property is used internally when the SQL dataset acts as the detail dataset in a master/detail relationship that is not based on nested datasets. The datalink object manages communication between another (master) dataset and this SQL dataset. For example, it is the datalink that tells this SQL dataset to refresh its data when the active record of the master dataset changes.

Do not use DataLink to establish the relationship between an SQL dataset and another (master) dataset from which it obtains parameter values. Instead, use the DataSource (see page 1457) property, or, if the dataset is TSQLTable (see page 1604), the MasterSource property.

See Also

DataSource (see page 1457)

TSQLDataLink (see page 1553)

MasterSource (see page 1615)

24.1.2.5.7 TCustomSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 1455) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 1463) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1455)

SQL (see page 1587)

Params (see page 1463)

MasterSource (see page 1615)

24.1.2.5.8 TCustomSQLDataSet.DesignerData Property

Stores custom data.

Pascal

```
property DesignerData: string;
```

C++

```
__property UnicodeString DesignerData;
```

Description

DesignerData has no predefined meaning. Application developers can use the DesignerData property to associate application-specific information with an SQL dataset instance.

See Also

Tag

24.1.2.5.9 TCustomSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider

Fetching the Data using TSQLDataSet

24.1.2.5.10 TCustomSQLDataSet.IndexDefs Property

Contains the definitions of all indexes defined on the server for the dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Read (see page 3727) IndexDefs to obtain information about what indexes are available on the database server for this dataset.

TCustomSQLDataSet (see page 1421) does not explicitly use the information in IndexDefs. The order of its records is determined by the ORDER BY clause of its SQL statement.

If the SQL dataset is an instance of TSQLTable (see page 1604), you can use IndexDefs to identify the index to use for the IndexName property. Even with other TCustomSQLDataSet (see page 1421) descendants, IndexDefs can be useful for composing SQL commands dynamically that need to use an index on the server. For example, provider components use the index information to uniquely identify records when composing update commands.

See Also

TIndexDef

SortFieldNames (see page 1466)

CommandText (see page 1455)

SQL (see page 1587)

TDataSetProvider

IndexName (see page 1614)

GetKeyFieldNames (see page 1434)

24.1.2.5.11 TCustomSQLDataSet.InternalConnection Property

Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.

Pascal

```
property InternalConnection: TSqlConnection;
```

C++

```
__property TSqlConnection InternalConnection;
```

Description

Applications can't use this protected property. Instead, SQL datasets use InternalConnection internally when they need to access the dbExpress interfaces for the connection to the database server. Unlike the published SQLConnection (see page 1467) property, InternalConnection handles the case when the dataset must use a clone of the primary connection because the server limits the number of active statements that the connection supports.

It is not necessary to use InternalConnection except when using the connection component to reference its dbExpress interfaces.

See Also

CloneConnection (see page 1515)

AutoClone (see page 1532)

SQLConnection (see page 1467)

24.1.2.5.12 TCustomSQLDataSet.LastError Property

Saves the error message for the last error returned by dbExpress.

Pascal

```
property LastError: string;
```

C++

```
__property UnicodeString LastError;
```

Description

When the SQL dataset encounters an error in a call to dbExpress, it caches the error message. This allows it to continue until it is in an appropriate state to raise the associated exception. The message is used to raise an exception with the error message when it is safe to do so.

24.1.2.5.13 TCustomSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1546)

SQLConnection (see page 1467)

24.1.2.5.14 TCustomSQLDataSet.NativeCommand Property

Represents the SQL command that is sent to the server.

Pascal

```
property NativeCommand: WideString;
```

C++

```
__property BSTR NativeCommand;
```

Description

SQL datasets sometimes need to alter a query before it can be sent to the dbExpress driver. For example, dbExpress does not support named parameters, so the SQL dataset converts named parameters to unnamed parameters before sending off a command.

NativeCommand is the adjusted command that is sent to the dbExpress driver. It is based on the value of CommandText (see page 1455).

See Also

CommandText (see page 1455)

24.1.2.5.15 TCustomSQLDataSet.NumericMapping Property

Maps fldBCD to either ftFMTBCD or ftBCD field types.

Pascal

```
property NumericMapping: Boolean;
```

C++

```
__property Boolean NumericMapping;
```

Description

Use NumericMapping to control how to map fldBCD (dbExpress Core logical type) to eitheir ftFMTBCD or ftBCD field types. The default is False. If True, all dbExpress fldBCD types will map to ftFMTBCD across all dbExpress drivers for field type compatibility.

See Also

TFieldType

24.1.2.5.16 TCustomSQLDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView;
```

C++

```
__property ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased

compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList

Fields

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

24.1.2.5.17 TCustomSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 1463) property is updated automatically to reflect any parameters that appear in the query specified by the CommandText (see page 1455) property (or the SQL property for TSQLQuery (see page 1577)) at runtime. That is, whenever the CommandText (see page 1455) property changes, the Params (see page 1463) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 1463) property in response to changes in CommandText (see page 1455), set ParamCheck to false.

See Also

Params (see page 1463)

CommandText (see page 1455)

SQL (see page 1587)

24.1.2.5.18 TCustomSQLDataSet.ParamCount Property

Indicates the number of parameters listed by the Params (see page 1463) property.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters listed in the Params (see page 1463) property. Applications can't use this protected property. It is used internally when preparing SQL statements for execution.

See Also

Params (see page 1463)

Prepared (see page 1464)

24.1.2.5.19 TCustomSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 1455) property (or the SQL property in TSQLQuery (see page 1577)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 1455) property (or the StoredProcName property in TSQLStoredProc (see page 1594)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 1457) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1462)

SQL (see page 1587)

StoredProcName (see page 1603)

CommandText (see page 1455)

TParam

Value

DataType

DataSource (see page 1457)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

24.1.2.5.20 TCustomSQLDataSet.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

SQL datasets are automatically prepared when you call Open or ExecSQL (see page 1430) (or ExecProc). When a SQL dataset is prepared, dbExpress and the server allocate resources for the SQL statement and its parameters. If CommandType (see page 1456) is ctTable, the dataset generates its SELECT query when it is prepared. Any parameters that are not bound by the server are folded into a query when it is prepared.

If you execute the query or stored procedure more than once, you can improve performance by setting Prepared to true before opening the dataset. Setting Prepared to true prepares the dataset at once, rather than waiting until the dataset is opened.

Conversely, set Prepared to false if you want to ensure that the dataset is re-prepared before it executes (for example, if you change a parameter value or the SortFieldNames (see page 1466) property).

See Also

Params (see page 1463)

ExecSQL (see page 1560)

Open

Executing the Command

Fetching the Data

24.1.2.5.21 TCustomSQLDataSet.ProcParams Property

ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.

Pascal

```
property ProcParams: TList;
```

C++

```
__property TList ProcParams;
```

Description

When the SQL dataset represents a stored procedure, it fetches information about stored procedure parameters from the server as a list of parameter descriptors. It stores this list as the value of ProcParams.

Applications never use the protected ProcParams directly. Instead, the information from the list is used to initialize the Params (see page 1463) property.

See Also

Params (see page 1463)

GetProcedureParams (see page 1524)

24.1.2.5.22 TCustomSQLDataSet.RecordCount Property

Indicates the total number of records associated with the dataset.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Read (see page 3727) RecordCount to determine the total number of records in the dataset. An application can read RecordCount even if the dataset is not active.

Reading RecordCount will generate an exception if the dataset can't determine the number of records. Do not read RecordCount if

The dataset represents stored procedure.

The dataset represents a query that contains parameters.

The dataset represents a multi-table join.

See Also

RecNo

Eof

24.1.2.5.23 TCustomSQLDataSet.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount (see page 1465) property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 1560)

RecordCount (see page 1465)

ExecProc (see page 1599)

24.1.2.5.24 TCustomSQLDataSet.SchemaInfo Property

Identifies the metadata with which to populate the dataset.

Pascal

```
property SchemaInfo: TSQLSchemaInfo;
```

C++

```
__property TSQLSchemaInfo SchemaInfo;
```

Description

SQL datasets can be used to represent metadata from the database server rather than the results of a query or stored procedure. SchemaInfo controls whether the dataset represents metadata, and if so, what kind.

When SchemaInfo specifies that the dataset represents schema information, the dataset is populated with that information when it is opened. Each record in the resulting dataset represents a single entity (a single table, stored procedure, field, parameter, or index).

Each type of schema information results in a different set of columns (fields), so it is not advisable to use persistent fields when accessing metadata through an SQL dataset.

Applications do not work directly with the protected SchemaInfo property. Instead, they specify the desired schema information by calling the SetSchemaInfo (see page 1450) method.

See Also

SetSchemaInfo (see page 1450)

Open

24.1.2.5.25 TCustomSQLDataSet.SchemaName Property

Specifies the name of the database schema.

Pascal

```
property SchemaName: string;
```

C++

```
__property UnicodeString SchemaName;
```

Description

This name will be available in the appropriate drop down list box after you have connected to a given data source.

24.1.2.5.26 TCustomSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 1456) is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 1456) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1456)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

24.1.2.5.27 TCustomSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 1510) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 1510) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 1455) property or populate the dataset with data by setting Active (see page 1453) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 1463) property when you set CommandText (see page 1455).

See Also

Active

CommandText (see page 1455)

Params (see page 1463)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

24.1.2.5.28 TCustomSQLDataSet.TransactionLevel Property

Indicates the transaction to which this dataset belongs.

Pascal

```
property TransactionLevel: SmallInt;
```

C++

```
__property SmallInt TransactionLevel;
```

Description

Use TransactionLevel to indicate the active transaction to use when opening this dataset or using it to execute commands. TransactionLevel is the same as the transaction identifier that is supplied to the connection's StartTransaction method. If TransactionLevel is unspecified (0), the dataset fetches its records or executes commands in the context of the most recently started transaction.

Note: Most applications do not need to use TransactionLevel because most database servers require that SQL commands belong to the most recently started transaction. However, the InterBase server allows multiple overlapping transactions to be active simultaneously.

See Also

InTransaction (see page 1537)

Commit (see page 1516)

Rollback (see page 1529)

Params (see page 1546)

TransactionsSupported (see page 1549)

StartTransaction

Managing Transactions

24.1.2.6 TCustomSQLDataSet Events

The events of the TCustomSQLDataSet class are listed here.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

24.1.2.6.1 TCustomSQLDataSet.AfterClose Event

Occurs after a data source connection is closed.

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

24.1.2.6.2 TCustomSQLDataSet.AfterOpen Event

Occurs after the connection to a data source is opened.

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

24.1.2.6.3 TCustomSQLDataSet.AfterRefresh Event

Occurs after a DataSet has been changed and is refreshed.

Pascal

```
property AfterRefresh;
```

C++

```
__property AfterRefresh;
```

24.1.2.6.4 TCustomSQLDataSet.AfterScroll Event

Occurs after a result set is scrolled.

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

24.1.2.6.5 TCustomSQLDataSet.BeforeClose Event

Occurs just before a data connection is closed.

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

24.1.2.6.6 TCustomSQLDataSet.BeforeOpen Event

Occurs just before a data connection is opened, but after an open operation has been initiated.

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

24.1.2.6.7 TCustomSQLDataSet.BeforeRefresh Event

Occurs before a DataSet refresh occurs.

Pascal

```
property BeforeRefresh;
```

C++

```
__property BeforeRefresh;
```

24.1.2.6.8 TCustomSQLDataSet.BeforeScroll Event

Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

24.1.2.6.9 TCustomSQLDataSet.OnCalcFields Event

Occurs when calculated or derived fields are updated.

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

24.1.2.6.10 TCustomSQLDataSet.ParseDeleteSql Event

Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.

Pascal

```
property ParseDeleteSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseDeleteSql;
```

24.1.2.6.11 TCustomSQLDataSet.ParseInsertSql Event

Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.

Pascal

```
property ParseInsertSql: TParseInsertSqlEvent;
```

C++

```
__property TParseInsertSqlEvent ParseInsertSql;
```

24.1.2.6.12 TCustomSQLDataSet.ParseSelectSql Event

Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.

Pascal

```
property ParseSelectSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseSelectSql;
```

24.1.2.6.13 TCustomSQLDataSet.ParseUpdateSql Event

Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Pascal

```
property ParseUpdateSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseUpdateSql;
```

24.1.3 TFLDDesc Class

Class Hierarchy

Pascal

```
TFLDDesc = class;
```

C++

```
class TFLDDesc;
```

File

SqlExpr

Description

This is class SqlExpr.TFLDDesc.

24.1.3.1 TFLDDesc Members

The following tables list the members exposed by TFLDDesc.

Protected Methods

	Name	Description
	GetbCalcField (see page 1473)	This is GetbCalcField, a member of class TFLDDesc.
	GetefldrRights (see page 1473)	This is GetefldrRights, a member of class TFLDDesc.
	GetefldvVchk (see page 1473)	This is GetefldvVchk, a member of class TFLDDesc.
	GetiFldNum (see page 1473)	This is GetiFldNum, a member of class TFLDDesc.
	GetiFldType (see page 1474)	This is GetiFldType, a member of class TFLDDesc.
	GetiLen (see page 1474)	This is GetiLen, a member of class TFLDDesc.
	GetiNullOffset (see page 1474)	This is GetiNullOffset, a member of class TFLDDesc.
	GetiOffset (see page 1474)	This is GetiOffset, a member of class TFLDDesc.
	GetiSubType (see page 1474)	This is GetiSubType, a member of class TFLDDesc.
	GetiUnits1 (see page 1475)	This is GetiUnits1, a member of class TFLDDesc.

	GetiUnits2 (see page 1475)	This is GetiUnits2, a member of class TFLDDesc.
	GetszName (see page 1475)	This is GetszName, a member of class TFLDDesc.
	SetbCalcField (see page 1475)	This is SetbCalcField, a member of class TFLDDesc.
	SetefldrRights (see page 1476)	This is SetefldrRights, a member of class TFLDDesc.
	SetefldvVchk (see page 1476)	This is SetefldvVchk, a member of class TFLDDesc.
	SetiFldNum (see page 1476)	This is SetiFldNum, a member of class TFLDDesc.
	SetiFldType (see page 1476)	This is SetiFldType, a member of class TFLDDesc.
	SetiLen (see page 1476)	This is SetiLen, a member of class TFLDDesc.
	SetiNullOffset (see page 1477)	This is SetiNullOffset, a member of class TFLDDesc.
	SetiOffset (see page 1477)	This is SetiOffset, a member of class TFLDDesc.
	SetiSubType (see page 1477)	This is SetiSubType, a member of class TFLDDesc.
	SetiUnits1 (see page 1477)	This is SetiUnits1, a member of class TFLDDesc.
	SetiUnits2 (see page 1478)	This is SetiUnits2, a member of class TFLDDesc.
	SetszName (see page 1478)	This is SetszName, a member of class TFLDDesc.

Public Properties

	Name	Description
	bCalcField (see page 1478)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 1479)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 1479)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 1479)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 1479)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 1479)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 1480)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 1480)	This is iOffset, a member of class TFLDDesc.
	iSubType (see page 1480)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 1480)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 1481)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 1481)	This is szName, a member of class TFLDDesc.

24.1.3.2 TFLDDesc Methods

The methods of the TFLDDesc class are listed here.

Protected Methods

	Name	Description
	GetbCalcField (see page 1473)	This is GetbCalcField, a member of class TFLDDesc.
	GetefldrRights (see page 1473)	This is GetefldrRights, a member of class TFLDDesc.
	GetefldvVchk (see page 1473)	This is GetefldvVchk, a member of class TFLDDesc.
	GetiFldNum (see page 1473)	This is GetiFldNum, a member of class TFLDDesc.
	GetiFldType (see page 1474)	This is GetiFldType, a member of class TFLDDesc.
	GetiLen (see page 1474)	This is GetiLen, a member of class TFLDDesc.
	GetiNullOffset (see page 1474)	This is GetiNullOffset, a member of class TFLDDesc.
	GetiOffset (see page 1474)	This is GetiOffset, a member of class TFLDDesc.
	GetiSubType (see page 1474)	This is GetiSubType, a member of class TFLDDesc.
	GetiUnits1 (see page 1475)	This is GetiUnits1, a member of class TFLDDesc.
	GetiUnits2 (see page 1475)	This is GetiUnits2, a member of class TFLDDesc.
	GetszName (see page 1475)	This is GetszName, a member of class TFLDDesc.
	SetbCalcField (see page 1475)	This is SetbCalcField, a member of class TFLDDesc.
	SetefldrRights (see page 1476)	This is SetefldrRights, a member of class TFLDDesc.
	SetefldvVchk (see page 1476)	This is SetefldvVchk, a member of class TFLDDesc.
	SetiFldNum (see page 1476)	This is SetiFldNum, a member of class TFLDDesc.
	SetiFldType (see page 1476)	This is SetiFldType, a member of class TFLDDesc.

	SetiLen (see page 1476)	This is SetiLen, a member of class TFLDDesc.
	SetiNullOffset (see page 1477)	This is SetiNullOffset, a member of class TFLDDesc.
	SetiOffset (see page 1477)	This is SetiOffset, a member of class TFLDDesc.
	SetiSubType (see page 1477)	This is SetiSubType, a member of class TFLDDesc.
	SetiUnits1 (see page 1477)	This is SetiUnits1, a member of class TFLDDesc.
	SetiUnits2 (see page 1478)	This is SetiUnits2, a member of class TFLDDesc.
	SetszName (see page 1478)	This is SetszName, a member of class TFLDDesc.

24.1.3.2.1 TFLDDesc.GetbCalcField Method

Pascal

```
function GetbCalcField: WordBool; virtual; abstract;
```

C++

```
virtual __fastcall WordBool GetbCalcField() = 0;
```

Description

This is GetbCalcField, a member of class TFLDDesc.

24.1.3.2.2 TFLDDesc.GetefldrRights Method

Pascal

```
function GetefldrRights: FLDRights; virtual; abstract;
```

C++

```
virtual __fastcall FLDRights GetefldrRights() = 0;
```

Description

This is GetefldrRights, a member of class TFLDDesc.

24.1.3.2.3 TFLDDesc.GetefldvVchk Method

Pascal

```
function GetefldvVchk: FLDVchk; virtual; abstract;
```

C++

```
virtual __fastcall FLDVchk GetefldvVchk() = 0;
```

Description

This is GetefldvVchk, a member of class TFLDDesc.

24.1.3.2.4 TFLDDesc.GetiFldNum Method

Pascal

```
function GetiFldNum: Word; virtual; abstract;
```

C++

```
virtual __fastcall Word GetiFldNum() = 0;
```

Description

This is GetiFldNum, a member of class TFLDDesc.

24.1.3.2.5 TFLDDesc.GetiFldType Method

Pascal

```
function GetiFldType: Word; virtual; abstract;
```

C++

```
virtual __fastcall Word GetiFldType() = 0;
```

Description

This is GetiFldType, a member of class TFLDDesc.

24.1.3.2.6 TFLDDesc.GetiLen Method

Pascal

```
function GetiLen: LongWord; virtual; abstract;
```

C++

```
virtual __fastcall LongWord GetiLen() = 0;
```

Description

This is GetiLen, a member of class TFLDDesc.

24.1.3.2.7 TFLDDesc.GetiNullOffset Method

Pascal

```
function GetiNullOffset: Word; virtual; abstract;
```

C++

```
virtual __fastcall Word GetiNullOffset() = 0;
```

Description

This is GetiNullOffset, a member of class TFLDDesc.

24.1.3.2.8 TFLDDesc.GetiOffset Method

Pascal

```
function GetiOffset: Word; virtual; abstract;
```

C++

```
virtual __fastcall Word GetiOffset() = 0;
```

Description

This is GetiOffset, a member of class TFLDDesc.

24.1.3.2.9 TFLDDesc.GetiSubType Method

Pascal

```
function GetiSubType: Word; virtual; abstract;
```

C++

```
virtual __fastcall Word GetiSubType() = 0;
```

Description

This is GetiSubType, a member of class TFLDDesc.

24.1.3.2.10 TFLDDesc.GetiUnits1 Method

Pascal

```
function GetiUnits1: SmallInt; virtual; abstract;
```

C++

```
virtual __fastcall SmallInt GetiUnits1() = 0;
```

Description

This is GetiUnits1, a member of class TFLDDesc.

24.1.3.2.11 TFLDDesc.GetiUnits2 Method

Pascal

```
function GetiUnits2: SmallInt; virtual; abstract;
```

C++

```
virtual __fastcall SmallInt GetiUnits2() = 0;
```

Description

This is GetiUnits2, a member of class TFLDDesc.

24.1.3.2.12 TFLDDesc.GetszName Method

Pascal

```
function GetszName: WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR GetszName() = 0;
```

Description

This is GetszName, a member of class TFLDDesc.

24.1.3.2.13 TFLDDesc.SetbCalcField Method

Pascal

```
procedure SetbCalcField(Value: WordBool); virtual; abstract;
```

C++

```
virtual __fastcall SetbCalcField(WordBool Value) = 0;
```

Description

This is SetbCalcField, a member of class TFLDDesc.

24.1.3.2.14 TFLDDesc.SetefldrRights Method

Pascal

```
procedure SetefldrRights(Value: FLDRights); virtual; abstract;
```

C++

```
virtual __fastcall SetefldrRights(FLDRights Value) = 0;
```

Description

This is SetefldrRights, a member of class TFLDDesc.

24.1.3.2.15 TFLDDesc.SetefldvVchk Method

Pascal

```
procedure SetefldvVchk(Value: FLDVchk); virtual; abstract;
```

C++

```
virtual __fastcall SetefldvVchk(FLDVchk Value) = 0;
```

Description

This is SetefldvVchk, a member of class TFLDDesc.

24.1.3.2.16 TFLDDesc.SetiFldNum Method

Pascal

```
procedure SetiFldNum(Value: Word); virtual; abstract;
```

C++

```
virtual __fastcall SetiFldNum(Word Value) = 0;
```

Description

This is SetiFldNum, a member of class TFLDDesc.

24.1.3.2.17 TFLDDesc.SetiFldType Method

Pascal

```
procedure SetiFldType(Value: Word); virtual; abstract;
```

C++

```
virtual __fastcall SetiFldType(Word Value) = 0;
```

Description

This is SetiFldType, a member of class TFLDDesc.

24.1.3.2.18 TFLDDesc.SetiLen Method

Pascal

```
procedure SetiLen(Value: LongWord); virtual; abstract;
```

C++

```
virtual __fastcall SetiLen(LongWord Value) = 0;
```

Description

This is SetiLen, a member of class TFLDDesc.

24.1.3.2.19 TFLDDesc.SetiNullOffset Method

Pascal

```
procedure SetiNullOffset(Value: Word); virtual; abstract;
```

C++

```
virtual __fastcall SetiNullOffset(Word Value) = 0;
```

Description

This is SetiNullOffset, a member of class TFLDDesc.

24.1.3.2.20 TFLDDesc.SetiOffset Method

Pascal

```
procedure SetiOffset(Value: Word); virtual; abstract;
```

C++

```
virtual __fastcall SetiOffset(Word Value) = 0;
```

Description

This is SetiOffset, a member of class TFLDDesc.

24.1.3.2.21 TFLDDesc.SetiSubType Method

Pascal

```
procedure SetiSubType(Value: Word); virtual; abstract;
```

C++

```
virtual __fastcall SetiSubType(Word Value) = 0;
```

Description

This is SetiSubType, a member of class TFLDDesc.

24.1.3.2.22 TFLDDesc.SetiUnits1 Method

Pascal

```
procedure SetiUnits1(Value: SmallInt); virtual; abstract;
```

C++

```
virtual __fastcall SetiUnits1(SmallInt Value) = 0;
```

Description

This is SetiUnits1, a member of class TFLDDesc.

24.1.3.2.23 TFLDDesc.SetiUnits2 Method

Pascal

```
procedure SetiUnits2(Value: SmallInt); virtual; abstract;
```

C++

```
virtual __fastcall SetiUnits2(SmallInt Value) = 0;
```

Description

This is SetiUnits2, a member of class TFLDDesc.

24.1.3.2.24 TFLDDesc.SetszName Method

Pascal

```
procedure SetszName(Value: WideString); virtual; abstract;
```

C++

```
virtual __fastcall SetszName(BSTR Value) = 0;
```

Description

This is SetszName, a member of class TFLDDesc.

24.1.3.3 TFLDDesc Properties

The properties of the TFLDDesc class are listed here.

Public Properties

	Name	Description
	bCalcField (see page 1478)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 1479)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 1479)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 1479)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 1479)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 1479)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 1480)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 1480)	This is iOffset, a member of class TFLDDesc.
	iSubType (see page 1480)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 1480)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 1481)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 1481)	This is szName, a member of class TFLDDesc.

24.1.3.3.1 TFLDDesc.bCalcField Property

Pascal

```
property bCalcField: WordBool;
```

C++

```
__property WordBool bCalcField;
```

Description

This is bCalcField, a member of class TFLDDesc.

24.1.3.3.2 TFLDDesc.efldrRights Property

Pascal

```
property efldrRights: FLDRights;
```

C++

```
__property FLDRights efldrRights;
```

Description

This is efldrRights, a member of class TFLDDesc.

24.1.3.3.3 TFLDDesc.efldvVchk Property

Pascal

```
property efldvVchk: FLDVchk;
```

C++

```
__property FLDVchk efldvVchk;
```

Description

This is efldvVchk, a member of class TFLDDesc.

24.1.3.3.4 TFLDDesc.iFldNum Property

Pascal

```
property iFldNum: Word;
```

C++

```
__property Word iFldNum;
```

Description

This is iFldNum, a member of class TFLDDesc.

24.1.3.3.5 TFLDDesc.iFldType Property

Pascal

```
property iFldType: Word;
```

C++

```
__property Word iFldType;
```

Description

This is iFldType, a member of class TFLDDesc.

24.1.3.3.6 TFLDDesc.iLen Property

Pascal

```
property iLen: LongWord;
```

C++

```
__property LongWord iLen;
```

Description

This is iLen, a member of class TFLDDesc.

24.1.3.3.7 TFLDDesc.iNullOffset Property

Pascal

```
property iNullOffset: Word;
```

C++

```
__property Word iNullOffset;
```

Description

This is iNullOffset, a member of class TFLDDesc.

24.1.3.3.8 TFLDDesc.iOffset Property

Pascal

```
property iOffset: Word;
```

C++

```
__property Word iOffset;
```

Description

This is iOffset, a member of class TFLDDesc.

24.1.3.3.9 TFLDDesc.iSubType Property

Pascal

```
property iSubType: Word;
```

C++

```
__property Word iSubType;
```

Description

This is iSubType, a member of class TFLDDesc.

24.1.3.3.10 TFLDDesc.iUnits1 Property

Pascal

```
property iUnits1: SmallInt;
```

C++

```
__property SmallInt iUnits1;
```

Description

This is iUnits1, a member of class TFLDDesc.

24.1.3.3.11 TFLDDesc.iUnits2 Property

Pascal

```
property iUnits2: SmallInt;
```

C++

```
__property SmallInt iUnits2;
```

Description

This is iUnits2, a member of class TFLDDesc.

24.1.3.3.12 TFLDDesc.szName Property

Pascal

```
property szName: WideString;
```

C++

```
__property BSTR szName;
```

Description

This is szName, a member of class TFLDDesc.

24.1.4 TISQLCommand Class

Class Hierarchy

Pascal

```
TISQLCommand = class;
```

C++

```
class TISQLCommand;
```

File

SqlExpr

Description

This is class SqlExpr.TISQLCommand.

24.1.4.1 TISQLCommand Members

The following tables list the members exposed by TISQLCommand.

Public Methods

	Name	Description
	close (see page 1482)	This is close, a member of class TISQLCommand.
	Destroy (see page 1482)	This is Destroy, a member of class TISQLCommand.
	execute (see page 1483)	This is execute, a member of class TISQLCommand.

	executeImmediate (see page 1483)	This is executeImmediate, a member of class TISQLCommand.
	getErrorMessage (see page 1483)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1484)	This is getErrorMessageLen, a member of class TISQLCommand.
	getNextCursor (see page 1484)	This is getNextCursor, a member of class TISQLCommand.
	GetOption (see page 1484)	This is GetOption, a member of class TISQLCommand.
	getParameter (see page 1484)	This is getParameter, a member of class TISQLCommand.
	getRowsAffected (see page 1485)	This is getRowsAffected, a member of class TISQLCommand.
	GetStringOption (see page 1485)	This is GetStringOption, a member of class TISQLCommand.
	prepare (see page 1485)	This is prepare, a member of class TISQLCommand.
	SetOption (see page 1485)	This is SetOption, a member of class TISQLCommand.
	setParameter (see page 1486)	This is setParameter, a member of class TISQLCommand.
	SetStringOption (see page 1486)	This is SetStringOption, a member of class TISQLCommand.

24.1.4.2 TISQLCommand.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TISQLCommand();
```

Description

This is Destroy, a member of class TISQLCommand.

24.1.4.3 TISQLCommand Methods

The methods of the TISQLCommand class are listed here.

Public Methods

	Name	Description
	close (see page 1482)	This is close, a member of class TISQLCommand.
	execute (see page 1483)	This is execute, a member of class TISQLCommand.
	executeImmediate (see page 1483)	This is executeImmediate, a member of class TISQLCommand.
	getErrorMessage (see page 1483)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1484)	This is getErrorMessageLen, a member of class TISQLCommand.
	getNextCursor (see page 1484)	This is getNextCursor, a member of class TISQLCommand.
	GetOption (see page 1484)	This is GetOption, a member of class TISQLCommand.
	getParameter (see page 1484)	This is getParameter, a member of class TISQLCommand.
	getRowsAffected (see page 1485)	This is getRowsAffected, a member of class TISQLCommand.
	GetStringOption (see page 1485)	This is GetStringOption, a member of class TISQLCommand.
	prepare (see page 1485)	This is prepare, a member of class TISQLCommand.
	SetOption (see page 1485)	This is SetOption, a member of class TISQLCommand.
	setParameter (see page 1486)	This is setParameter, a member of class TISQLCommand.
	SetStringOption (see page 1486)	This is SetStringOption, a member of class TISQLCommand.

24.1.4.3.1 TISQLCommand.close Method

Pascal

```
function close: SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult close() = 0;
```

Description

This is close, a member of class TISQLCommand.

24.1.4.3.2 TISQLCommand.execute Method

Pascal

```
function execute(var Cursor: TISQLCursor): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult execute(TISQLCursor Cursor) = 0;
```

Description

This is execute, a member of class TISQLCommand.

24.1.4.3.3 TISQLCommand.executeImmediate Method

Pascal

```
function executeImmediate(SQL: PWideChar; var Cursor: TISQLCursor): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult executeImmediate(PWideChar SQL, TISQLCursor Cursor) = 0;
```

Description

This is executeImmediate, a member of class TISQLCommand.

24.1.4.3.4 getErrorMessage Method

This is the overview for the getErrorMessage method overload.

Overload List

	Name	Description
	TISQLCommand.getErrorMessage (PWideChar) (see page 1483)	This is getErrorMessage, a member of class TISQLCommand.
	TISQLCommand.getErrorMessage (WideString) (see page 1484)	This is getErrorMessage, a member of class TISQLCommand.

24.1.4.3.4.1 TISQLCommand.getErrorMessage Method (PWideChar)

Pascal

```
function getErrorMessage(Error: PWideChar): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(PWideChar Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLCommand.

24.1.4.3.4.2 TISQLCommand.getErrorMessage Method (WideString)

Pascal

```
function getErrorMessage(var Error: WideString): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(BSTR Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLCommand.

24.1.4.3.5 TISQLCommand.getErrorMessageLen Method

Pascal

```
function getErrorMessageLen(out ErrorLen: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getErrorMessageLen(SmallInt ErrorLen) = 0;
```

Description

This is getErrorMessageLen, a member of class TISQLCommand.

24.1.4.3.6 TISQLCommand.getNextCursor Method

Pascal

```
function getNextCursor(var Cursor: TISQLCursor): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getNextCursor(TISQLCursor Cursor) = 0;
```

Description

This is getNextCursor, a member of class TISQLCommand.

24.1.4.3.7 TISQLCommand.GetOption Method

Pascal

```
function GetOption(eSqlCommandOption: TSQLCommandOption; PropValue: Pointer; MaxLength:
SmallInt; out Length: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetOption(TSQLCommandOption eSqlCommandOption, void * PropValue,
SmallInt MaxLength, SmallInt Length) = 0;
```

Description

This is GetOption, a member of class TISQLCommand.

24.1.4.3.8 TISQLCommand.getParameter Method

Pascal

```
function getParameter(ParameterNumber: Word; ulChildPos: Word; Value: Pointer; Length:
Integer; var IsBlank: Integer): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getParameter(Word ParameterNumber, Word ulChildPos, void * Value,
int Length, int IsBlank) = 0;
```

Description

This is getParameter, a member of class TISQLCommand.

24.1.4.3.9 TISQLCommand.getRowsAffected Method

Pascal

```
function getRowsAffected(var Rows: LongWord): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getRowsAffected(LongWord Rows) = 0;
```

Description

This is getRowsAffected, a member of class TISQLCommand.

24.1.4.3.10 TISQLCommand.GetStringOption Method

Pascal

```
function GetStringOption(eSqlCommandOption: TSQLCommandOption; var ulValue: WideString):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetStringOption(TSQLCommandOption eSqlCommandOption, BSTR
ulValue) = 0;
```

Description

This is GetStringOption, a member of class TISQLCommand.

24.1.4.3.11 TISQLCommand.prepare Method

Pascal

```
function prepare(SQL: PWideChar; ParamCount: Word): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult prepare(PWideChar SQL, Word ParamCount) = 0;
```

Description

This is prepare, a member of class TISQLCommand.

24.1.4.3.12 TISQLCommand.SetOption Method

Pascal

```
function SetOption(eSqlCommandOption: TSQLCommandOption; ulValue: Integer): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult SetOption(TSQLCommandOption eSqlCommandOption, int ulValue) = 0;
```

Description

This is SetOption, a member of class TISQLCommand.

24.1.4.3.13 TISQLCommand.setParameter Method

Pascal

```
function setParameter(ulParameter: Word; ulChildPos: Word; eParamType: TSTMTParamType;
uLogType: Word; uSubType: Word; iPrecision: Integer; iScale: Integer; Length: LongWord;
pBuffer: Pointer; lInd: Integer): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult setParameter(Word ulParameter, Word ulChildPos, TSTMTParamType
eParamType, Word uLogType, Word uSubType, int iPrecision, int iScale, LongWord Length, void *
pBuffer, int lInd) = 0;
```

Description

This is setParameter, a member of class TISQLCommand.

24.1.4.3.14 TISQLCommand.SetStringOption Method

Pascal

```
function SetStringOption(eSqlCommandOption: TSQLCommandOption; const ulValue: WideString):
SQLResult; virtual; abstract;
```

C++

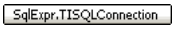
```
virtual __fastcall SQLResult SetStringOption(TSQLCommandOption eSqlCommandOption, const BSTR
ulValue) = 0;
```

Description

This is SetStringOption, a member of class TISQLCommand.

24.1.5 TISQLConnection Class

Class Hierarchy



Pascal

```
TISQLConnection = class;
```

C++

```
class TISQLConnection;
```

File

SqlExpr

Description

This is class SqlExpr.TISQLConnection.

24.1.5.1 TISQLConnection Members

The following tables list the members exposed by TISQLConnection.

Public Methods

	Name	Description
	beginTransaction (see page 1488)	This is beginTransaction, a member of class TISQLConnection.
	commit (see page 1488)	This is commit, a member of class TISQLConnection.
	connect (see page 1488)	This is the overview for the connect method overload.
	Create (see page 1487)	This is Create, a member of class TISQLConnection.
	disconnect (see page 1489)	This is disconnect, a member of class TISQLConnection.
	getErrorMessage (see page 1489)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1490)	This is getErrorMessageLen, a member of class TISQLConnection.
	getFldDescClass (see page 1490)	This is getFldDescClass, a member of class TISQLConnection.
	GetOption (see page 1490)	This is GetOption, a member of class TISQLConnection.
	getSQLCommand (see page 1490)	This is getSQLCommand, a member of class TISQLConnection.
	getSQLMetaData (see page 1490)	This is getSQLMetaData, a member of class TISQLConnection.
	GetStringOption (see page 1491)	This is GetStringOption, a member of class TISQLConnection.
	rollback (see page 1491)	This is rollback, a member of class TISQLConnection.
	SetOption (see page 1491)	This is SetOption, a member of class TISQLConnection.
	SetStringOption (see page 1491)	This is SetStringOption, a member of class TISQLConnection.

24.1.5.2 TISQLConnection.Create Constructor

Pascal

```
constructor Create(NewConnection: ISQLConnection); virtual; abstract;
```

C++

```
virtual __fastcall TISQLConnection(ISQLConnection NewConnection) = 0;
```

Description

This is Create, a member of class TISQLConnection.

24.1.5.3 TISQLConnection Methods

The methods of the TISQLConnection class are listed here.

Public Methods

	Name	Description
	beginTransaction (see page 1488)	This is beginTransaction, a member of class TISQLConnection.
	commit (see page 1488)	This is commit, a member of class TISQLConnection.
	connect (see page 1488)	This is the overview for the connect method overload.
	disconnect (see page 1489)	This is disconnect, a member of class TISQLConnection.
	getErrorMessage (see page 1489)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1490)	This is getErrorMessageLen, a member of class TISQLConnection.
	getFldDescClass (see page 1490)	This is getFldDescClass, a member of class TISQLConnection.
	GetOption (see page 1490)	This is GetOption, a member of class TISQLConnection.
	getSQLCommand (see page 1490)	This is getSQLCommand, a member of class TISQLConnection.
	getSQLMetaData (see page 1490)	This is getSQLMetaData, a member of class TISQLConnection.
	GetStringOption (see page 1491)	This is GetStringOption, a member of class TISQLConnection.

	rollback (see page 1491)	This is rollback, a member of class TISQLConnection.
	SetOption (see page 1491)	This is SetOption, a member of class TISQLConnection.
	SetStringOption (see page 1491)	This is SetStringOption, a member of class TISQLConnection.

24.1.5.3.1 TISQLConnection.beginTransaction Method

Pascal

```
function beginTransaction(TranID: LongWord): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult beginTransaction(LongWord TranID) = 0;
```

Description

This is beginTransaction, a member of class TISQLConnection.

24.1.5.3.2 TISQLConnection.commit Method

Pascal

```
function commit(TranID: LongWord): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult commit(LongWord TranID) = 0;
```

Description

This is commit, a member of class TISQLConnection.

24.1.5.3.3 connect Method

This is the overview for the connect method overload.

Overload List

	Name	Description
	TISQLConnection.connect () (see page 1488)	This is connect, a member of class TISQLConnection.
	TISQLConnection.connect (PWideChar, PWideChar, PWideChar) (see page 1488)	This is connect, a member of class TISQLConnection.

24.1.5.3.3.1 TISQLConnection.connect Method ()

Pascal

```
function connect: SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult connect() = 0;
```

Description

This is connect, a member of class TISQLConnection.

24.1.5.3.3.2 TISQLConnection.connect Method (PWideChar, PWideChar, PWideChar)

Pascal

```
function connect(ServerName: PWideChar; UserName: PWideChar; Password: PWideChar): SQLResult;
```

```
virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult connect(PWideChar ServerName, PWideChar UserName, PWideChar
Password) = 0;
```

Description

This is connect, a member of class TISQLConnection.

24.1.5.3.4 TISQLConnection.disconnect Method

Pascal

```
function disconnect: SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult disconnect() = 0;
```

Description

This is disconnect, a member of class TISQLConnection.

24.1.5.3.5 getErrorMessage Method

This is the overview for the getErrorMessage method overload.

Overload List

	Name	Description
	TISQLConnection.getErrorMessage (PWideChar) (see page 1489)	This is getErrorMessage, a member of class TISQLConnection.
	TISQLConnection.getErrorMessage (WideString) (see page 1489)	This is getErrorMessage, a member of class TISQLConnection.

24.1.5.3.5.1 TISQLConnection.getErrorMessage Method (PWideChar)

Pascal

```
function getErrorMessage(Error: PWideChar): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(PWideChar Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLConnection.

24.1.5.3.5.2 TISQLConnection.getErrorMessage Method (WideString)

Pascal

```
function getErrorMessage(var Error: WideString): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(BSTR Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLConnection.

24.1.5.3.6 TISQLConnection.getErrorMessageLen Method

Pascal

```
function getErrorMessageLen(out ErrorLen: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getErrorMessageLen(SmallInt ErrorLen) = 0;
```

Description

This is getErrorMessageLen, a member of class TISQLConnection.

24.1.5.3.7 TISQLConnection.getFldDescClass Method

Pascal

```
function getFldDescClass: TFldDescRef; virtual; abstract;
```

C++

```
virtual __fastcall TFldDescRef getFldDescClass() = 0;
```

Description

This is getFldDescClass, a member of class TISQLConnection.

24.1.5.3.8 TISQLConnection.GetOption Method

Pascal

```
function GetOption(eDOption: TSQLConnectionOption; PropValue: Pointer; MaxLength: SmallInt;
out Length: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetOption(TSQLConnectionOption eDOption, void * PropValue,
SmallInt MaxLength, SmallInt Length) = 0;
```

Description

This is GetOption, a member of class TISQLConnection.

24.1.5.3.9 TISQLConnection.getSQLCommand Method

Pascal

```
function getSQLCommand(var pComm: TISQLCommand): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getSQLCommand(TISQLCommand pComm) = 0;
```

Description

This is getSQLCommand, a member of class TISQLConnection.

24.1.5.3.10 TISQLConnection.getSQLMetaData Method

Pascal

```
function getSQLMetaData(var pMetaData: TISQLMetaData): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getSQLMetaData(TISQLMetaData pMetaData) = 0;
```

Description

This is getSQLMetaData, a member of class TISQLConnection.

24.1.5.3.11 TISQLConnection.GetStringOption Method

Pascal

```
function GetStringOption(eDOption: TSQLConnectionOption; var str: WideString): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetStringOption(TSQLConnectionOption eDOption, BSTR str) = 0;
```

Description

This is GetStringOption, a member of class TISQLConnection.

24.1.5.3.12 TISQLConnection.rollback Method

Pascal

```
function rollback(TranID: LongWord): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult rollback(LongWord TranID) = 0;
```

Description

This is rollback, a member of class TISQLConnection.

24.1.5.3.13 TISQLConnection.SetOption Method

Pascal

```
function SetOption(eConnectOption: TSQLConnectionOption; lValue: LongInt): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult SetOption(TSQLConnectionOption eConnectOption, LongInt lValue) =
0;
```

Description

This is SetOption, a member of class TISQLConnection.

24.1.5.3.14 TISQLConnection.SetStringOption Method

Pascal

```
function SetStringOption(eConnectOption: TSQLConnectionOption; const lValue: WideString):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult SetStringOption(TSQLConnectionOption eConnectOption, const BSTR
lValue) = 0;
```

Description

This is SetStringOption, a member of class TISQLConnection.

24.1.6 TISQLCursor Class

Class Hierarchy

Pascal

```
TISQLCursor = class;
```

C++

```
class TISQLCursor;
```

File

SqlExpr

Description

This is class SqlExpr.TISQLCursor.

24.1.6.1 TISQLCursor Members

The following tables list the members exposed by TISQLCursor.

Public Methods

	Name	Description
	getBcd (see page 1494)	This is getBcd, a member of class TISQLCursor.
	getBlob (see page 1494)	This is getBlob, a member of class TISQLCursor.
	getBlobSize (see page 1494)	This is getBlobSize, a member of class TISQLCursor.
	getBytes (see page 1494)	This is getBytes, a member of class TISQLCursor.
	getColumnCount (see page 1495)	This is getColumnCount, a member of class TISQLCursor.
	getColumnLength (see page 1495)	This is getColumnLength, a member of class TISQLCursor.
	getColumnName (see page 1495)	This is the overview for the getColumnName method overload.
	getColumnNameLength (see page 1496)	This is getColumnNameLength, a member of class TISQLCursor.
	getColumnPrecision (see page 1496)	This is getColumnPrecision, a member of class TISQLCursor.
	getColumnScale (see page 1496)	This is getColumnScale, a member of class TISQLCursor.
	getColumnType (see page 1496)	This is getColumnType, a member of class TISQLCursor.
	getCurObjectTypeName (see page 1497)	This is getCurObjectTypeName, a member of class TISQLCursor.
	getDate (see page 1497)	This is getDate, a member of class TISQLCursor.
	getDouble (see page 1497)	This is getDouble, a member of class TISQLCursor.
	getErrorMessage (see page 1497)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1498)	This is getErrorMessageLen, a member of class TISQLCursor.
	getInt64 (see page 1498)	This is getInt64, a member of class TISQLCursor.
	getLong (see page 1498)	This is getLong, a member of class TISQLCursor.
	GetOption (see page 1499)	This is GetOption, a member of class TISQLCursor.
	getShort (see page 1499)	This is getShort, a member of class TISQLCursor.
	getString (see page 1499)	This is getString, a member of class TISQLCursor.
	GetStringOption (see page 1499)	This is GetStringOption, a member of class TISQLCursor.
	getTime (see page 1500)	This is getTime, a member of class TISQLCursor.

	getTimeStamp (see page 1500)	This is getTimeStamp, a member of class TISQLCursor.
	getWideString (see page 1500)	This is getWideString, a member of class TISQLCursor.
	isAutoIncrement (see page 1500)	This is isAutoIncrement, a member of class TISQLCursor.
	isBlobSizeExact (see page 1501)	This is isBlobSizeExact, a member of class TISQLCursor.
	isNullable (see page 1501)	This is isNullable, a member of class TISQLCursor.
	isReadOnly (see page 1501)	This is isReadOnly, a member of class TISQLCursor.
	isSearchable (see page 1501)	This is isSearchable, a member of class TISQLCursor.
	next (see page 1501)	This is next, a member of class TISQLCursor.
	SetOption (see page 1502)	This is SetOption, a member of class TISQLCursor.
	SetStringOption (see page 1502)	This is SetStringOption, a member of class TISQLCursor.

24.1.6.2 TISQLCursor Methods

The methods of the TISQLCursor class are listed here.

Public Methods

	Name	Description
	getBcd (see page 1494)	This is getBcd, a member of class TISQLCursor.
	getBlob (see page 1494)	This is getBlob, a member of class TISQLCursor.
	getBlobSize (see page 1494)	This is getBlobSize, a member of class TISQLCursor.
	getBytes (see page 1494)	This is getBytes, a member of class TISQLCursor.
	getColumnCount (see page 1495)	This is getColumnCount, a member of class TISQLCursor.
	getColumnLength (see page 1495)	This is getColumnLength, a member of class TISQLCursor.
	getColumnName (see page 1495)	This is the overview for the getColumnName method overload.
	getColumnNameLength (see page 1496)	This is getColumnNameLength, a member of class TISQLCursor.
	getColumnPrecision (see page 1496)	This is getColumnPrecision, a member of class TISQLCursor.
	getColumnScale (see page 1496)	This is getColumnScale, a member of class TISQLCursor.
	getColumnType (see page 1496)	This is getColumnType, a member of class TISQLCursor.
	getCurObjectTypeName (see page 1497)	This is getCurObjectTypeName, a member of class TISQLCursor.
	getDate (see page 1497)	This is getDate, a member of class TISQLCursor.
	getDouble (see page 1497)	This is getDouble, a member of class TISQLCursor.
	getErrorMessage (see page 1497)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1498)	This is getErrorMessageLen, a member of class TISQLCursor.
	getInt64 (see page 1498)	This is getInt64, a member of class TISQLCursor.
	getLong (see page 1498)	This is getLong, a member of class TISQLCursor.
	GetOption (see page 1499)	This is GetOption, a member of class TISQLCursor.
	getShort (see page 1499)	This is getShort, a member of class TISQLCursor.
	getString (see page 1499)	This is getString, a member of class TISQLCursor.
	GetStringOption (see page 1499)	This is GetStringOption, a member of class TISQLCursor.
	getTime (see page 1500)	This is getTime, a member of class TISQLCursor.
	getTimeStamp (see page 1500)	This is getTimeStamp, a member of class TISQLCursor.
	getWideString (see page 1500)	This is getWideString, a member of class TISQLCursor.
	isAutoIncrement (see page 1500)	This is isAutoIncrement, a member of class TISQLCursor.
	isBlobSizeExact (see page 1501)	This is isBlobSizeExact, a member of class TISQLCursor.
	isNullable (see page 1501)	This is isNullable, a member of class TISQLCursor.
	isReadOnly (see page 1501)	This is isReadOnly, a member of class TISQLCursor.
	isSearchable (see page 1501)	This is isSearchable, a member of class TISQLCursor.
	next (see page 1501)	This is next, a member of class TISQLCursor.
	SetOption (see page 1502)	This is SetOption, a member of class TISQLCursor.
	SetStringOption (see page 1502)	This is SetStringOption, a member of class TISQLCursor.

24.1.6.2.1 TISQLCursor.getBcd Method

Pascal

```
function getBcd(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getBcd(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getBcd, a member of class TISQLCursor.

24.1.6.2.2 TISQLCursor.getBlob Method

Pascal

```
function getBlob(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool; Length: LongWord):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getBlob(Word ColumnNumber, void * Value, LongBool IsBlank,
LongWord Length) = 0;
```

Description

This is getBlob, a member of class TISQLCursor.

24.1.6.2.3 TISQLCursor.getBlobSize Method

Pascal

```
function getBlobSize(ColumnNumber: Word; var Length: LongWord; var IsBlank: LongBool):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getBlobSize(Word ColumnNumber, LongWord Length, LongBool IsBlank)
= 0;
```

Description

This is getBlobSize, a member of class TISQLCursor.

24.1.6.2.4 TISQLCursor.getBytes Method

Pascal

```
function getBytes(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getBytes(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getBytes, a member of class TISQLCursor.

24.1.6.2.5 TISQLCursor.getColumnCount Method

Pascal

```
function getColumnCount(var pColumns: Word): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getColumnCount(Word pColumns) = 0;
```

Description

This is getColumnCount, a member of class TISQLCursor.

24.1.6.2.6 TISQLCursor.getColumnLength Method

Pascal

```
function getColumnLength(ColumnNumber: Word; var pLength: LongWord): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult getColumnLength(Word ColumnNumber, LongWord pLength) = 0;
```

Description

This is getColumnLength, a member of class TISQLCursor.

24.1.6.2.7 getColumnName Method

This is the overview for the getColumnName method overload.

Overload List

	Name	Description
	TISQLCursor.getColumnName (Word) (see page 1495)	This is getColumnName, a member of class TISQLCursor.
	TISQLCursor.getColumnName (Word, PWideChar) (see page 1495)	This is getColumnName, a member of class TISQLCursor.

24.1.6.2.7.1 TISQLCursor.getColumnName Method (Word)

Pascal

```
function getColumnName(ColumnNumber: Word): WideString; virtual; abstract; overload;
```

C++

```
virtual __fastcall BSTR getColumnName(Word ColumnNumber) = 0;
```

Description

This is getColumnName, a member of class TISQLCursor.

24.1.6.2.7.2 TISQLCursor.getColumnName Method (Word, PWideChar)

Pascal

```
function getColumnName(ColumnNumber: Word; pColumnName: PWideChar): SQLResult; virtual;
abstract; overload;
```

C++

```
virtual __fastcall SQLResult getColumnName(Word ColumnNumber, PWideChar pColumnName) = 0;
```

Description

This is getColumnName, a member of class TISQLCursor.

24.1.6.2.8 TISQLCursor.getColumnNameLength Method

Pascal

```
function getColumnNameLength(ColumnNumber: Word; var pLen: Word): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getColumnNameLength(Word ColumnNumber, Word pLen) = 0;
```

Description

This is getColumnNameLength, a member of class TISQLCursor.

24.1.6.2.9 TISQLCursor.getColumnPrecision Method

Pascal

```
function getColumnPrecision(ColumnNumber: Word; var piPrecision: SmallInt): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getColumnPrecision(Word ColumnNumber, SmallInt piPrecision) = 0;
```

Description

This is getColumnPrecision, a member of class TISQLCursor.

24.1.6.2.10 TISQLCursor.getColumnScale Method

Pascal

```
function getColumnScale(ColumnNumber: Word; var piScale: SmallInt): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult getColumnScale(Word ColumnNumber, SmallInt piScale) = 0;
```

Description

This is getColumnScale, a member of class TISQLCursor.

24.1.6.2.11 TISQLCursor.getColumnType Method

Pascal

```
function getColumnType(ColumnNumber: Word; var puType: Word; var puSubType: Word): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getColumnType(Word ColumnNumber, Word puType, Word puSubType) = 0;
```

Description

This is getColumnType, a member of class TISQLCursor.

24.1.6.2.12 TISQLCursor.getCurObjectTypeName Method

Pascal

```
function getCurObjectTypeName(const iFldNum: Word): WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR getCurObjectTypeName(const Word iFldNum) = 0;
```

Description

This is getCurObjectTypeName, a member of class TISQLCursor.

24.1.6.2.13 TISQLCursor.getDate Method

Pascal

```
function getDate(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getDate(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getDate, a member of class TISQLCursor.

24.1.6.2.14 TISQLCursor.getDouble Method

Pascal

```
function getDouble(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getDouble(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getDouble, a member of class TISQLCursor.

24.1.6.2.15 getErrorMessage Method

This is the overview for the getErrorMessage method overload.

Overload List

	Name	Description
	TISQLCursor.getErrorMessage (PWideChar) (see page 1497)	This is getErrorMessage, a member of class TISQLCursor.
	TISQLCursor.getErrorMessage (WideString) (see page 1498)	This is getErrorMessage, a member of class TISQLCursor.

24.1.6.2.15.1 TISQLCursor.getErrorMessage Method (PWideChar)

Pascal

```
function getErrorMessage(Error: PWideChar): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(PWideChar Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLCursor.

24.1.6.2.15.2 TISQLCursor.getErrorMessage Method (WideString)

Pascal

```
function getErrorMessage(var Error: WideString): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(BSTR Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLCursor.

24.1.6.2.16 TISQLCursor.getErrorMessageLen Method

Pascal

```
function getErrorMessageLen(out ErrorLen: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getErrorMessageLen(SmallInt ErrorLen) = 0;
```

Description

This is getErrorMessageLen, a member of class TISQLCursor.

24.1.6.2.17 TISQLCursor.getInt64 Method

Pascal

```
function getInt64(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getInt64(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getInt64, a member of class TISQLCursor.

24.1.6.2.18 TISQLCursor.getLong Method

Pascal

```
function getLong(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getLong(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getLong, a member of class TISQLCursor.

24.1.6.2.19 TISQLCursor.GetOption Method

Pascal

```
function GetOption(eOption: TSQLCursorOption; PropValue: Pointer; MaxLength: SmallInt; out
Length: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetOption(TSQLCursorOption eOption, void * PropValue, SmallInt
MaxLength, SmallInt Length) = 0;
```

Description

This is GetOption, a member of class TISQLCursor.

24.1.6.2.20 TISQLCursor.getShort Method

Pascal

```
function getShort(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getShort(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getShort, a member of class TISQLCursor.

24.1.6.2.21 TISQLCursor.getString Method

Pascal

```
function getString(ColumnNumber: Word; Value: PChar; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getString(Word ColumnNumber, const char * Value, LongBool
IsBlank) = 0;
```

Description

This is getString, a member of class TISQLCursor.

24.1.6.2.22 TISQLCursor.GetStringOption Method

Pascal

```
function GetStringOption(eOption: TSQLCursorOption; var str: WideString): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult GetStringOption(TSQLCursorOption eOption, BSTR str) = 0;
```

Description

This is GetStringOption, a member of class TISQLCursor.

24.1.6.2.23 TISQLCursor.getTime Method

Pascal

```
function getTime(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getTime(Word ColumnNumber, void * Value, LongBool IsBlank) = 0;
```

Description

This is getTime, a member of class TISQLCursor.

24.1.6.2.24 TISQLCursor.getTimeStamp Method

Pascal

```
function getTimeStamp(ColumnNumber: Word; Value: Pointer; var IsBlank: LongBool): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getTimeStamp(Word ColumnNumber, void * Value, LongBool IsBlank) =
0;
```

Description

This is getTimeStamp, a member of class TISQLCursor.

24.1.6.2.25 TISQLCursor.getWideString Method

Pascal

```
function getWideString(ColumnNumber: Word; Value: PWideChar; var IsBlank: LongBool):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getWideString(Word ColumnNumber, PWideChar Value, LongBool
IsBlank) = 0;
```

Description

This is getWideString, a member of class TISQLCursor.

24.1.6.2.26 TISQLCursor.isAutoIncrement Method

Pascal

```
function isAutoIncrement(ColumnNumber: Word; var AutoIncr: LongBool): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult isAutoIncrement(Word ColumnNumber, LongBool AutoIncr) = 0;
```

Description

This is isAutoIncrement, a member of class TISQLCursor.

24.1.6.2.27 TISQLCursor.isBlobSizeExact Method

Pascal

```
function isBlobSizeExact(ColumnNumber: Word; var IsExact: LongBool): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult isBlobSizeExact(Word ColumnNumber, LongBool IsExact) = 0;
```

Description

This is isBlobSizeExact, a member of class TISQLCursor.

24.1.6.2.28 TISQLCursor.isNullable Method

Pascal

```
function isNullable(ColumnNumber: Word; var Nullable: LongBool): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult isNullable(Word ColumnNumber, LongBool Nullable) = 0;
```

Description

This is isNullable, a member of class TISQLCursor.

24.1.6.2.29 TISQLCursor.isReadOnly Method

Pascal

```
function isReadOnly(ColumnNumber: Word; var ReadOnly: LongBool): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult isReadOnly(Word ColumnNumber, LongBool ReadOnly) = 0;
```

Description

This is isReadOnly, a member of class TISQLCursor.

24.1.6.2.30 TISQLCursor.isSearchable Method

Pascal

```
function isSearchable(ColumnNumber: Word; var Searchable: LongBool): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult isSearchable(Word ColumnNumber, LongBool Searchable) = 0;
```

Description

This is isSearchable, a member of class TISQLCursor.

24.1.6.2.31 TISQLCursor.next Method

Pascal

```
function next: SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult next() = 0;
```

Description

This is next, a member of class TISQLCursor.

24.1.6.2.32 TISQLCursor.SetOption Method

Pascal

```
function SetOption(eOption: TSQLCursorOption; PropValue: LongInt): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult SetOption(TSQLCursorOption eOption, LongInt PropValue) = 0;
```

Description

This is SetOption, a member of class TISQLCursor.

24.1.6.2.33 TISQLCursor.SetStringOption Method

Pascal

```
function SetStringOption(eOption: TSQLCursorOption; const str: WideString): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult SetStringOption(TSQLCursorOption eOption, const BSTR str) = 0;
```

Description

This is SetStringOption, a member of class TISQLCursor.

24.1.7 TISQLMetaData Class

Class Hierarchy

Pascal

```
TISQLMetaData = class;
```

C++

```
class TISQLMetaData;
```

File

SqlExpr

Description

This is class SqlExpr.TISQLMetaData.

24.1.7.1 TISQLMetaData Members

The following tables list the members exposed by TISQLMetaData.

Public Methods

	Name	Description
	getColumns (see page 1503)	This is getColumns, a member of class TISQLMetaData.
	getErrorMessage (see page 1504)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1504)	This is getErrorMessageLen, a member of class TISQLMetaData.
	getIndices (see page 1504)	This is getIndices, a member of class TISQLMetaData.
	getObjectList (see page 1505)	This is getObjectList, a member of class TISQLMetaData.
	GetOption (see page 1505)	This is GetOption, a member of class TISQLMetaData.
	getProcedureParams (see page 1505)	This is getProcedureParams, a member of class TISQLMetaData.
	getProcedures (see page 1505)	This is getProcedures, a member of class TISQLMetaData.
	GetStringOption (see page 1506)	This is GetStringOption, a member of class TISQLMetaData.
	getTables (see page 1506)	This is getTables, a member of class TISQLMetaData.
	SetOption (see page 1506)	This is SetOption, a member of class TISQLMetaData.
	SetStringOption (see page 1506)	This is SetStringOption, a member of class TISQLMetaData.

24.1.7.2 TISQLMetaData Methods

The methods of the TISQLMetaData class are listed here.

Public Methods

	Name	Description
	getColumns (see page 1503)	This is getColumns, a member of class TISQLMetaData.
	getErrorMessage (see page 1504)	This is the overview for the getErrorMessage method overload.
	getErrorMessageLen (see page 1504)	This is getErrorMessageLen, a member of class TISQLMetaData.
	getIndices (see page 1504)	This is getIndices, a member of class TISQLMetaData.
	getObjectList (see page 1505)	This is getObjectList, a member of class TISQLMetaData.
	GetOption (see page 1505)	This is GetOption, a member of class TISQLMetaData.
	getProcedureParams (see page 1505)	This is getProcedureParams, a member of class TISQLMetaData.
	getProcedures (see page 1505)	This is getProcedures, a member of class TISQLMetaData.
	GetStringOption (see page 1506)	This is GetStringOption, a member of class TISQLMetaData.
	getTables (see page 1506)	This is getTables, a member of class TISQLMetaData.
	SetOption (see page 1506)	This is SetOption, a member of class TISQLMetaData.
	SetStringOption (see page 1506)	This is SetStringOption, a member of class TISQLMetaData.

24.1.7.2.1 TISQLMetaData.getColumns Method

Pascal

```
function getColumns(TableName: PWideChar; ColumnName: PWideChar; ColType: LongWord; var
Cursor: TISQLCursor): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getColumns(PWideChar TableName, PWideChar ColumnName, LongWord
ColType, TISQLCursor Cursor) = 0;
```

Description

This is getColumns, a member of class TISQLMetaData.

24.1.7.2.2 getErrorMessage Method

This is the overview for the getErrorMessage method overload.

Overload List

	Name	Description
	TISQLMetaData.getErrorMessage (PWideChar) (see page 1504)	This is getErrorMessage, a member of class TISQLMetaData.
	TISQLMetaData.getErrorMessage (WideString) (see page 1504)	This is getErrorMessage, a member of class TISQLMetaData.

24.1.7.2.2.1 TISQLMetaData.getErrorMessage Method (PWideChar)

Pascal

```
function getErrorMessage(Error: PWideChar): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(PWideChar Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLMetaData.

24.1.7.2.2.2 TISQLMetaData.getErrorMessage Method (WideString)

Pascal

```
function getErrorMessage(var Error: WideString): SQLResult; virtual; abstract; overload;
```

C++

```
virtual __fastcall SQLResult getErrorMessage(BSTR Error) = 0;
```

Description

This is getErrorMessage, a member of class TISQLMetaData.

24.1.7.2.3 TISQLMetaData.getErrorMessageLen Method

Pascal

```
function getErrorMessageLen(out ErrorLen: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getErrorMessageLen(SmallInt ErrorLen) = 0;
```

Description

This is getErrorMessageLen, a member of class TISQLMetaData.

24.1.7.2.4 TISQLMetaData.getIndices Method

Pascal

```
function getIndices(TableName: PWideChar; IndexType: LongWord; var Cursor: TISQLCursor):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getIndices(PWideChar TableName, LongWord IndexType, TISQLCursor
Cursor) = 0;
```

Description

This is getIndices, a member of class TISQLMetaData.

24.1.7.2.5 TISQLMetaData.getObjectList Method

Pascal

```
function getObjectList(eObjType: TSQLObjectType; var Cursor: TISQLCursor): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult getObjectList(TSQLObjectType eObjType, TISQLCursor Cursor) = 0;
```

Description

This is getObjectList, a member of class TISQLMetaData.

24.1.7.2.6 TISQLMetaData.GetOption Method

Pascal

```
function GetOption(eDOption: TSQLMetaDataOption; PropValue: Pointer; MaxLength: SmallInt; out
Length: SmallInt): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetOption(TSQLMetaDataOption eDOption, void * PropValue, SmallInt
MaxLength, SmallInt Length) = 0;
```

Description

This is GetOption, a member of class TISQLMetaData.

24.1.7.2.7 TISQLMetaData.getProcedureParams Method

Pascal

```
function getProcedureParams(ProcName: PWideChar; ParamName: PWideChar; var Cursor:
TISQLCursor): SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getProcedureParams(PWideChar ProcName, PWideChar ParamName,
TISQLCursor Cursor) = 0;
```

Description

This is getProcedureParams, a member of class TISQLMetaData.

24.1.7.2.8 TISQLMetaData.getProcedures Method

Pascal

```
function getProcedures(ProcedureName: PWideChar; ProcType: LongWord; var Cursor: TISQLCursor):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getProcedures(PWideChar ProcedureName, LongWord ProcType,
TISQLCursor Cursor) = 0;
```

Description

This is getProcedures, a member of class TISQLMetaData.

24.1.7.2.9 TISQLMetaData.GetStringOption Method

Pascal

```
function GetStringOption(eDOption: TSQLMetaDataOption; var str: WideString): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult GetStringOption(TSQLMetaDataOption eDOption, BSTR str) = 0;
```

Description

This is GetStringOption, a member of class TISQLMetaData.

24.1.7.2.10 TISQLMetaData.getTables Method

Pascal

```
function getTables(TableName: PWideChar; TableType: LongWord; var Cursor: TISQLCursor):
SQLResult; virtual; abstract;
```

C++

```
virtual __fastcall SQLResult getTables(PWideChar TableName, LongWord TableType, TISQLCursor
Cursor) = 0;
```

Description

This is getTables, a member of class TISQLMetaData.

24.1.7.2.11 TISQLMetaData.SetOption Method

Pascal

```
function SetOption(eDOption: TSQLMetaDataOption; PropValue: LongInt): SQLResult; virtual;
abstract;
```

C++

```
virtual __fastcall SQLResult SetOption(TSQLMetaDataOption eDOption, LongInt PropValue) = 0;
```

Description

This is SetOption, a member of class TISQLMetaData.

24.1.7.2.12 TISQLMetaData.SetStringOption Method

Pascal

```
function SetStringOption(eDOption: TSQLMetaDataOption; const str: WideString): SQLResult;
virtual; abstract;
```

C++

```
virtual __fastcall SQLResult SetStringOption(TSQLMetaDataOption eDOption, const BSTR str) = 0;
```

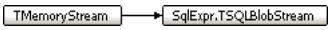
Description

This is SetStringOption, a member of class TISQLMetaData.

24.1.8 TSQLBlobStream Class

TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.

Class Hierarchy



Pascal

```
TSQLBlobStream = class(TMemoryStream);
```

C++

```
class TSQLBlobStream : public TMemoryStream;
```

File

SqlExpr

Description

Use TSQLBlobStream to read the value of a Binary Large Object (BLOB) field in a dbExpress dataset (TSQLLDataSet, TSQLTable (see page 1604), TSQLQuery (see page 1577), or TSQLStoredProc (see page 1594)). BLOB fields are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. BLOB fields in client datasets use client BLOB streams to implement many of their data access properties and methods.

TSQLBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field to read such data by employing the uniform stream interface.

To use an SQL BLOB stream, create an instance of TSQLBlobStream, use the methods of the stream to read the data, and then free the BLOB stream. If you use the same instance of TSQLBlobStream to access data from more than one record, you must call the ReadBlobData (see page 1509) method to update the stream when the current record changes.

Warning: Although TSQLBlobStream inherits several write methods from TMemoryStream, it can't be used to write data back to the dataset because dbExpress datasets are read-only. Writing to TSQLBlobStream simply alters the in-memory buffer that contains a BLOB field's data.

See Also

TBlobField

TGraphicField

TMemoField

TCustomSQLDataSet (see page 1421)

TSQLDataSet (see page 1556)

TSQLTable (see page 1604)

TSQLQuery (see page 1577)

TSQLStoredProc (see page 1594)

24.1.8.1 TSQLBlobStream Members

The following tables list the members exposed by TSQLBlobStream.

Public Methods

	Name	Description
	Create (see page 1508)	Creates and initializes an instance of TSQLBlobStream (see page 1507).
	Destroy (see page 1508)	Frees an instance of TSQLBlobStream (see page 1507).
	ReadBlobData (see page 1509)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

24.1.8.2 TSQLBlobStream.Create Constructor

Creates and initializes an instance of TSQLBlobStream (see page 1507).

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode = bmRead);
```

C++

```
__fastcall TSQLBlobStream(TBlobField Field, TBlobStreamMode Mode = bmRead);
```

Description

Call Create to obtain an instance of TSQLBlobStream (see page 1507) for reading from a specific TBlobField object in a TCustomSQLDataSet (see page 1421) descendant.

Field specifies the BLOB field from which to read data.

The Mode parameter is provided for compatibility with other BLOB stream constructors. However, in TSQLBlobStream (see page 1507), the Mode parameter is ignored because dbExpress datasets are read-only. SQL Blob streams can only be used to read BLOB data.

Create links the TSQLBlobStream (see page 1507) to the field object specified by the Field parameter. It then calls ReadBlobData (see page 1509) to read the information from the BLOB field into a memory buffer, where it is available for reading.

Warning: If you write to the SQLBlobStream using inherited methods, you are overwriting the in-memory buffer that is filled by the constructor. This will cause subsequent read operations to provide inaccurate information, because the BLOB stream no longer accurately reflects the contents of the BLOB field.

Instead of calling the TSQLBlobStream (see page 1507) constructor, you can also use the dataset's CreateBlobStream method.

See Also

CreateBlobStream (see page 1429)

ReadBlobData (see page 1509)

24.1.8.3 TSQLBlobStream.Destroy Destructor

Frees an instance of TSQLBlobStream (see page 1507).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLBlobStream();
```

Description

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

Destroy the TSQLBlobStream (see page 1507) object by calling Free when it is no longer needed for reading from the BLOB field. BLOB streams should be destroyed when the dataset changes the current record.

See Also

System::TObject::Free (see page 3382)

OnDataChange

24.1.8.4 TSQLBlobStream Methods

The methods of the TSQLBlobStream class are listed here.

Public Methods

	Name	Description
	ReadBlobData (see page 1509)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

24.1.8.4.1 TSQLBlobStream.ReadBlobData Method

Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

Pascal

```
procedure ReadBlobData;
```

C++

```
__fastcall ReadBlobData();
```

Description

Call ReadBlobData to refresh the contents of the BLOB stream when the current record in the dataset changes. ReadBlobData reads the contents of a BLOB field into the SQL BLOB stream's in-memory buffer and sets the Size property to the number of bytes of BLOB data. It then sets Position to 0 so that the stream is ready for you to read the BLOB data.

Note: It is not necessary to call ReadBlobData when you first create a SQL BLOB stream, because the constructor already performs this task. You need only call ReadBlobData when the current record changes, so that the stream reflects the BLOB data for the new record.

See Also

Size

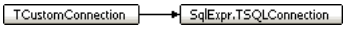
Position

Create (see page 1508)

24.1.9 TSQLConnection Class

TSQLConnection encapsulates a dbExpress connection to a database server.

Class Hierarchy



Pascal

```
TSQLConnection = class(TCustomConnection);
```

C++

```
class TSQLConnection : public TCustomConnection;
```

File

SqlExpr

Description

Use TSQLConnection to represent a dbExpress database connection. The connection provided by a single TSQLConnection component can be shared by multiple SQL dataset components through their SQLConnection (see page 1547) properties.

TSQLConnection interacts with a dbExpress driver and two files. The first, dbxdrivers.ini on Windows or dbxdrivers on Linux, lists the installed driver types (Interbase, Oracle, and so on) and, for each driver, lists the libraries (DLLs or shared objects) it requires as well as the default settings for all connection parameters. The second file, dbxconnections.ini on Windows or dbxconnections on Linux, lists named sets of connection configurations. Each configuration represents a set of settings for TSQLConnection and describes a particular database connection.

You can use TSQLConnection to choose one of the named configurations in dbxconnections.ini and use it to connect to a database or to define new configurations and add them to dbxconnections.ini. To perform either of these tasks, double-click on a TSQLConnection component and use the Connection (see page 1534) Builder.

In addition to describing and establishing database connections, TSQLConnection surfaces properties, methods, and events for controlling server login, managing transactions, and fetching metadata from the server.

See Also

TCustomSQLDataSet (see page 1421)

TSQLDataSet (see page 1556)

TSQLTable (see page 1604)

TSQLQuery (see page 1577)

TSQLStoredProc (see page 1594)

Connecting to Databases: Overview

Connecting Directly to a Database Server

Accessing Schema Information

Connecting to the Database Server

Setting Up TSQLConnection

24.1.9.1 TSQLConnection Members

The following tables list the members exposed by TSQLConnection.

Protected Methods

	Name	Description
	Check (see page 1514)	This is Check, a member of class TSQLConnection.
	CheckConnection (see page 1514)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 1515)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 1516)	This is ConnectionOptions, a member of class TSQLConnection.
V	DoConnect (see page 1516)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1517)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 1518)	Returns the value of the Connected property.
	GetDataSet (see page 1519)	Returns a dataset, given its index in the DataSets property array.
V	Loaded (see page 1526)	Initializes the connection component after all objects in the form or data module have been loaded.
	LoadSQLDll (see page 1527)	This is LoadSQLDll, a member of class TSQLConnection.
V	Notification (see page 1528)	Forwards notification messages to all owned components.
	OpenSchema (see page 1528)	This is the overview for the OpenSchema method overload.
	SQLError (see page 1530)	This is SQLError, a member of class TSQLConnection.

Protected Properties

	Name	Description
R	Connection (see page 1534)	This is Connection, a member of class TSQLConnection.
R	ConnectionRegistryFile (see page 1535)	This is ConnectionRegistryFile, a member of class TSQLConnection.
R	Driver (see page 1536)	This is Driver, a member of class TSQLConnection.
R	DriverRegistryFile (see page 1537)	This is DriverRegistryFile, a member of class TSQLConnection.
	LastError (see page 1538)	This is LastError, a member of class TSQLConnection.
R	QuoteChar (see page 1547)	This is QuoteChar, a member of class TSQLConnection.
	SQLDllHandle (see page 1547)	This is SQLDllHandle, a member of class TSQLConnection.

Public Methods

	Name	Description
	CloneConnection (see page 1515)	Returns a copy of the SQL connection object.
	CloseDataSets (see page 1515)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 1516)	Commits an open transaction.
V	Create (see page 1513)	Creates and initializes an instance TSQLConnection (see page 1510).
V	Destroy (see page 1513)	Destroys the SQL connection object.
	Execute (see page 1517)	Executes an SQL command on the server.
	ExecuteDirect (see page 1518)	Executes an SQL command that does not include parameters.
	GetDefaultSchemaName (see page 1519)	Returns the default schema name of the current object.
	GetFieldNames (see page 1519)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1520)	This is the overview for the GetIndexNames method overload.
	GetLoginUsername (see page 1521)	Returns the username that is logged in on the current connection.
	GetPackageNames (see page 1521)	This is the overview for the GetPackageNames method overload.
	GetProcedureNames (see page 1522)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 1524)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 1525)	This is the overview for the GetSchemaNames method overload.
	GetTableNames (see page 1525)	This is the overview for the GetTableNames method overload.
	LoadParamsFromIniFile (see page 1527)	Sets the DriverName (see page 1536) and Params (see page 1546) properties to the values associated with the ConnectionName (see page 1534) property.

	Rollback (see page 1529)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	SetTraceCallbackEvent (see page 1529)	This is SetTraceCallbackEvent, a member of class TSQLConnection.
	StartTransaction (see page 1530)	Initiates a new transaction in the associated database.

Public Properties

	Name	Description
R	ActiveStatements (see page 1532)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 1532)	Specifies whether the SQL connection automatically clones database connections when they are needed.
	ConnectionState (see page 1535)	Indicates the current state of the TSQLConnection (see page 1510) object.
R	DataSets (see page 1535)	Lists all active datasets for the connection component.
R	InTransaction (see page 1537)	Indicates whether a transaction is in progress.
	LocaleCode (see page 1540)	Identifies the locale that is used for sorting items from datasets that use this connection.
R	MaxStmtsPerConn (see page 1544)	Indicates the server limit on the number of statements supported by a single database connection.
R	MetaData (see page 1545)	Provides an interface to the dbExpress object that supplies server metadata.
R	MultipleTransactionsSupported (see page 1545)	Indicates whether the database server supports multiple transactions.
	ParamsLoaded (see page 1547)	Indicates when parameter values have been loaded from dbxconnections.ini.
	SQLConnection (see page 1547)	This is SQLConnection, a member of class TSQLConnection.
	SQLHourGlass (see page 1548)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
R	TraceCallbackEvent (see page 1549)	Provides access to the callback function that executes for every SQL command passed to or from the server.
R	TransactionsSupported (see page 1549)	Indicates whether the database server supports transactions.

Published Events

	Name	Description
	AfterConnect (see page 1550)	Occurs after a connection is established.
	AfterDisconnect (see page 1551)	Occurs after the connection closes.
	BeforeConnect (see page 1551)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1552)	Occurs immediately before the connection closes.
	OnLogin (see page 1552)	Occurs after the BeforeConnect (see page 1551) event and before a connection is established.

Published Properties

	Name	Description
	Connected (see page 1533)	Specifies whether the connection is active.
	ConnectionName (see page 1534)	Names the connection configuration.
	DriverName (see page 1536)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 1537)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 1538)	Specifies whether the connection stays active when no datasets are open.
	LibraryName (see page 1538)	Specifies the dbExpress driver that TSQLConnection (see page 1510) uses to establish a connection.
	LoadParamsOnConnect (see page 1539)	Specifies whether the TSQLConnection (see page 1510) component loads the configuration named by ConnectionName (see page 1534) immediately before connecting to the server.
	LoginPrompt (see page 1540)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 1546)	Lists the connection parameters.
	TableScope (see page 1548)	Indicates what types of tables are returned when fetching schema information about tables.
	VendorLib (see page 1550)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

24.1.9.2 TSQLConnection.Create Constructor

Creates and initializes an instance TSQLConnection (see page 1510).

Pascal

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

C++

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

Description

Use Create to instantiate TSQLConnection (see page 1510) at runtime. SQL connections added to forms or data modules at design time are created automatically.

The AOwner parameter is the component, typically a form or data module, that is responsible for freeing the SQL connection.

See Also

Destroy (see page 1513)

Owner

24.1.9.3 TSQLConnection.Destroy Destructor

Destroys the SQL connection object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLConnection();
```

Description

To free a TSQLConnection (see page 1510) instance, do not call Destroy. Instead, call Free. Free verifies that the ADO connection is not nil, and only then calls Destroy.

There is no need to explicitly free instances of TSQLConnection (see page 1510) that were created with an Owner.

See Also

Owner

Create (see page 1513)

delete

System::TObject::Free (see page 3382)

24.1.9.4 TSQLConnection Methods

The methods of the TSQLConnection class are listed here.

Protected Methods

	Name	Description
	Check (see page 1514)	This is Check, a member of class TSQLConnection.

	CheckConnection (see page 1514)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 1515)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 1516)	This is ConnectionOptions, a member of class TSQLConnection.
V	DoConnect (see page 1516)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1517)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 1518)	Returns the value of the Connected property.
	GetDataSet (see page 1519)	Returns a dataset, given its index in the DataSets property array.
V	Loaded (see page 1526)	Initializes the connection component after all objects in the form or data module have been loaded.
	LoadSQLDll (see page 1527)	This is LoadSQLDll, a member of class TSQLConnection.
V	Notification (see page 1528)	Forwards notification messages to all owned components.
	OpenSchema (see page 1528)	This is the overview for the OpenSchema method overload.
	SQLError (see page 1530)	This is SQLError, a member of class TSQLConnection.

Public Methods

	Name	Description
	CloneConnection (see page 1515)	Returns a copy of the SQL connection object.
	CloseDataSets (see page 1515)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 1516)	Commits an open transaction.
	Execute (see page 1517)	Executes an SQL command on the server.
	ExecuteDirect (see page 1518)	Executes an SQL command that does not include parameters.
	GetDefaultSchemaName (see page 1519)	Returns the default schema name of the current object.
	GetFieldNames (see page 1519)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1520)	This is the overview for the GetIndexNames method overload.
	GetLoginUsername (see page 1521)	Returns the username that is logged in on the current connection.
	GetPackageNames (see page 1521)	This is the overview for the GetPackageNames method overload.
	GetProcedureNames (see page 1522)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 1524)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 1525)	This is the overview for the GetSchemaNames method overload.
	GetTableNames (see page 1525)	This is the overview for the GetTableNames method overload.
	LoadParamsFromIniFile (see page 1527)	Sets the DriverName (see page 1536) and Params (see page 1546) properties to the values associated with the ConnectionName (see page 1534) property.
	Rollback (see page 1529)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	SetTraceCallbackEvent (see page 1529)	This is SetTraceCallbackEvent, a member of class TSQLConnection.
	StartTransaction (see page 1530)	Initiates a new transaction in the associated database.

24.1.9.4.1 TSQLConnection.Check Method

Pascal

```
function Check(status: SQLResult): SQLResult;
```

C++

```
__fastcall SQLResult Check(SQLResult status);
```

Description

This is Check, a member of class TSQLConnection.

24.1.9.4.2 TSQLConnection.CheckConnection Method

Pascal

```
procedure CheckConnection(eFlag: eConnectFlag);
```

C++

```
__fastcall CheckConnection(eConnectFlag eFlag);
```

Description

This is CheckConnection, a member of class TSQLConnection.

24.1.9.4.3 TSQLConnection.CheckDisconnect Method

Pascal

```
procedure CheckDisconnect; virtual;
```

C++

```
virtual __fastcall CheckDisconnect();
```

Description

This is CheckDisconnect, a member of class TSQLConnection.

24.1.9.4.4 TSQLConnection.CloneConnection Method

Returns a copy of the SQL connection object.

Pascal

```
function CloneConnection: TSQLConnection;
```

C++

```
__fastcall TSQLConnection CloneConnection();
```

Description

Call CloneConnection to create a duplicate of this SQLConnection (see page 1547) object. CloneConnection instantiates a new TSQLConnection (see page 1510) object and assigns its properties to match those of this SQLConnection (see page 1547) object.

The returned SQLConnection (see page 1547) is owned by this TSQLConnection (see page 1510). That is, this SQL connection component frees the clone from its own destructor.

See Also

AutoClone (see page 1532)

24.1.9.4.5 TSQLConnection.CloseDataSets Method

Closes all datasets associated with the SQL connection object component without disconnecting from the database.

Pascal

```
procedure CloseDataSets;
```

C++

```
__fastcall CloseDataSets();
```

Description

Call CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close (see page 3643), all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close (see page 3643) ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

See Also

Close

Close

DataSets (see page 1535)

KeepConnection (see page 1538)

24.1.9.4.6 TSQLConnection.Commit Method

Commits an open transaction.

Pascal

```
procedure Commit(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Commit(TTransactionDesc TransDesc);
```

Description

Call Commit to save any changes made during the transaction described by TransDesc and to end that transaction.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 1530) method when starting the transaction.

If the specified transaction is the only current transaction, Commit sets the InTransaction (see page 1537) property to false after committing the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 1537)

StartTransaction (see page 1530)

Rollback (see page 1529)

TransactionsSupported (see page 1549)

TransactionLevel (see page 1467)

24.1.9.4.7 TSQLConnection.ConnectionOptions Method

Pascal

```
procedure ConnectionOptions; virtual;
```

C++

```
virtual __fastcall ConnectionOptions();
```

Description

This is ConnectionOptions, a member of class TSQLConnection.

24.1.9.4.8 TSQLConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

24.1.9.4.9 TSQLConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

24.1.9.4.10 TSQLConnection.Execute Method

Executes an SQL command on the server.

Pascal

```
function Execute(const SQL: WideString; Params: TParams; ResultSet: Pointer = nil): Integer;
```

C++

```
__fastcall int Execute(const BSTR SQL, TParams Params, void * ResultSet = nil);
```

Description

Call Execute to execute a single command on the server without the overhead of using an SQL dataset. Execute is especially useful for statements that do not return a result set such as data definition language (DDL) statements.

SQL is the command to execute.

Params (see page 1546) is a TParams object that contains any parameters used by the SQL statement. Parameter binding is by index only (not by name), so the order of parameters is important and the order of the TParam objects in Params (see page 1546) corresponds to the order of the parameters in the SQL statement. Use properties and methods of TParams to create a TParams object, add TParam objects for each parameter, and assign parameter properties such as datatype and value. If the SQL statement does not include any parameters, pass a nil (Delphi) or NULL (C++) value for Params (see page 1546).

ResultSet is a pointer to a variable of type TCustomSQLDataSet (see page 1421)*. If the SQL statement returns a cursor, Execute creates a new TCustomSQLDataSet (see page 1421) instance and populates it with the result set. The new TCustomSQLDataSet (see page 1421) reference is returned as the value to which ResultSet points. If the SQL statement does not return a result set, pass a nil (Delphi) or NULL (C++) value for ResultSet. TPSResult is is set to Pointer for native code and TObject (see page 3364) for managed code.

Execute returns the number of records affected by executing the SQL statement.

Tip: If the SQL statement does not include any parameters, you can use the ExecuteDirect (see page 1518) method instead.

See Also

ExecSQL (see page 1560)

ExecuteDirect (see page 1518)

24.1.9.4.11 TSQLConnection.ExecuteDirect Method

Executes an SQL command that does not include parameters.

Pascal

```
function ExecuteDirect(const SQL: WideString): Integer;
```

C++

```
__fastcall int ExecuteDirect(const BSTR SQL);
```

Description

Call ExecuteDirect to execute a single parameterless command on the server without the overhead of using an SQL dataset or preparing the statement before it executes. If the statement ordinarily returns a cursor, ExecuteDirect executes the statement, but does not return a cursor.

SQL is the command to execute.

ExecuteDirect returns 0 if the command was successfully executed. Otherwise, it returns a dbExpress error code.

Note: To execute an SQL statement that includes parameters or that returns a cursor, use the Execute (see page 1517) method instead.

See Also

ExecSQL (see page 1560)

Execute (see page 1517)

24.1.9.4.12 TSQLConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

24.1.9.4.13 TSQLConnection.GetDataSet Method

Returns a dataset, given its index in the DataSets property array.

Pascal

```
function GetDataSet(Index: Integer): TCustomSQLDataSet; reintroduce;
```

C++

```
__fastcall TCustomSQLDataSet GetDataSet(int Index);
```

Description

GetDataSet is the protected read implementation of the DataSets property.

See Also

DataSets

24.1.9.4.14 TSQLConnection.GetDefaultSchemaName Method

Returns the default schema name of the current object.

Pascal

```
function GetDefaultSchemaName: WideString;
```

C++

```
__fastcall BSTR GetDefaultSchemaName();
```

24.1.9.4.15 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetFieldNames (string, string, TStrings) (see page 1519)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (string, TStrings) (see page 1519)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (WideString, TWideStrings) (see page 1519)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (WideString, WideString, TWideStrings) (see page 1519)	Populates a string list with the names of fields in a table.

24.1.9.4.15.1 TSQLConnection.GetFieldNames Method (WideString, TWideStrings)

Populates a string list with the names of fields in a table.

Pascal

```
procedure GetFieldNames(const TableName: string; List: TStrings); overload; deprecated;
procedure GetFieldNames(const TableName: string; SchemaName: string; List: TStrings);
overload; deprecated;
procedure GetFieldNames(const TableName: WideString; List: TWideStrings); overload;
procedure GetFieldNames(const TableName: WideString; SchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetFieldNames(const UnicodeString TableName, TStrings List);
__fastcall GetFieldNames(const UnicodeString TableName, UnicodeString SchemaName, TStrings
List);
__fastcall GetFieldNames(const BSTR TableName, TWideStrings List);
__fastcall GetFieldNames(const BSTR TableName, BSTR SchemaName, TWideStrings List);
```

Description

Call GetFieldNames to retrieve a list of fields in a table.

TableName names the table whose field names you want added to the list.

List is a TStrings descendant that receives the field names. Any existing strings are deleted from the list before GetFieldNames adds the names of all the fields in TableName.

For example, the following line fills a list box with the names of all fields in the Employee table:

```
SQLConnection1.GetFieldNames('Employee', ListBox1.Items);
SQLConnection1->GetFieldNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 1523)

GetPackageNames (see page 1522)

GetTableNames (see page 1525)

GetIndexNames (see page 1521)

GetProcedureParams (see page 1524)

SetSchemaInfo (see page 1450)

24.1.9.4.16 GetIndexNames Method

This is the overview for the GetIndexNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetIndexNames (string, string, TStrings) (see page 1521)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (string, TStrings) (see page 1521)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (WideString, TWideStrings) (see page 1521)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (WideString, WideString, TWideStrings) (see page 1521)	Populates a string list with the names of indexes on a table.

24.1.9.4.16.1 TSQLConnection.GetIndexNames Method (WideString, TWideStrings)

Populates a string list with the names of indexes on a table.

Pascal

```
procedure GetIndexNames(const TableName: string; List: TStrings); overload; deprecated;
procedure GetIndexNames(const TableName: string; const SchemaName: string; List: TStrings);
overload; deprecated;
procedure GetIndexNames(const TableName: WideString; List: TWideStrings); overload;
procedure GetIndexNames(const TableName: WideString; const SchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetIndexNames(const UnicodeString TableName, TStrings List);
__fastcall GetIndexNames(const UnicodeString TableName, const UnicodeString SchemaName,
TStrings List);
__fastcall GetIndexNames(const BSTR TableName, TWideStrings List);
__fastcall GetIndexNames(const BSTR TableName, const BSTR SchemaName, TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of the indexes defined for a table.

List is a TStrings descendant that receives the index names. Any existing strings are deleted from the list before GetIndexNames adds the names of all the indexes on TableName.

TableName names the table whose indexes you want added to the list.

For example, the following line fills a list box with the names of all indexes on the Employee table:

```
SQLConnection1.GetIndexNames('Employee', ListBox1.Items);
SQLConnection1->GetIndexNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 1523)

GetTableNames (see page 1525)

GetPackageNames (see page 1522)

GetFieldNames (see page 1519)

GetProcedureParams (see page 1524)

SetSchemaInfo (see page 1450)

Obtaining Information About Indexes

24.1.9.4.17 TSQLConnection.GetLoginUsername Method

Returns the username that is logged in on the current connection.

Pascal

```
function GetLoginUsername: WideString;
```

C++

```
__fastcall BSTR GetLoginUsername();
```

24.1.9.4.18 GetPackageNames Method

This is the overview for the GetPackageNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetPackageNames (TStrings) (see page 1522)	Populates a string list with the names of all packages defined on the server.
	TSQLConnection.GetPackageNames (TWideStrings) (see page 1522)	Populates a string list with the names of all packages defined on the server.

24.1.9.4.18.1 TSQLConnection.GetPackageNames Method (TStrings)

Populates a string list with the names of all packages defined on the server.

Pascal

```
procedure GetPackageNames(List: TStrings); overload; deprecated;
procedure GetPackageNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetPackageNames(TStrings List);
__fastcall GetPackageNames(TWideStrings List);
```

Description

Call GetPackageNames to retrieve a list of packages defined on the associated database. This method is only relevant to Oracle servers.

List is a TStrings descendant that receives the package names. Any existing strings are deleted from the list before GetPackageNames adds the names of all packages that the database defines.

The following example fills a list box with the names of all packages defined by the database:

```
SQLConnection1.GetPackageNames(ListBox1.Items);
SQLConnection1->GetPackageNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 1521)

GetProcedureNames (see page 1523)

GetTableNames (see page 1525)

GetFieldNames (see page 1519)

GetProcedureParams (see page 1524)

SetSchemaInfo (see page 1450)

24.1.9.4.19 GetProcedureNames Method

This is the overview for the GetProcedureNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureNames (string, string, TStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (string, TStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.

	TSQLConnection.GetProcedureNames (TStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (TWideStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (Widestring, TWideStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (WideString, WideString, TWideStrings) (see page 1523)	Populates a string list with the names of stored procedures in the database.

24.1.9.4.19.1 TSQLConnection.GetProcedureNames Method (TStrings)

Populates a string list with the names of stored procedures in the database.

Pascal

```
procedure GetProcedureNames(List: TStrings); overload; deprecated;
procedure GetProcedureNames(const PackageName: string; List: TStrings); overload; deprecated;
procedure GetProcedureNames(const PackageName: string; const SchemaName: string; List:
TStrings); overload; deprecated;
procedure GetProcedureNames(List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: Widestring; List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: WideString; const SchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetProcedureNames(TStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, TStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, const UnicodeString SchemaName,
TStrings List);
__fastcall GetProcedureNames(TWideStrings List);
__fastcall GetProcedureNames(const BSTR PackageName, TWideStrings List);
__fastcall GetProcedureNames(const BSTR PackageName, const BSTR SchemaName, TWideStrings List);
```

Description

Call GetProcedureNames to retrieve a list of stored procedures in the associated database.

List is a TStrings descendant that receives the stored procedure names. Any existing strings are deleted from the list before GetProcedureNames adds the names of all stored procedures in the database.

PackageName is the name of an Oracle package that contains the stored procedures. This syntax should only be used with Oracle servers when you only want the stored procedures from a specific package.

The following example fills a list box with the names of all stored procedures in the database:

```
SQLConnection1.GetProcedureNames(ListBox1.Items);
SQLConnection1->GetProcedureNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 1521)

GetTableNames (see page 1525)

GetPackageNames (see page 1522)

GetFieldNames (see page 1519)

GetProcedureParams (see page 1524)

SetSchemaInfo (see page 1450)

24.1.9.4.20 GetProcedureParams Method

This is the overview for the GetProcedureParams method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureParams (WideString, TList) (see page 1524)	Populates a list with parameter descriptors for the parameters in a stored procedure.
	TSQLConnection.GetProcedureParams (WideString, WideString, TList) (see page 1524)	Populates a list with parameter descriptors for the parameters in a stored procedure.
	TSQLConnection.GetProcedureParams (Widestring, Widestring, Widestring, TList) (see page 1524)	Populates a list with parameter descriptors for the parameters in a stored procedure.

24.1.9.4.20.1 TSQLConnection.GetProcedureParams Method (WideString, TList)

Populates a list with parameter descriptors for the parameters in a stored procedure.

Pascal

```
procedure GetProcedureParams(ProcedureName: WideString; List: TList); overload;
procedure GetProcedureParams(ProcedureName: WideString; PackageName: WideString; List: TList);
overload;
procedure GetProcedureParams(ProcedureName: Widestring; PackageName: Widestring; SchemaName:
Widestring; List: TList); overload;
```

C++

```
__fastcall GetProcedureParams(BSTR ProcedureName, TList List);
__fastcall GetProcedureParams(BSTR ProcedureName, BSTR PackageName, TList List);
__fastcall GetProcedureParams(BSTR ProcedureName, BSTR PackageName, BSTR SchemaName, TList
List);
```

Description

Call GetProcedureParams to retrieve information about the parameters of a specified stored procedure.

ProcedureName is the name of the stored procedure for whose parameters you want descriptions.

PackageName is the name of the Oracle package to which the stored procedure belongs. This parameter should only be used if TSQLConnection (see page 1510) connects to an Oracle server.

List is a TList object that receives the parameter descriptions. It should be an empty list on entry. For each parameter, GetProcedureParams adds a reference to an SPParamDesc (see page 1625) class (Delphi) or structure (C++). Each SPParamDesc (see page 1625) entity describes a parameter of a specified stored procedure, including its name, index, parameter type, field type, and so on. You are responsible for freeing the individual SPPParamDesc entities that GetProcedureParams dynamically allocates. (You can use FreeProcParams (see page 1620) to free all the entries and the list object as well).

Note: You can convert the list of SPParamDesc (see page 1625) entities to the more familiar TParams object by calling the global LoadParamListItems (see page 1622) procedure.

See Also

GetIndexNames (see page 1521)

GetTableNames (see page 1525)

GetPackageNames (see page 1522)

GetFieldNames (see page 1519)

GetProcedureNames (see page 1523)

SetSchemaInfo (see page 1450)

LoadParamListItems

FreeProcParams

24.1.9.4.21 GetSchemaNames Method

This is the overview for the GetSchemaNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetSchemaNames (TStrings) (see page 1525)	Returns the schema names for all objects in the given data source.
	TSQLConnection.GetSchemaNames (TWideStrings) (see page 1525)	Returns the schema names for all objects in the given data source.

24.1.9.4.21.1 TSQLConnection.GetSchemaNames Method (TStrings)

Returns the schema names for all objects in the given data source.

Pascal

```
procedure GetSchemaNames(List: TStrings); overload;
procedure GetSchemaNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetSchemaNames(TStrings List);
__fastcall GetSchemaNames(TWideStrings List);
```

24.1.9.4.22 GetTableNames Method

This is the overview for the GetTableNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetTableNames (TStrings, Boolean) (see page 1525)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TStrings, WideString, Boolean) (see page 1525)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TWideStrings, Boolean) (see page 1525)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TWideStrings, WideString, Boolean) (see page 1525)	Populates a string list with the names of tables in the database.

24.1.9.4.22.1 TSQLConnection.GetTableNames Method (TStrings, Boolean)

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False); overload;
procedure GetTableNames(List: TStrings; SchemaName: WideString; SystemTables: Boolean =
False); overload;
procedure GetTableNames(List: TWideStrings; SystemTables: Boolean = False); overload;
procedure GetTableNames(List: TWideStrings; SchemaName: WideString; SystemTables: Boolean =
False); overload;
```

C++

```
__fastcall GetTableNames(TStrings List, Boolean SystemTables = False);
__fastcall GetTableNames(TStrings List, BSTR SchemaName, Boolean SystemTables = False);
__fastcall GetTableNames(TWideStrings List, Boolean SystemTables = False);
__fastcall GetTableNames(TWideStrings List, BSTR SchemaName, Boolean SystemTables = False);
```

Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is a TStrings descendant that receives the table names. Any existing strings are deleted from the list before GetTableNames adds the names of all tables in the database.

SystemTables specifies whether the list of table names should include only the database's system tables. If SystemTables is true, only the system tables are added to List. If SystemTables is false, the list is filled with any tables that match the criteria specified by the TableScope (see page 1548) property.

For example, the following line fills a list box with the names of all data tables in the database:

```
SQLConnection1.GetTableNames(ListBox1.Items, False);
SQLConnection1->GetTableNames(ListBox1->Items, false);
```

See Also

GetIndexNames (see page 1521)

GetProcedureParams (see page 1524)

GetFieldNames (see page 1519)

GetProcedureNames (see page 1523)

GetPackageNames (see page 1522)

SetSchemaInfo (see page 1450)

TableScope (see page 1548)

24.1.9.4.23 TSQLConnection.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected

Loaded

24.1.9.4.24 TSQLConnection.LoadParamsFromIniFile Method

Sets the DriverName (see page 1536) and Params (see page 1546) properties to the values associated with the ConnectionName (see page 1534) property.

Pascal

```
procedure LoadParamsFromIniFile(FFileName: WideString = '');
```

C++

```
__fastcall LoadParamsFromIniFile(BSTR FFileName = '');
```

Description

Call LoadParamsFromIniFile to ensure that the DriverName (see page 1536) and Params (see page 1546) properties accurately reflect the connection configuration for the ConnectionName (see page 1534) property that is stored in the file specified by the AFileName parameter. If the AFileName parameter is omitted, LoadParamsFromIniFile uses the dbxconnections.ini file. Otherwise, itis a fully-qualified path name to a file that has the same format as the dbxconnections.ini file. Typically, this file is created by making a copy of the dbxconnections.ini file under a different name.

In most cases, it is unnecessary to load connection configurations at runtime because when you set ConnectionName (see page 1534) at design time, the DriverName (see page 1536) and Params (see page 1546) properties are automatically set as well. However, LoadParamsFromIniFile lets you use a customized set of connection configurations at runtime that differ from those stored in the dbxconnections.ini file, or to temporarily override the DriverName (see page 1536) or connection parameter values at design time and then reestablish the defaults at runtime.

Note: Do not set the LoadParamsOnConnect (see page 1539) property to true if your application calls LoadParamsFromIniFile and specifies an alternate connections file. When LoadParamsOnConnect (see page 1539) is true, it loads the connection configuration from the dbxconnections.ini

file immediately before opening a connection, overriding any values loaded by this method.

See Also

ConnectionName (see page 1534)

LoadParamsOnConnect (see page 1539)

DriverName (see page 1536)

Params (see page 1546)

Setting Up TSQLConnection

Configuring TSQL Connection

24.1.9.4.25 TSQLConnection.LoadSQLDll Method

Pascal

```
procedure LoadSQLDll;
```

C++

```
__fastcall LoadSQLDll();
```

Description

This is LoadSQLDll, a member of class TSQLConnection.

24.1.9.4.26 TSQLConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

24.1.9.4.27 OpenSchema Method

This is the overview for the OpenSchema method overload.

Overload List

	Name	Description
	TSQLConnection.OpenSchema (TSchemaType, Widestring, TWideStrings) (see page 1529)	This is OpenSchema, a member of class TSQLConnection.
	TSQLConnection.OpenSchema (TSchemaType, WideString, WideString, TWideStrings) (see page 1528)	This is OpenSchema, a member of class TSQLConnection.

24.1.9.4.27.1 TSQLConnection.OpenSchema Method (TSchemaType, WideString, WideString, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: WideString; SSchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, BSTR sInfo, BSTR SSchemaName, TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

24.1.9.4.27.2 TSQLConnection.OpenSchema Method (TSchemaType, Widestring, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: Widestring; List: TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, BSTR sInfo, TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

24.1.9.4.28 TSQLConnection.Rollback Method

Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.

Pascal

```
procedure Rollback(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Rollback(TTransactionDesc TransDesc);
```

Description

Call Rollback to cancel any changes made during the transaction described by TransDesc and to end it.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 1530) method when starting the transaction.

If the specified transaction is the only current transaction, Rollback sets the InTransaction (see page 1537) property to false after canceling the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 1537)

StartTransaction (see page 1530)

Commit (see page 1516)

TransactionsSupported (see page 1549)

TransactionLevel (see page 1467)

24.1.9.4.29 TSQLConnection.SetTraceCallbackEvent Method

Pascal

```
procedure SetTraceCallbackEvent(Event: TSQLCallbackEvent; IClientInfo: Integer);
```

C++

```
__fastcall SetTraceCallbackEvent(TSQLCallbackEvent Event, int IClientInfo);
```

Description

This is SetTraceCallbackEvent, a member of class TSQLConnection.

24.1.9.4.30 TSQLConnection.SQLError Method

Pascal

```
procedure SQLError(OpStatus: SQLResult; eType: TSQLExceptionType; const Command: TISQLCommand
= nil);
```

C++

```
__fastcall SQLError(SQLResult OpStatus, TSQLExceptionType eType, const TISQLCommand Command =
nil);
```

Description

This is SQLError, a member of class TSQLConnection.

24.1.9.4.31 TSQLConnection.StartTransaction Method

Initiates a new transaction in the associated database.

Pascal

```
procedure StartTransaction(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall StartTransaction(TTransactionDesc TransDesc);
```

Description

Call StartTransaction to start a new transaction against a database server that supports transactions. (MySQL servers do not support transactions.) You can check the TransactionsSupported (see page 1549) property to determine whether the server supports transactions. If the server does not support transactions, StartTransaction does nothing.

TransDesc describes the transaction you are starting. It lets you specify

A user-defined transaction identifier. If the server supports nested or overlapped transactions, you can initiate multiple transactions by calling StartTransaction and providing each transaction with its own unique identifier.

The transaction isolation level. This indicates the degree to which this transaction can "see" changes made by other transactions.

If the transaction is successfully started, StartTransaction sets the InTransaction (see page 1537) property to true.

Updates, insertions, and deletions that take place after a call to StartTransaction are enlisted in that transaction until another overlapping transaction is started or until the application calls Commit (see page 1516) to save the changes or Rollback (see page 1529) to cancel them.

Tip: If the server does not support multiple transactions, check the status of the InTransaction (see page 1537) property to be sure that another transaction is not already in progress.

Tip: When working with overlapped transactions, you can indicate the transaction to which a dataset belongs by setting its TransactionLevel property to the transaction identifier you assign in this method.

Note: This method has been deprecated.

See Also

InTransaction (see page 1537)

Commit (see page 1516)

Rollback (see page 1529)

Params (see page 1546)

TransactionsSupported (see page 1549)

TransactionLevel (see page 1467)

24.1.9.5 TSQLConnection Properties

The properties of the TSQLConnection class are listed here.

Protected Properties

	Name	Description
	Connection (see page 1534)	This is Connection, a member of class TSQLConnection.
	ConnectionRegistryFile (see page 1535)	This is ConnectionRegistryFile, a member of class TSQLConnection.
	Driver (see page 1536)	This is Driver, a member of class TSQLConnection.
	DriverRegistryFile (see page 1537)	This is DriverRegistryFile, a member of class TSQLConnection.
	LastError (see page 1538)	This is LastError, a member of class TSQLConnection.
	QuoteChar (see page 1547)	This is QuoteChar, a member of class TSQLConnection.
	SQLDllHandle (see page 1547)	This is SQLDllHandle, a member of class TSQLConnection.

Public Properties

	Name	Description
	ActiveStatements (see page 1532)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 1532)	Specifies whether the SQL connection automatically clones database connections when they are needed.
	ConnectionState (see page 1535)	Indicates the current state of the TSQLConnection (see page 1510) object.
	DataSets (see page 1535)	Lists all active datasets for the connection component.
	InTransaction (see page 1537)	Indicates whether a transaction is in progress.
	LocaleCode (see page 1540)	Identifies the locale that is used for sorting items from datasets that use this connection.
	MaxStmtsPerConn (see page 1544)	Indicates the server limit on the number of statements supported by a single database connection.
	MetaData (see page 1545)	Provides an interface to the dbExpress object that supplies server metadata.
	MultipleTransactionsSupported (see page 1545)	Indicates whether the database server supports multiple transactions.
	ParamsLoaded (see page 1547)	Indicates when parameter values have been loaded from dbxconnections.ini.
	SQLConnection (see page 1547)	This is SQLConnection, a member of class TSQLConnection.
	SQLHourGlass (see page 1548)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
	TraceCallbackEvent (see page 1549)	Provides access to the callback function that executes for every SQL command passed to or from the server.
	TransactionsSupported (see page 1549)	Indicates whether the database server supports transactions.

Published Properties

	Name	Description
	Connected (see page 1533)	Specifies whether the connection is active.
	ConnectionName (see page 1534)	Names the connection configuration.
	DriverName (see page 1536)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 1537)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 1538)	Specifies whether the connection stays active when no datasets are open.

	LibraryName (see page 1538)	Specifies the dbExpress driver that TSQLConnection (see page 1510) uses to establish a connection.
	LoadParamsOnConnect (see page 1539)	Specifies whether the TSQLConnection (see page 1510) component loads the configuration named by ConnectionName (see page 1534) immediately before connecting to the server.
	LoginPrompt (see page 1540)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 1546)	Lists the connection parameters.
	TableScope (see page 1548)	Indicates what types of tables are returned when fetching schema information about tables.
	VendorLib (see page 1550)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

24.1.9.5.1 TSQLConnection.ActiveStatements Property

Indicates the number of statements currently active with the database server.

Pascal

```
property ActiveStatements: LongWord;
```

C++

```
__property LongWord ActiveStatements;
```

Description

Read (see page 3727) ActiveStatements to determine how many statements are currently executing using the database connection. While a dataset is prepared, it creates an active statement that lasts until the dataset is unprepared. When you set a dataset's Prepared property to true, the statement remains active until you change the Prepared property to false. When the Prepared property is false, the dataset implicitly prepares its statement when you open the dataset, and the statement remains active until you close the dataset. In addition, executing an SQL statement (for example, using the Execute (see page 1517) method) creates an active statement temporarily, until the statement finishes executing.

Use ActiveStatements to check whether an application has reached the server-imposed limit on the number of active statements allowed to a connection (the MaxStmtsPerConn (see page 1544) property). This is especially important if AutoClone (see page 1532) is false, because if you exceed MaxStmtsPerConn (see page 1544) when AutoClone (see page 1532) is false, the SQL connection raises an exception. To avoid exceeding the server-imposed limit, do not open any datasets or execute SQL statements when ActiveStatements is the same as MaxStmtsPerConn (see page 1544) (unless MaxStmtsPerConn (see page 1544) is 0).

See Also

DataSets (see page 1535)

Execute (see page 1517)

MaxStmtsPerConn (see page 1544)

Prepared (see page 1464)

AutoClone (see page 1532)

24.1.9.5.2 TSQLConnection.AutoClone Property

Specifies whether the SQL connection automatically clones database connections when they are needed.

Pascal

```
property AutoClone: Boolean;
```

C++

```
__property Boolean AutoClone;
```

Description

AutoClone controls whether the TSQLConnection (see page 1510) component automatically creates duplicate database connections when the number of statements allowed by the database server is exceeded. By default, AutoClone is true, meaning that when the ActiveStatements (see page 1532) property is the same as MaxStmtsPerConn (see page 1544), trying to execute another statement (open another dataset or call the Execute (see page 1517) method) causes TSQLConnection (see page 1510) to open a secondary (cloned) connection. The result behaves just like a connection to a server that does not limit the number of statements, except that it consumes more resources (for the additional, cloned connections).

Change AutoClone to false to prevent TSQLConnection (see page 1510) from cloning the database connection when the application attempts to exceed the maximum number of statements. When AutoClone is false, attempts to exceed MaxStmtsPerConn (see page 1544) result in an exception.

Tip: When AutoClone is false, you can check the ActiveStatements (see page 1532) property to ensure that you do not exceed the server-imposed limit on the number of statements. If you are about to exceed that limit, you can explicitly call CloneConnection (see page 1515) to clone the connection yourself.

See Also

MaxStmtsPerConn (see page 1544)

ActiveStatements (see page 1532)

CloneConnection (see page 1515)

Connecting to the Database Server

24.1.9.5.3 TSQLConnection.Connected Property

Specifies whether the connection is active.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to trueto establish a connection to the database server. When a dataset that uses the TSQLConnection (see page 1510) object is activated, it automatically sets Connected to true. Set Connected to false to close a connection.

Setting Connected to true generates a BeforeConnect (see page 1551) event, establishes the connection, and generates an AfterConnect (see page 1550) event. In addition, when setting Connected to true, TSQLConnection (see page 1510) may display a login dialog, depending on the value of LoginPrompt (see page 1540).

Setting Connected to false generates a BeforeDisconnect (see page 1552) event, drops the connection, and generates an AfterDisconnect (see page 1551) event.

See Also

Open

Close

ConnectionState (see page 1535)

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

24.1.9.5.4 TSQLConnection.Connection Property

Pascal

```
property Connection: TISQLConnection;
```

C++

```
__property TISQLConnection Connection;
```

Description

This is Connection, a member of class TSQLConnection.

24.1.9.5.5 TSQLConnection.ConnectionName Property

Names the connection configuration.

Pascal

```
property ConnectionName: string;
```

C++

```
__property UnicodeString ConnectionName;
```

Description

Set ConnectionName to use a named connection configuration. Setting ConnectionName automatically sets the DriverName (see page 1536) and Params (see page 1546) properties to reflect the driver and connection parameters stored under that name in the dbxconnections.ini file.

Tip: You can define new connection configurations using the Connection (see page 1534) Editor. To launch the Connection (see page 1534) Editor, double-click on the TSQLConnection (see page 1510) component.

If you are using a different dbxconnections.ini file at design time than the one used at runtime, you can force the configuration to reload at runtime by setting the LoadParamsOnConnect (see page 1539) property or calling LoadParamsFromIniFile (see page 1527). In most cases, however, this is unnecessary, because the DriverName (see page 1536) and Params (see page 1546) property settings that are loaded when you set ConnectionName are saved in the form file. In fact, unless you set LoadParamsOnConnect (see page 1539) or call LoadParamsFromIniFile (see page 1527), you do not need to deploy the dbxconnections.ini file with your application even if you are using named connections.

See Also

LoadParamsOnConnect (see page 1539)

LoadParamsFromIniFile (see page 1527)

DriverName (see page 1536)

Params (see page 1546)

Setting Up a Simple Dataset

Setting Up TSQLConnection

Configuring TSQL Connection

24.1.9.5.6 TSQLConnection.ConnectionRegistryFile Property

Pascal

```
property ConnectionRegistryFile: string;
```

C++

```
__property UnicodeString ConnectionRegistryFile;
```

Description

This is ConnectionRegistryFile, a member of class TSQLConnection.

24.1.9.5.7 TSQLConnection.ConnectionState Property

Indicates the current state of the TSQLConnection (see page 1510) object.

Pascal

```
property ConnectionState: TConnectionState;
```

C++

```
__property TConnectionState ConnectionState;
```

Description

Read (see page 3727) ConnectionState at runtime to determine the activity being performed, if any, by the connection at a specific point in time. Use ConnectionState to coordinate the activities of the connection component, so that you do not start one activity while it is currently engaged in another.

See Also

Open

Connected (see page 1533)

Close

Execute (see page 1517)

24.1.9.5.8 TSQLConnection.DataSets Property

Lists all active datasets for the connection component.

Pascal

```
property DataSets [Index: Integer]: TCustomSQLDataSet;
```

C++

```
__property TCustomSQLDataSet DataSets[int Index];
```

Description

Use DataSets to access active datasets associated with a connection component. An active dataset is one that is currently open.

The Index parameter ranges between 0 and DataSetCount.

The value of DataSets changes as the datasets that use this TSQLConnection (see page 1510) component open and close.

Note: If a TSQLClientDataSet uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

See Also

DataSetCount

Active

MaxStmtsPerConn (see page 1544)

TSQLClientDataSet

24.1.9.5.9 TSQLConnection.Driver Property

Pascal

```
property Driver: ISQLDriver;
```

C++

```
__property ISQLDriver Driver;
```

Description

This is Driver, a member of class TSQLConnection.

24.1.9.5.10 TSQLConnection.DriverName Property

Specifies the database driver associated with the SQL connection.

Pascal

```
property DriverName: string;
```

C++

```
__property UnicodeString DriverName;
```

Description

If you are not using named connections, set DriverName to the name of an installed dbExpress driver such as INTERBASE, MYSQL, INFORMIX, ORACLE, or DB2. Installed drivers are listed in the dbxdrivers.ini file, and their names can be retrieved using the GetDriverNames (see page 1621) function.

If you are using named connections, the DriverName property is automatically set when you set the ConnectionName (see page 1534) property.

When you set DriverName, TSQLConnection (see page 1510) automatically sets the LibraryName (see page 1538) and VendorLib (see page 1550) properties to the names of the libraries (DLLs or shared objects) that implement the specified driver.

See Also

LibraryName (see page 1538)

VendorLib (see page 1550)

ConnectionName (see page 1534)

GetDriverNames (see page 1621)

Setting Up TSQLConnection

Configuring TSQL Connection

Connecting to Databases with TDatabase

24.1.9.5.11 TSQLConnection.DriverRegistryFile Property

Pascal

```
property DriverRegistryFile: string;
```

C++

```
__property UnicodeString DriverRegistryFile;
```

Description

This is DriverRegistryFile, a member of class TSQLConnection.

24.1.9.5.12 TSQLConnection.GetDriverFunc Property

Specifies the function that returns the top-level dbExpress driver.

Pascal

```
property GetDriverFunc: string;
```

C++

```
__property UnicodeString GetDriverFunc;
```

Description

GetDriverFunc is the name of a function in the dbExpress driver specified by the LibraryName (see page 1538) property. The specified function returns the top-level dbExpress interface. Typically, this property is set to the function name that is stored in the dbxdrivers.ini file.

Note: The GetDriverFunc, LibraryName (see page 1538), and VenderLib properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

LibraryName (see page 1538)

VendorLib (see page 1550)

24.1.9.5.13 TSQLConnection.InTransaction Property

Indicates whether a transaction is in progress.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Read (see page 3727) InTransaction at runtime to ascertain whether a transaction is currently active for the SQL connection component. If the connection component is currently in a transaction, all communication with the server that passes through the connection is enlisted in that transaction.

InTransaction is true if there is an active transaction. If no transaction is active, InTransaction is false.

A transaction begins when the application calls BeginTransaction and lasts until the application calls CommitFreeAndNil, RollbackFreeAndNil, or RollbackIncompleteFreeAndNi. If there are multiple transactions, InTransaction remains true until all

transactions are either committed or rolled back.

See Also

BeginTransaction

CommitFreeAndNil

RollbackFreeAndNil

RollbackIncompleteFreeAndNi

TransactionsSupported (see page 1549)

TransactionLevel (see page 1467)

MultipleTransactionsSupported (see page 1545)

24.1.9.5.14 TSQLConnection.KeepConnection Property

Specifies whether the connection stays active when no datasets are open.

Pascal

```
property KeepConnection: Boolean;
```

C++

```
__property Boolean KeepConnection;
```

Description

Use KeepConnection to specify whether TSQLConnection (see page 1510) maintains an open database connection even if no associated dataset components are currently active. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or in applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be reestablished and initialized.

See Also

DataSets (see page 1535)

Connected (see page 1533)

24.1.9.5.15 TSQLConnection.LastError Property

Pascal

```
property LastError: string;
```

C++

```
__property UnicodeString LastError;
```

Description

This is LastError, a member of class TSQLConnection.

24.1.9.5.16 TSQLConnection.LibraryName Property

Specifies the dbExpress driver that TSQLConnection (see page 1510) uses to establish a connection.

Pascal

```
property LibraryName: string;
```

C++

```
__property UnicodeString LibraryName;
```

Description

LibraryName is the dbExpress library associated with the driver specified by DriverName (see page 1536). Most applications do not need to use LibraryName directly, because this property is set automatically when the DriverName (see page 1536) property is set.

Note: The association between a dbExpress library and a driver name is stored in the dbxdrivers.ini

file when you install the dbExpress driver. The file named by this property is the only dbExpress file that must be deployed with your application.

Note: The LibraryName, GetDriverFunc (see page 1537), and VenderLib properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 1536)

VendorLib (see page 1550)

GetDriverFunc (see page 1537)

Setting Up TSQLConnection

Configuring TSQL Connection

24.1.9.5.17 TSQLConnection.LoadParamsOnConnect Property

Specifies whether the TSQLConnection (see page 1510) component loads the configuration named by ConnectionName (see page 1534) immediately before connecting to the server.

Pascal

```
property LoadParamsOnConnect: Boolean;
```

C++

```
__property Boolean LoadParamsOnConnect;
```

Description

Set LoadParamsOnConnect to true if you want the TSQLConnection (see page 1510) object to load the connection configuration (DriverName (see page 1536) and Params (see page 1546) properties) associated with ConnectionName (see page 1534) at runtime. Typically, this is unnecessary because when you set ConnectionName (see page 1534) at design time, the DriverName (see page 1536) and Params (see page 1546) properties are automatically set as well. However, if you use a different dbxconnections.ini file at runtime, or if you temporarily override the DriverName (see page 1536) or any connection parameter values at design time, setting LoadParamsOnConnect to true ensures that at runtime the connection is established using the configuration associated with ConnectionName (see page 1534) in the dbxconnections.ini file.

Note: If you set LoadParamsOnConnect to true, you must be sure to deploy the dbxconnections.ini

file with your application. When you rely on the design-time associations, dbxconnections.ini is not needed at runtime.

Warning: If you set LoadParamsOnConnect to true, you must specify a valid connection name as the value for ConnectionName (see page 1534).

See Also

ConnectionName (see page 1534)

LoadParamsFromIniFile (see page 1527)

DriverName (see page 1536)

Params (see page 1546)

Setting Up TSQLConnection

Configuring TSQL Connection

24.1.9.5.18 TSQLConnection.LocaleCode Property

Identifies the locale that is used for sorting items from datasets that use this connection.

Pascal

```
property LocaleCode: TLocaleCode;
```

C++

```
__property TLocaleCode LocaleCode;
```

Description

LocaleCode represents the locale that client datasets use to sort strings when they are provided in fields from a dataset associated with this connection. By default, this value is 0, which signals that strings should be sorted without regard to a locale. Changing LocaleCode can affect performance, because sorting is slower when it must take locale into account. Therefore, it is recommended that you only change LocaleCode if the data is known to contain extended characters that will affect the sort order.

Note: Locale codes are treated as connection parameters: they are stored in the dbxconnections.ini

file along with the other parameters, and appear in the Params (see page 1546) property list. Changes to this property change the entry in the Params (see page 1546) property list.

See Also

DataSets (see page 1535)

SQLConnection (see page 1467)

LoadParamsFromIniFile (see page 1527)

Params (see page 1546)

LoadParamsOnConnect (see page 1539)

TClientDataSet

24.1.9.5.19 TSQLConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt;
```

C++

```
__property LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

C++ Examples:

```
//
//This example demostrates the use of ADO for database conectivity.
//Example assumes that a TDBGrid is placed on the form.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Login details */
    String UserName = "db_user_name";
    String PassWord = "db_pass_word";
    String Server = "my.db.server";

    /* Connection String */
    String ConnString =
        "Provider=SQLOLEDB.1;Persist Security Info=False;";
    ConnString +=
        "User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;";
    ConnString +=
        "Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;";
    ConnString +=
```

```
        "Tag with column collation when possible=False";

    /* SQL Query */
    String SQLStr = "SELECT * FROM customer WHERE customer_id = :AnId;";

    /* All ADO variables */
    TADOConnection* ADOConn;
    TADOQuery* ADOQuery;
    TDataSource* DataSrc;
    TParameter* Param;

    /* Create an ADO connection */
    ADOConn = new TADOConnection(this);

    /* Setup the connection string */
    ADOConn->ConnectionString = Format(ConnString,
        ARRAYOFCONST((UserName, PassWord, Server)));

    /* Disable login prompt */
    ADOConn->LoginPrompt = False;

    try
    {
        ADOConn->Connected = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the query */
    ADOQuery = new TADOQuery(this);
    ADOQuery->Connection = ADOConn;
    ADOQuery->SQL->Add(SQLStr);

    /* Update the parameter that was parsed from the SQL query: AnId */
    Param = ADOQuery->Parameters->ParamByName("AnId");
    Param->DataType = ftInteger;
    Param->Value = 1;

    /* Set the query to Prepared - will improve performance */
    ADOQuery->Prepared = true;

    try
    {
        ADOQuery->Active = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the data source */
    DataSrc = new TDataSource(this);
    DataSrc->DataSet = ADOQuery;
    DataSrc->Enabled = true;

    /* Finally initilalize the grid */
    DBGrid1->DataSource = DataSrc;
}
```

Delphi Examples:

```
{
This example demostrates the use of ADO for database conectivity.
Example assumes that a TDBGrid is placed on the form.
}
procedure TForm2.FormCreate(Sender: TObject);
const
  { Connection string }
  ConnString =
  'Provider=SQLOLEDB.1;Persist Security Info=False;' +
  'User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;' +
  'Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;'+
  'Tag with column collation when possible=False';

  { SQL Query }
  SQLStr = 'SELECT * FROM customer WHERE customer_id = :AnId;';

  { User access }
  UserName = 'db_user_name';
  PassWord = 'db_pass_word';
  Server = 'my.db.server';

var
  ADOConn  : TADOConnection;
  ADOQuery : TADOQuery;
  DataSrc  : TDataSource;
  Param    : TParameter;

begin
  { Create an ADO connection }
  ADOConn := TADOConnection.Create(Self);
  { Setup the provider engine }

  { Setup the connection string }
  ADOConn.ConnectionString := Format(ConnString,
    [UserName, PassWord, Server]);

  { Disable login prompt }
  ADOConn.LoginPrompt := False;

  try
    ADOConn.Connected := True;
  except
    on e: EADOError do
    begin
      MessageDlg('Error while connecting', mtError,
                 [mbOK], 0);

      Exit;
    end;
  end;

  { Create the query }
  ADOQuery := TADOQuery.Create(Self);
  ADOQuery.Connection := ADOConn;
  ADOQuery.SQL.Add(SQLStr);

  { Update the parameter that was parsed from the SQL query: AnId }
  Param := ADOQuery.Parameters.ParamByName('AnId');
  Param.DataType := ftInteger;
  Param.Value := 1;

  { Set the query to Prepared - will improve performance }
  ADOQuery.Prepared := true;

  try
    ADOQuery.Active := True;
```

```
    except
      on e: EADOError do
      begin
        MessageDlg('Error while doing query', mtError,
                   [mbOK], 0);

        Exit;
      end;
    end;

    { Create the data source }
    DataSrc := TDataSource.Create(Self);
    DataSrc.DataSet := ADOQuery;
    DataSrc.Enabled := true;

    { Finally initilalize the grid }
    DBGrid1.DataSource := DataSrc;
  end;
```

See Also

OnLogin

BeforeConnect

AfterConnect

OnGetUserName

OnLogin (see page 1552)

OnLogin

Controlling Server Login

24.1.9.5.20 TSQLConnection.MaxStmtsPerConn Property

Indicates the server limit on the number of statements supported by a single database connection.

Pascal

```
property MaxStmtsPerConn: LongWord;
```

C++

```
__property LongWord MaxStmtsPerConn;
```

Description

Read (see page 3727) MaxStmtsPerConn when a connection is active to determine whether the database server imposes a limit on the number of statements that can be executed using the current connection. When MaxStmtsPerConn is 0, the server imposes no restrictions: a single database connection can be used to fetch multiple datasets and execute statements while there are active datasets. When MaxStmtsPerConn is greater than 0, the database server limits the number of statements that can execute. The server allows only MaxStmtsPerConn datasets to be opened using a single connection, and when that number of datasets are open, no more statements can be executed (for example, using the Execute (see page 1517) method).

By default, when MaxStmtsPerConn is greater than 0, TSQLConnection (see page 1510) clones the database connection when this server-imposed limit is reached. For example, if MaxStmtsPerConn is 1, then trying to open a second dataset using the same TSQLConnection (see page 1510) component causes TSQLConnection (see page 1510) to establish a second database connection for the second dataset. If AutoClone (see page 1532) is false, however, TSQLConnection (see page 1510) does not clone connections, but rather generates an exception when the application tries to exceed MaxStmtsPerConn.

Note: The value of MaxStmtsPerConn is only valid when the Connected (see page 1533) property is true.

See Also

DataSets (see page 1535)

Execute (see page 1517)

Connected (see page 1533)

AutoClone (see page 1532)

ActiveStatements (see page 1532)

Connecting to the Database Server

24.1.9.5.21 TSQLConnection.MetaData Property

Provides an interface to the dbExpress object that supplies server metadata.

Pascal

```
property MetaData: TISQLMetaData;
```

C++

```
__property TISQLMetaData MetaData;
```

Description

Typically, applications do not use the MetaData property to work directly with the dbExpress MetaData interface. Instead, they use one of methods that assembles schema information into a list (GetFieldNames (see page 1519), GetIndexNames (see page 1521), GetProcedureNames (see page 1523), GetProcedureParams (see page 1524), or GetTableNames (see page 1525)). When more detailed metadata is required, you can also use the SQL dataset's SetSchemaInfo method to populate an SQL dataset with metadata.

See Also

GetFieldNames (see page 1519)

GetIndexNames (see page 1521)

GetProcedureNames (see page 1523)

GetProcedureParams (see page 1524)

GetTableNames (see page 1525)

SetSchemaInfo (see page 1450)

24.1.9.5.22 TSQLConnection.MultipleTransactionsSupported Property

Indicates whether the database server supports multiple transactions.

Pascal

```
property MultipleTransactionsSupported: LongBool;
```

C++

```
__property LongBool MultipleTransactionsSupported;
```

Description

Read (see page 3727) MultipleTransactionsSupported at runtime to ascertain whether the database server supports multiple transactions (either nested or overlapping).

Note: To determine whether the server supports transactions at all, use the TransactionsSupported (see page 1549) property. If a server supports only a single transaction at a time, TransactionsSupported (see page 1549) is true, and MultipleTransactionsSupported is false.

See Also

CommitFreeAndNil

BeginTransaction

RollbackFreeAndNil

InTransaction (see page 1537)

TransactionsSupported (see page 1549)

24.1.9.5.23 TSQLConnection.Params Property

Lists the connection parameters.

Pascal

```
property Params: TWideStrings;
```

C++

```
__property TWideStrings Params;
```

Description

Params is a TStrings object listing the connection parameters. Each entry in the string list has the form

```
Name=Value
```

where Name is the name of the parameter and Value is its value. If you are using named connection configurations (the ConnectionName (see page 1534) property), the values for each parameter are loaded from the dbxconnections.ini file when you set ConnectName.

At design time, you can set parameter values using the string list editor by double clicking on the Params property in the Object Inspector. Any new values you set override the values loaded when you set ConnectionName (see page 1534) unless you load the connection configuration at runtime using LoadParamsFromIniFile (see page 1527) or LoadParamsOnConnect (see page 1539).

The particular parameters that appear depend on the database server. The Params property is initialized to include a string for each parameter when you set the DriverName (see page 1536) property. One particular parameter, Database, is required for all servers. Its value depends on the server you are using. For example, with InterBase, Database is the name of the .gdb file, while with MySQL, it is the database name that was assigned by the CREATE DATABASE command.

Other typical parameters include HostName (the machine name or IP address of where the server is located), and TransIsolation (the degree to which transactions are isolated from changes made by other users of the database).

See Also

ConnectionName (see page 1534)

LoadParamsFromIniFile (see page 1527)

LoadParamsOnConnect (see page 1539)

DriverName (see page 1536)

LocaleCode (see page 1540)

Setting Up TSQLConnection

24.1.9.5.24 TSQLConnection.ParamsLoaded Property

Indicates when parameter values have been loaded from dbxconnections.ini.

Pascal

```
property ParamsLoaded: Boolean;
```

C++

```
__property Boolean ParamsLoaded;
```

Description

Do not use the ParamsLoaded property. It is used internally by the Connections Editor to keep track of when parameter values have been loaded from the dbxconnections.ini file.

See Also

Params (see page 1546)

LoadParamsFromIniFile (see page 1527)

LoadParamsOnConnect (see page 1539)

24.1.9.5.25 TSQLConnection.QuoteChar Property

Pascal

```
property QuoteChar: WideString;
```

C++

```
__property BSTR QuoteChar;
```

Description

This is QuoteChar, a member of class TSQLConnection.

24.1.9.5.26 TSQLConnection.SQLConnection Property

Pascal

```
property SQLConnection: TISQLConnection;
```

C++

```
__property TISQLConnection SQLConnection;
```

Description

This is SQLConnection, a member of class TSQLConnection.

24.1.9.5.27 TSQLConnection.SQLDllHandle Property

Pascal

```
property SQLDllHandle: THandle;
```

C++

```
__property THandle SQLDllHandle;
```

Description

This is SQLDllHandle, a member of class TSQLConnection.

24.1.9.5.28 TSQLConnection.SQLHourGlass Property

Specifies whether the cursor should change to crSQLWait during lengthy operations.

Pascal

```
property SQLHourGlass: Boolean;
```

C++

```
__property Boolean SQLHourGlass;
```

Description

Set SQLHourGlass to true to specify that the cursor should indicate when the connection is performing a potentially lengthy operation. Such operations include

Establishing a connection to the database server.

Fetching metadata from the database server.

Executing a statement using the Execute (see page 1517) method.

When SQLHourGlass is false, the cursor is not changed when the SQL connection performs such operations.

See Also

Execute (see page 1517)

Open

TCursorType

24.1.9.5.29 TSQLConnection.TableScope Property

Indicates what types of tables are returned when fetching schema information about tables.

Pascal

```
property TableScope: TTableScopes;
```

C++

```
__property TTableScopes TableScope;
```

Description

Set TableScope to determine the type of tables listed by the GetTableNames (see page 1525) method or that populate an SQL dataset after calling its SetSchemaInfo method with a parameter of stTables. TableScope lets you request information about a combination of table types. By default, TableScope requests only ordinary tables and views (tsTable and tsView).

Note: TableScope only affects GetTableNames (see page 1525) when the SystemTables parameter is false and only affects SetSchemaInfo when the SchemaType parameter is stTables. If an application calls GetTablesNames with SystemTables set to true, or calls SetSchemaInfo with SchemaType set to stSysTables, system tables, and only system tables are returned, regardless of the value of TableScope.

See Also

GetTableNames (see page 1525)

SetSchemaInfo (see page 1450)

Fetching Metadata into a Unidirectional Dataset

24.1.9.5.30 TSQLConnection.TraceCallbackEvent Property

Provides access to the callback function that executes for every SQL command passed to or from the server.

Pascal

```
property TraceCallbackEvent: TSQLCallbackEvent;
```

C++

```
__property TSQLCallbackEvent TraceCallbackEvent;
```

Description

Read (see page 3727) TraceCallbackEvent to determine whether the SQL connection is monitoring communication between the application and the SQL server to which it connects. For example, if the application uses a TSQLMonitor (see page 1567) component, that component assigns a value to TraceCallbackEvent, which it uses to monitor SQL commands. Changing the value of the callback would then interfere with the operation of the TSQLMonitor (see page 1567) component.

TraceCallbackEvent is a read-only property. To assign a callback for monitoring SQL commands, use the SetTraceEvent method.

See Also

SetTraceEvent

24.1.9.5.31 TSQLConnection.TransactionsSupported Property

Indicates whether the database server supports transactions.

Pascal

```
property TransactionsSupported: LongBool;
```

C++

```
__property LongBool TransactionsSupported;
```

Description

Read (see page 3727) TransactionsSupported at runtime to ascertain whether the database server supports transactions. If TransactionsSupported is false, then the BeginTransaction, CommitFreeAndNil, RollbackFreeAndNil, and RollbackIncompleteFreeAndNil methods do nothing. If TransactionsSupported is true, you can use the SQL connection component to manage transactions.

Note: Some servers support multiple transactions (either nested or overlapping). When TransactionsSupported is true, you can test whether the server supports multiple transactions by reading the MultipleTransactionsSupported (see page 1545) property.

See Also

BeginTransaction

CommitFreeAndNil

RollbackFreeAndNil

RollbackIncompleteFreeAndNil

InTransaction (see page 1537)

MultipleTransactionsSupported (see page 1545)

24.1.9.5.32 TSQLConnection.VendorLib Property

Specifies the client software library (DLL or shared object) supplied by the database vendor.

Pascal

```
property VendorLib: string;
```

C++

```
__property UnicodeString VendorLib;
```

Description

VendorLib is the library supplied by the database vendor to support client-side use of the database. Most applications do not need to use VendorLib directly, because this property is set automatically when the DriverName (see page 1536) property is set.

Note: The association between a vendor library and a driver name is stored in the dbxdrivers.ini

file when you install the associated dbExpress driver. The file named by this property must be running on the same system as your application when it is deployed. Typically, it is installed when the database is installed rather than with your application.

Note: The VendorLib, LibraryName (see page 1538), and GetDriverFunc (see page 1537) properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 1536)

VendorLib

GetDriverFunc (see page 1537)

Setting Up TSQLConnection

Configuring TSQL Connection

24.1.9.6 TSQLConnection Events

The events of the TSQLConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 1550)	Occurs after a connection is established.
	AfterDisconnect (see page 1551)	Occurs after the connection closes.
	BeforeConnect (see page 1551)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1552)	Occurs immediately before the connection closes.
	OnLogin (see page 1552)	Occurs after the BeforeConnect (see page 1551) event and before a connection is established.

24.1.9.6.1 TSQLConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

24.1.9.6.2 TSQLConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

24.1.9.6.3 TSQLConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

24.1.9.6.4 TSQLConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

24.1.9.6.5 TSQLConnection.OnLogin Event

Occurs after the BeforeConnect (see page 1551) event and before a connection is established.

Pascal

```
property OnLogin: TSQLConnectionLoginEvent;
```

C++

```
__property TSQLConnectionLoginEvent OnLogin;
```

Description

Use the OnLogin event to assign values to the User_Name, Password, and Database parameters immediately before TSQLConnection (see page 1510) attempts to connect to the database server. OnLogin only occurs if the LoginPrompt (see page 1540) property is true. If LoginPrompt (see page 1540) is true but there is no OnLogin event handler, a default login dialog appears in which the user can enter a user name and password. The connection fails if correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler.

See Also

Params (see page 1546)

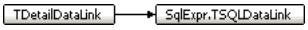
LoginPrompt

BeforeConnect

24.1.10 TSQLDataLink Class

TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.

Class Hierarchy



Pascal

```
TSQLDataLink = class(TDetailDataLink);
```

C++

```
class TSQLDataLink : public TDetailDataLink;
```

File

SqlExpr

Description

SQL datasets use TSQLDataLink to coordinate with another (master) dataset when they represent the detail records for that other dataset. TSQLDataLink passes on data events to the data source that owns it. It is also used to implement the DataSource or MasterSource property of the dataset that owns it.

Do not confuse the dataset that is the value of the TSQLDataLink object's DataSet property with the dataset that uses TSQLDataLink to link to a master dataset. The DataSet property is the master dataset, while the dataset that owns the TSQLDataLink object is the detail set.

See Also

TSQLDataSet (see page 1556)

TSQLQuery (see page 1577)

TSQLTable (see page 1604)

TMasterDataLink

TCustomSQLDataSet (see page 1421)

TSQLTable (see page 1604)

24.1.10.1 TSQLDataLink Members

The following tables list the members exposed by TSQLDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 1554)	Refreshes the parameter values of the detail dataset and re-executes it.
	CheckBrowseMode (see page 1555)	Informs the dataset that owns the TSQLDataLink (see page 1553) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 1555)	Returns the value of the DetailDataSet property.
	RecordChanged (see page 1555)	Refreshes the parameter values of the detail dataset and re-executes it.

Public Methods

	Name	Description
	Create (see page 1554)	Creates an instance of TSQLDataLink (see page 1553).

24.1.10.2 TSQLDataLink.Create Constructor

Creates an instance of TSQLDataLink (see page 1553).

Pascal

```
constructor Create(ADataSet: TCustomSQLDataSet);
```

C++

```
__fastcall TSQLDataLink(TCustomSQLDataSet ADataSet);
```

Description

Do not create instances of TSQLDataLink (see page 1553). It is instantiated by the SQL dataset that uses it to link to a master dataset. The ADataSet parameter is the dataset that creates and owns this SQL datalink. It becomes the value of the DetailDataSet property.

See Also

DetailDataSet

24.1.10.3 TSQLDataLink Methods

The methods of the TSQLDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 1554)	Refreshes the parameter values of the detail dataset and re-executes it.
	CheckBrowseMode (see page 1555)	Informs the dataset that owns the TSQLDataLink (see page 1553) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 1555)	Returns the value of the DetailDataSet property.
	RecordChanged (see page 1555)	Refreshes the parameter values of the detail dataset and re-executes it.

24.1.10.3.1 TSQLDataLink.ActiveChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. ActiveChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Active

Params (see page 1463)

DetailDataSet

24.1.10.3.2 TSQLDataLink.CheckBrowseMode Method

Informs the dataset that owns the TSQLDataLink (see page 1553) object when the current record in the source dataset is about to change.

Pascal

```
procedure CheckBrowseMode; override;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

Applications cannot call the protected CheckBrowseMode procedure. CheckBrowseMode is called immediately before the current record in the source dataset changes. It calls the CheckBrowseMode method of the dataset that owns the TSQLDataLink (see page 1553) object.

See Also

CheckBrowseMode

24.1.10.3.3 TSQLDataLink.GetDetailDataSet Method

Returns the value of the DetailDataSet property.

Pascal

```
function GetDetailDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

GetDetailDataSet is the protected read implementation of the DetailDataSet property.

See Also

DetailDataSet

24.1.10.3.4 TSQLDataLink.RecordChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications cannot call the protected RecordChanged method. It is called automatically when the contents of the current record change. The Field parameter indicates which field in the current record has changed. If Field is nil (Delphi) or NULL (C++), the entire record has changed.

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. RecordChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Params (see page 1463)

DetailDataSet

24.1.11 TSQLDataSet Class

TSQLDataSet represents data retrieved using dbExpress.

Class Hierarchy

Pascal

```
TSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TSQLDataSet : public TCustomSQLDataSet;
```

File

SqlExpr

Description

TSQLDataSet is a general-purpose unidirectional dataset for accessing database information using dbExpress. You can add a TSQLDataSet component to a form or data module at design time, or create one dynamically at runtime.

Use TSQLDataSet to

Represent the records in a database table, the result set of a SELECT query, or the result set returned by a stored procedure.

Execute a query or stored procedure that does not return a result set.

Represent metadata that describes what is available on the database server (tables, stored procedures, fields in a table, and so on).

TSQLDataSet is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL dataset by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLDataSet component, it must be connected to the database server. Therefore, the first step to take when working with TSQLDataSet is to set the SQLConnection (see page 1566) property.

Once the dataset is connected to a server, you can use the Command Text editor to set the CommandText (see page 1561) property if you are using the dataset to execute a query. To execute a stored procedure or represent a single database table, set the CommandType (see page 1562) property and then choose the table or stored procedure name from a drop-down list on the CommandText (see page 1561) property. If you want to access metadata with the SQL dataset, you must specify the metadata you want at runtime by calling the SetSchemaInfo (see page 1450) method.

DefaultRowsetSize (see page 1646) is 20 (DefaultRowsetSize (see page 1646) = 20). To use a different RowSetSize for existing Oracle database connections, manually add the RowSetSize property (for example, RowsetSize = 200) to dbxconnections.ini (Windows) or dbxconnections.conf (Linux). To have the RowSetSize property included in new connections, manually add the RowSetSize property to dbxdrivers.ini (Windows) or dbxdrivers.conf (Linux). RowsetSize can also be modified in code (for example, SQLConnection1.Params.Values['RowsetSize'] := '200'). Currently, only Oracle databases support this feature.

See Also

TSQLQuery (see page 1577)

TSQLStoredProc (see page 1594)

TSQLTable (see page 1604)

TSQLConnection (see page 1510)

TClientDataSet

TDataSetProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Types of Unidirectional Datasets

24.1.11.1 TSQLDataSet Members

The following tables list the members exposed by TSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.

	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
V	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.

Public Methods

	Name	Description
	Create (see page 1560)	Creates an instance of a TSQLDataSet (see page 1556) component.
	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1426)	Destroys the TCustomSQLDataSet (see page 1421) instance.
	ExecSQL (see page 1560)	Executes a query or stored procedure that does not return a set of records.
	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
	PrepareStatement (see page 1440)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	CommandText (see page 1561)	Specifies the command the dataset executes.

	CommandType (see page 1562)	Command type of CommandText (see page 1561) property.
	DataSource (see page 1563)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1564)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 1564)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1565)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1562) is ctTable.
	SQLConnection (see page 1566)	Specifies the SQL connection component that connects the dataset to a database server.

24.1.11.2 TSQLDataSet.Create Constructor

Creates an instance of a TSQLDataSet (see page 1556) component.

Pascal

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

C++

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

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

AOwner is the component, typically the form or data module, that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner

24.1.11.3 TSQLDataSet Methods

The methods of the TSQLDataSet class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 1560)	Executes a query or stored procedure that does not return a set of records.

24.1.11.3.1 TSQLDataSet.ExecSQL Method

Executes a query or stored procedure that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 1464) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 1465) property.

Note: Do not use ExecSQL for commands that return data. These include any use of the ctTable command type, SELECT queries, and stored procedures that return a cursor. When the command returns data, use the Open method or set the Active (see page 1453) property to true.

See Also

CommandText (see page 1455)

RowsAffected (see page 1465)

Open

Active

Prepared (see page 1464)

ExecuteDirect (see page 1518)

Executing the Command

24.1.11.4 TSQLDataSet Properties

The properties of the TSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 1561)	Specifies the command the dataset executes.
	CommandType (see page 1562)	Command type of CommandText (see page 1561) property.
	DataSource (see page 1563)	Links the SQL dataset to another (master) dataset.
	MaxBlobSize (see page 1564)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 1564)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1565)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1562) is ctTable.
	SQLConnection (see page 1566)	Specifies the SQL connection component that connects the dataset to a database server.

24.1.11.4.1 TSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 1562) property:

When CommandType (see page 1562) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL (see page 1560) method is called.

When CommandType (see page 1562) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL (see page 1560) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 1562) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 1566) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 1565) property to supply parameter values.

This command is passed on to the TDBXCommand.Text property.

See Also

TDBXCommand

TDBXCommand.CommandType

Params (see page 1565)

ExecSQL (see page 1560)

Open

SortFieldNames (see page 1566)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

24.1.11.4.2 TSQLDataSet.CommandType Property

Command type of CommandText (see page 1561) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the command type of the CommandText (see page 1561) property.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 1561)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

Note: Changing CommandType automatically changes the DbxCommandType property and vice versa. The additional settings supported by DbxCommandType are converted to ctQuery for CommandType.

See Also

CommandText (see page 1561)

DbxCommandType

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

24.1.11.4.3 TSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 1561) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 1565) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1561)

SQL (see page 1587)

Params (see page 1565)

MasterSource (see page 1615)

24.1.11.4.4 TSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1565)

SQLConnection (see page 1566)

24.1.11.4.5 TSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 1565) property is updated automatically to reflect any parameters that appear in the query specified by the CommandText (see page 1561) property (or the SQL property for TSQLQuery (see page 1577)). That is, whenever the CommandText (see page 1561) property changes, the Params (see page 1565) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they

are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 1565) property in response to changes in CommandText (see page 1561), set ParamCheck to false.

See Also

Params (see page 1565)

CommandText (see page 1561)

SQL (see page 1587)

24.1.11.4.6 TSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 1561) property (or the SQL property in TSQLQuery (see page 1577)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 1561) property (or the StoredProcName property in TSQLStoredProc (see page 1594)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 1563) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1564)

SQL (see page 1587)

StoredProcName (see page 1603)

CommandText (see page 1561)

TParam

Value

DataType

DataSource (see page 1457)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

24.1.11.4.7 TSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 1562) is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 1562) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1562)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

24.1.11.4.8 TSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 1510) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 1510) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 1561) property or populate the dataset with data by setting Active (see page 1453) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 1565) property when you set CommandText (see page 1561).

See Also

Active

CommandText (see page 1561)

Params (see page 1565)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

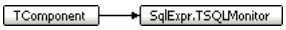
Connecting Directly to a Database Server

Connecting to the Database Server

24.1.12 TSQLMonitor Class

TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.

Class Hierarchy



Pascal

```
TSQLMonitor = class(TComponent);
```

C++

```
class TSQLMonitor : public TComponent;
```

File

SqlExpr

Description

Use TSQLMonitor when debugging the communication between your application and a database server. Each TSQLMonitor instance logs the SQL commands from a particular SQL connection component, adding them to a string list. This allows you to see not only the commands you explicitly add to a SQL dataset or SQL connection component, but also any commands that are generated on your behalf, such as those created by a dataset provider. Use the TraceList (see page 1577) property to access the list of SQL commands that has already been logged. The TraceList (see page 1577) property is automatically updated when the connection component passes a message to the database server.

Note: TSQLMonitor uses the TraceCallbackEvent property of the SQL connection component whose messages it monitors. If you assign your own callback to that property, the SQL monitor instance will cease to function.

See Also

TSQLConnection (see page 1510)

TSTringList

TDataSetProvider

TSQLConnection (see page 1510)

Debugging dbExpress Applications

24.1.12.1 TSQLMonitor Members

The following tables list the members exposed by TSQLMonitor.

Protected Methods

	Name	Description
	InvokeCallBack (see page 1569)	This is InvokeCallBack, a member of class TSQLMonitor.
	SetActive (see page 1571)	This is SetActive, a member of class TSQLMonitor.
	SetFileName (see page 1571)	This is SetFileName, a member of class TSQLMonitor.
	SetSQLConnection (see page 1571)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 1571)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 1571)	This is SetTraceList, a member of class TSQLMonitor.
	SwitchConnection (see page 1572)	This is SwitchConnection, a member of class TSQLMonitor.
V	Trace (see page 1572)	This is Trace, a member of class TSQLMonitor.
	UpdateTraceCallBack (see page 1572)	This is UpdateTraceCallBack, a member of class TSQLMonitor.

Public Methods

	Name	Description
V	Create (see page 1568)	Creates and initializes new TSQLMonitor (see page 1567) object.
V	Destroy (see page 1569)	Frees the memory associated with the TSQLMonitor (see page 1567) object.
	LoadFromFile (see page 1570)	Sets the contents of TraceList (see page 1577) to the strings contained in a file.
	SaveToFile (see page 1570)	Saves the contents of TraceList (see page 1577) to a file.

Public Properties

	Name	Description
	MaxTraceCount (see page 1574)	Indicates the maximum number of messages that can be logged.
R	TraceCount (see page 1576)	Indicates the number of messages currently logged.

Published Properties

	Name	Description
	Active (see page 1573)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 1573)	Specifies whether the monitored messages are automatically saved to a file.
	FileName (see page 1574)	Specifies the default file to which monitored messages are saved.
	OnLogTrace (see page 1575)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 1575)	Occurs when a message is detected but before it is logged to the list.
	SQLConnection (see page 1576)	Specifies the connection components whose messages are to be monitored.
	TraceList (see page 1577)	Lists the messages that have passed between SQLConnection (see page 1576) and the database server.

24.1.12.2 TSQLMonitor.Create Constructor

Creates and initializes new TSQLMonitor (see page 1567) object.

Pascal

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

C++

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

Description

Call Create to create a TSQLMonitor (see page 1567) component at runtime. SQL monitors added to forms or data modules at design time are created automatically.

The Owner parameter specifies the component (usually the form or data module) that is responsible for freeing the SQL monitor.

See Also

Owner

24.1.12.3 TSQLMonitor.Destroy Destructor

Frees the memory associated with the TSQLMonitor (see page 1567) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLMonitor();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the SQL monitor instance is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

24.1.12.4 TSQLMonitor Methods

The methods of the TSQLMonitor class are listed here.

Protected Methods

	Name	Description
	InvokeCallBack (see page 1569)	This is InvokeCallBack, a member of class TSQLMonitor.
	SetActive (see page 1571)	This is SetActive, a member of class TSQLMonitor.
	SetFileName (see page 1571)	This is SetFileName, a member of class TSQLMonitor.
	SetSQLConnection (see page 1571)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 1571)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 1571)	This is SetTraceList, a member of class TSQLMonitor.
	SwitchConnection (see page 1572)	This is SwitchConnection, a member of class TSQLMonitor.
	Trace (see page 1572)	This is Trace, a member of class TSQLMonitor.
	UpdateTraceCallBack (see page 1572)	This is UpdateTraceCallBack, a member of class TSQLMonitor.

Public Methods

	Name	Description
	LoadFromFile (see page 1570)	Sets the contents of TraceList (see page 1577) to the strings contained in a file.
	SaveToFile (see page 1570)	Saves the contents of TraceList (see page 1577) to a file.

24.1.12.4.1 TSQLMonitor.InvokeCallBack Method

Pascal

```
function InvokeCallBack(CallType: TRACECat; CBInfo: Pointer): CBRType; stdcall;
```

C++

```
__fastcall __stdcall CBRType InvokeCallBack(TRACECat CallType, void * CBInfo);
```

Description

This is InvokeCallBack, a member of class TSQLMonitor.

24.1.12.4.2 TSQLMonitor.LoadFromFile Method

Sets the contents of TraceList (see page 1577) to the strings contained in a file.

Pascal

```
procedure LoadFromFile(AFileName: string);
```

C++

```
__fastcall LoadFromFile(UnicodeString AFileName);
```

Description

Call LoadFromFile to initialize the message log with the contents of a file. This method can be used to start the log with standard header information you have saved in a file, or to pre-load the list with a log you saved previously using the SaveToFile (see page 1570) method.

AFileName is the file from which to load the contents of the message log. If AFileName is an empty string, LoadFromFile uses the value of the FileName (see page 1574) property.

Note: Any information currently in TraceList (see page 1577) is lost when you call LoadFromFile.

See Also

TraceList (see page 1577)

SaveToFile (see page 1570)

FileName (see page 1574)

24.1.12.4.3 TSQLMonitor.SaveToFile Method

Saves the contents of TraceList (see page 1577) to a file.

Pascal

```
procedure SaveToFile(AFileName: string);
```

C++

```
__fastcall SaveToFile(UnicodeString AFileName);
```

Description

Call SaveToFile to save all of the currently logged messages to a file.

AFileName specifies the file to which the strings in TraceList (see page 1577) are saved. If AFileName is an empty string, SaveToFile uses the value of the FileName (see page 1574) property.

Note: If the AutoSave (see page 1573) property is true, SaveToFile is called automatically whenever a message is added to TraceList (see page 1577).

See Also

TraceList (see page 1577)

LoadFromFile (see page 1570)

FileName (see page 1574)

AutoSave (see page 1573)

24.1.12.4.4 TSQLMonitor.SetActive Method

Pascal

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

C++

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

Description

This is SetActive, a member of class TSQLMonitor.

24.1.12.4.5 TSQLMonitor.SetFileName Method

Pascal

```
procedure SetFileName(const Value: String);
```

C++

```
__fastcall SetFileName(const UnicodeString Value);
```

Description

This is SetFileName, a member of class TSQLMonitor.

24.1.12.4.6 TSQLMonitor.SetSQLConnection Method

Pascal

```
procedure SetSQLConnection(Value: TSQLConnection);
```

C++

```
__fastcall SetSQLConnection(TSQLConnection Value);
```

Description

This is SetSQLConnection, a member of class TSQLMonitor.

24.1.12.4.7 TSQLMonitor.SetStreamedActive Method

Pascal

```
procedure SetStreamedActive;
```

C++

```
__fastcall SetStreamedActive();
```

Description

This is SetStreamedActive, a member of class TSQLMonitor.

24.1.12.4.8 TSQLMonitor.SetTraceList Method

Pascal

```
procedure SetTraceList(Value: TWideStrings);
```

C++

```
__fastcall SetTraceList(TWideStrings Value);
```

Description

This is SetTraceList, a member of class TSQLMonitor.

24.1.12.4.9 TSQLMonitor.SwitchConnection Method

Pascal

```
procedure SwitchConnection(const Value: TSQLConnection);
```

C++

```
__fastcall SwitchConnection(const TSQLConnection Value);
```

Description

This is SwitchConnection, a member of class TSQLMonitor.

24.1.12.4.10 TSQLMonitor.Trace Method

Pascal

```
procedure Trace(Desc: pSQLTraceDesc; LogTrace: Boolean); virtual;
```

C++

```
virtual __fastcall Trace(pSQLTraceDesc Desc, Boolean LogTrace);
```

Description

This is Trace, a member of class TSQLMonitor.

24.1.12.4.11 TSQLMonitor.UpdateTraceCallBack Method

Pascal

```
procedure UpdateTraceCallBack;
```

C++

```
__fastcall UpdateTraceCallBack();
```

Description

This is UpdateTraceCallBack, a member of class TSQLMonitor.

24.1.12.5 TSQLMonitor Properties

The properties of the TSQLMonitor class are listed here.

Public Properties

	Name	Description
	MaxTraceCount (see page 1574)	Indicates the maximum number of messages that can be logged.
R	TraceCount (see page 1576)	Indicates the number of messages currently logged.

Published Properties

	Name	Description
	Active (see page 1573)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 1573)	Specifies whether the monitored messages are automatically saved to a file.
	FileName (see page 1574)	Specifies the default file to which monitored messages are saved.

	OnLogTrace (see page 1575)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 1575)	Occurs when a message is detected but before it is logged to the list.
	SQLConnection (see page 1576)	Specifies the connection components whose messages are to be monitored.
	TraceList (see page 1577)	Lists the messages that have passed between SQLConnection (see page 1576) and the database server.

24.1.12.5.1 TSQLMonitor.Active Property

Starts or stops the monitor from recording SQL commands passed to the database server.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to start monitoring the communication between SQLConnection (see page 1576) and its database server. Set Active to false to stop the monitoring.

When you set Active to true, TSQLMonitor (see page 1567) sets the TraceCallbackEvent property of the associated SQL connection component. This causes all commands sent to the database server that match the criteria established by TraceFlags to be added to the monitor's string list.

When you set Active to false, TSQLMonitor (see page 1567) clears the TraceCallbackEvent property of the associated SQL connection component. No subsequent commands are added to TraceList (see page 1577), but the list is not cleared. That is, any commands currently in the list remain there until you explicitly clear the list.

See Also

TraceCallbackEvent (see page 1549)

SQLConnection (see page 1576)

TraceList (see page 1577)

TraceFlags

Debugging dbExpress Applications

Fetching the Data using TSQLDataSet

Using TSimpleDataSet

Using TSQLQuery

Using TSQLStoredProc

24.1.12.5.2 TSQLMonitor.AutoSave Property

Specifies whether the monitored messages are automatically saved to a file.

Pascal

```
property AutoSave: Boolean;
```

C++

```
__property Boolean AutoSave;
```

Description

Set AutoSave to true to automatically save all messages that match the criteria established by TraceFlags to the file specified by the FileName (see page 1574) property. The FileName (see page 1574) property must be set to a valid file name.

When AutoSave is false, messages that match the criteria specified by TraceFlags are automatically added to TraceList (see page 1577), but are not saved to a file.

See Also

FileName (see page 1574)

TraceList (see page 1577)

TraceFlags

SaveToFile (see page 1570)

Debugging dbExpress Applications

24.1.12.5.3 TSQLMonitor.FileName Property

Specifies the default file to which monitored messages are saved.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Set FileName to the name of a file to which the messages listed in TraceList (see page 1577) are saved. FileName can be a fully-qualified path name.

Once FileName is specified, setting AutoSave (see page 1573) to true causes all monitored messages to be automatically saved to the specified file. Even if AutoSave (see page 1573) is false, FileName specifies the default file name used by the LoadFromFile (see page 1570) and SaveToFile (see page 1570) methods when they are called with an empty string for the AFileName parameter.

See Also

AutoSave (see page 1573)

SaveToFile (see page 1570)

TraceList (see page 1577)

Debugging dbExpress Applications

24.1.12.5.4 TSQLMonitor.MaxTraceCount Property

Indicates the maximum number of messages that can be logged.

Pascal

```
property MaxTraceCount: Integer;
```

C++

```
__property int MaxTraceCount;
```

Description

Set MaxTraceCount to limit the number of messages that can be added to the TraceList (see page 1577) property. When MaxTraceCount is 0 or greater, the SQL monitor logs every message until the limit specified by MaxTraceCount. Once that limit has been reached, the SQL monitor discards messages until the number of messages in TraceList (see page 1577) has been reduced or the value of MaxTraceCount is changed.

If MaxTraceCount is –1, there is no limit on the number of messages that can be added to TraceList (see page 1577).

If MaxTraceCount is 0, the SQL monitor does not log any messages.

Note: Setting MaxTraceCount affects whether messages can be added to TraceList (see page 1577), but does not affect messages that have already been logged. Thus, if you set MaxTraceCount to a value less than the number of strings currently in TraceList (see page 1577), the number of logged messages will exceed MaxTraceCount.

See Also

TraceList (see page 1577)

TraceCount (see page 1576)

24.1.12.5.5 TSQLMonitor.OnLogTrace Property

Occurs immediately after a new message is logged to the list.

Pascal

```
property OnLogTrace: TTraceLogEvent;
```

C++

```
__property TTraceLogEvent OnLogTrace;
```

Description

Write (see page 3789) an OnLogTrace event handler to respond immediately after a new string is added to the TraceList (see page 1577) property. For example, you can use this event to periodically save the contents of the list to a file and clear the list so that it does not get prohibitively long. OnLogTrace occurs after the message is added to TraceList (see page 1577) but before the list is saved to a file when AutoSave (see page 1573) is true.

Note: OnLogTrace occurs only after logging messages that meet the criteria specified by TraceFlags.

See Also

TraceList (see page 1577)

OnTrace (see page 1575)

TraceFlags

AutoSave (see page 1573)

Debugging dbExpress Applications

24.1.12.5.6 TSQLMonitor.OnTrace Property

Occurs when a message is detected but before it is logged to the list.

Pascal

```
property OnTrace: TTraceEvent;
```

C++

```
__property TTraceEvent OnTrace;
```

Description

Write (see page 3789) an OnTrace event handler to screen or edit messages before they are added to TraceList (see page 1577). OnTrace occurs for every message that passes between the dbExpress driver and the database server, not just those that meet the criteria specified by TraceFlags.

See Also

TraceList (see page 1577)

OnLogTrace (see page 1575)

TraceFlags

24.1.12.5.7 TSQLMonitor.SQLConnection Property

Specifies the connection components whose messages are to be monitored.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Set SQLConnection to identify the TSQLConnection (see page 1510) component whose communication with a database server you want to monitor.

Note: Changing the value of SQLConnection automatically sets the Active (see page 1573) property to false.

See Also

Active (see page 1573)

Debugging dbExpress Applications

24.1.12.5.8 TSQLMonitor.TraceCount Property

Indicates the number of messages currently logged.

Pascal

```
property TraceCount: Integer;
```

C++

```
__property int TraceCount;
```

Description

TraceCount indicates the number of messages listed by the TraceList (see page 1577) property.

See Also

TraceList (see page 1577)

MaxTraceCount (see page 1574)

24.1.12.5.9 TSQLMonitor.TraceList Property

Lists the messages that have passed between SQLConnection (see page 1576) and the database server.

Pascal

```
property TraceList: TWideStrings;
```

C++

```
__property TWideStrings TraceList;
```

Description

Use the TraceList property to access the list of SQL commands that has already been logged. While the SQL monitor is active, the TraceList property is automatically updated when the connection component passes a message to the database server.

Only messages that meet the criteria specified by the TraceFlags property are added to TraceList.

TraceList is a TStrings descendant. As such, you can use its methods to

Clear the list.

Access the individual commands in the list.

Add your own comments to the list. For example, you may want to indicate when your application initiates certain actions so that you can see clearly what SQL messages correspond to what actions.

See Also

SQLConnection (see page 1576)

Active (see page 1573)

TraceFlags

SaveToFile (see page 1570)

LoadFromFile (see page 1570)

TraceCount (see page 1576)

Debugging dbExpress Applications

24.1.13 TSQLQuery Class

TSQLQuery represents a query that is executed using dbExpress.

Class Hierarchy

Pascal

```
TSQLQuery = class(TCustomSQLDataSet);
```

C++

```
class TSQLQuery : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLQuery to execute an SQL (see page 1587) command on a database server that is accessed via TSQLConnection (see page 1510). TSQLQuery can represent the results of a SELECT statement or perform actions on the database server using statements such as INSERT, DELETE, UPDATE, ALTER TABLE, and so on. You can add a TSQLQuery component to a form or data module at design time, or create one dynamically at runtime.

TSQLQuery is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL (see page 1587) query by explicitly creating an SQL (see page 1587) UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLQuery component, it must be connected to the database server. Therefore, the first step to take when working with TSQLQuery is to set the SQLConnection (see page 1588) property. Once the query is connected to a server, use the SQL (see page 1587) property to specify the command your query executes.

See Also

TSQLDataSet (see page 1556)

TSQLStoredProc (see page 1594)

TSQLTable (see page 1604)

TSQLConnection (see page 1510)

TClientDataSet

TDataSetProvider

Types of Datasets

Specifying the Command to Execute

Types of Unidirectional Datasets

24.1.13.1 TSQLQuery Members

The following tables list the members exposed by TSQLQuery.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.

	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
V	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.

	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Methods

	Name	Description
	Create (see page 1581)	Creates an instance of a TSQLQuery (see page 1577) component.
	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1581)	Destroys the TSQLQuery (see page 1577) instance.
	ExecSQL (see page 1582)	Executes a query that does not return a set of records.
	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
	PrepareStatement (see page 1583)	Prepares the query for execution.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	RowsAffected (see page 1586)	Indicates the number of rows operated on by the latest query execution.
	Text (see page 1588)	Indicates the SQL (see page 1587) statement as a single string.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.

	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	DataSource (see page 1584)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 1584)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1585)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 1585)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1586)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	SQL (see page 1587)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 1588)	Specifies the SQL connection component that connects the dataset to a database server.

24.1.13.2 TSQLQuery.Create Constructor

Creates an instance of a TSQLQuery (see page 1577) component.

Pascal

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

C++

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

Description

Call Create to instantiate an SQL (see page 1587) query component at runtime. SQL (see page 1587) queries placed in forms or data modules at design time are created automatically.

See Also

Create

24.1.13.3 TSQLQuery.Destroy Destructor

Destroys the TSQLQuery (see page 1577) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLQuery();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1581)

System::TObject::Free (see page 3382)

24.1.13.4 TSQLQuery Methods

The methods of the TSQLQuery class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 1582)	Executes a query that does not return a set of records.
	PrepareStatement (see page 1583)	Prepares the query for execution.

24.1.13.4.1 TSQLQuery.ExecSQL Method

Executes a query that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL (see page 1587) command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 1464) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 1586) property.

Note: Do not use ExecSQL for commands such as SELECT queries that return data. When the command returns data, use the Open method or set the Active (see page 1453) property to true.

See Also

SQL (see page 1587)

RowsAffected (see page 1465)

Open

Active

Prepared (see page 1464)

ExecuteDirect (see page 1518)

24.1.13.4.2 TSQLQuery.PrepareStatement Method

Prepares the query for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a query to be executed on the server. It ensures the dbExpress and the database server have allocated any resources they need for executing the query and binding its parameters. Any named parameters are temporarily converted to unnamed parameters because dbExpress does not support named parameters.

Do not call PrepareStatement when preparing a query for execution. Instead, set the Prepared (see page 1464) property to true. When you use the Prepared (see page 1464) property instead, the query keeps track of the fact that it has already been prepared and need not repeat the process every time it executes the query. In addition, when you set the Prepared (see page 1464) property, the query initializes internal properties that keep track of the number of rows affected by the query.

See Also

Prepared (see page 1464)

24.1.13.5 TSQLQuery Properties

The properties of the TSQLQuery class are listed here.

Public Properties

	Name	Description
	RowsAffected (see page 1586)	Indicates the number of rows operated on by the latest query execution.
R	Text (see page 1588)	Indicates the SQL (see page 1587) statement as a single string.

Published Properties

	Name	Description
	DataSource (see page 1584)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 1584)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1585)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 1585)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1586)	Represents the parameters for a query or stored procedure.
	SQL (see page 1587)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 1588)	Specifies the SQL connection component that connects the dataset to a database server.

24.1.13.5.1 TSQLQuery.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1455)

SQL (see page 1587)

Params (see page 1463)

MasterSource (see page 1615)

24.1.13.5.2 TSQLQuery.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider

Fetching the Data using TSQLDataSet

24.1.13.5.3 TSQLQuery.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1546)

SQLConnection (see page 1467)

24.1.13.5.4 TSQLQuery.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1463)

CommandText (see page 1455)

SQL (see page 1587)

24.1.13.5.5 TSQLQuery.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1462)

SQL (see page 1587)

StoredProcName (see page 1603)

CommandText (see page 1455)

TParam

Value

DataType

DataSource (see page 1457)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

24.1.13.5.6 TSQLQuery.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 1560)

RecordCount (see page 1465)

ExecProc (see page 1599)

24.1.13.5.7 TSQLQuery.SQL Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property SQL: TWideStrings;
```

C++

```
__property TWideStrings SQL;
```

Description

Use the properties and methods of SQL to specify the query that should be executed on the database server. If SQL is a SELECT statement, the query executes the statement when you call Open or set the Active (see page 1453) property to true. If the statement does not return a result set, call the ExecSQL (see page 1582) method to execute the statement.

When you set the SQL property, TSQLQuery (see page 1577) checks the statement for parameters and initializes the Params (see page 1586) property to reflect any parameters in the statement. You can subsequently use the Params (see page 1586) property to adjust the datatype of any parameters and supply values for the parameters.

At design-time, edit the SQL statement using the property editor that appears when you click the ellipsis button for the property in the Object Inspector. At runtime, use properties and methods of TStrings to clear the current contents, add new contents, or to modify existing contents:

SQLQuery1.SQL.Clear;

```
SQLQuery1.SQL.Add('SELECT ' + Edit1.Text + ' FROM ' + Edit2.Text);
if Length(Edit3.Text) <> 0 then
```

SQLQuery1.SQL.Add('ORDER BY ' + Edit3.Text (see page 1588))

SQLQuery1->SQL->Clear();

```
SQLQuery1->SQL->Add("SELECT " + Edit1->Text + " FROM " + Edit2.Text);
if (!Edit3->Text->IsEmpty())
```

SQLQuery1->SQL->Add("ORDER BY " + Edit3.Text (see page 1588));

You can also use the methods of SQL to load a query from a file.

Note: The SQL you supply for this property must be valid for the database server to which the TSQLQuery (see page 1577)

object is connected.

See Also

Add

Active

Open

ExecSQL (see page 1582)

Params (see page 1463)

Text (see page 1588)

Representing the Results of a Query

Specifying the Command to Execute

24.1.13.5.8 TSQLQuery.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1455)

Params (see page 1463)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

24.1.13.5.9 TSQLQuery.Text Property

Indicates the SQL (see page 1587) statement as a single string.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text as a convenient way to read the entire SQL (see page 1587) query as a single string, rather than using the SQL (see page 1587) property, which may divide the query into substrings. Text is the same as the Text property of the TStrings object that implements the SQL (see page 1587) property.

See Also

SQL (see page 1587)

Text

24.1.14 TSqlServerMethod Class

TDataSet used to call DataSnap server methods.

Class Hierarchy

Pascal

```
TSqlServerMethod = class(TCustomSQLDataSet);
```

C++

```
class TSqlServerMethod : public TCustomSQLDataSet;
```

File

SqlExpr

Description

TSqlServerMethod is a TDataSet used to call DataSnap server methods and retrieve results from them.

See Also

TCustomSQLDataSet (see page 1421)

TDataSet

24.1.14.1 TSqlServerMethod Members

The following tables list the members exposed by TSqlServerMethod.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.

	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.

	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1457)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	MaxBlobSize (see page 1460)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1462)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1592)	Create new instance of TSqlServerMethod (see page 1589).
V	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	Destroy (see page 1426)	Destroys the TCustomSQLDataSet (see page 1421) instance.
	ExecuteMethod (see page 1593)	Executes remote server method.
V	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
V	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
V	PrepareStatement (see page 1440)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.

	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Params (see page 1593)	TParams for method to be executed.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	ServerMethodName (see page 1594)	Sets method name of remote DataSnap server method.
	SQLConnection (see page 1594)	TSQLConnection (see page 1510) used.

24.1.14.2 TSqlServerMethod.Create Constructor

Create new instance of TSqlServerMethod (see page 1589).

Pascal

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

C++

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

Description

This method creates a new instance of TSqlServerMethod (see page 1589).

24.1.14.3 TSqlServerMethod Methods

The methods of the TSqlServerMethod class are listed here.

Public Methods

	Name	Description
	ExecuteMethod (see page 1593)	Executes remote server method.

24.1.14.3.1 TSqlServerMethod.ExecuteMethod Method

Executes remote server method.

Pascal

```
procedure ExecuteMethod;
```

C++

```
__fastcall ExecuteMethod();
```

Description

ExecuteMethod executes a remote server method. It does not return a result set. Instead, it uses the property Params (see page 1593) for input and output parameters of the server method.

See Also

Params (see page 1593)

24.1.14.4 TSqlServerMethod Properties

The properties of the TSqlServerMethod class are listed here.

Published Properties

	Name	Description
	Params (see page 1593)	TParams for method to be executed.
	ServerMethodName (see page 1594)	Sets method name of remote DataSnap server method.
	SQLConnection (see page 1594)	TSQLConnection (see page 1510) used.

24.1.14.4.1 TSqlServerMethod.Params Property

TParams for method to be executed.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params contains TParams for a server method to be executed by ExecuteMethod (see page 1593). ExecuteMethod (see page 1593) uses Params for input and output parameters of the server method.

See Also

TParams

ExecuteMethod (see page 1593)

24.1.14.4.2 TSqlServerMethod.ServerMethodName Property

Sets method name of remote DataSnap server method.

Pascal

```
property ServerMethodName: UnicodeString;
```

C++

```
__property UnicodeString ServerMethodName;
```

Description

ServerMethodName sets the method name of the remote DataSnap server method to be executed by ExecuteMethod (see page 1593).

The method name may also be specified in CommandText (see page 1455). In this case, it is of the form *partialClassName.methodName*, where *partialClassName* is the name of the server methods class without the T prefix and *methodName* is the server method name.

See Also

CommandText

Params (see page 1593)

ExecuteMethod (see page 1593)

24.1.14.4.3 TSqlServerMethod.SQLConnection Property

TSQLConnection (see page 1510) used.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

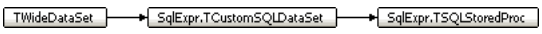
Description

SQLConnection contains the TSQLConnection (see page 1510).

24.1.15 TSQLStoredProc Class

TSQLStoredProc represents a stored procedure that is executed using dbExpress.

Class Hierarchy



Pascal

```
TSQLStoredProc = class(TCustomSQLDataSet);
```

C++

```
class TSQLStoredProc : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLStoredProc to execute a stored procedure on a database server that is accessed via TSQLConnection (see page 1510). TSQLStoredProc can represent the result set if the stored procedure returns a cursor. You can add a TSQLStoredProc component to a form or data module at design time, or create one dynamically at runtime.

TSQLStoredProc is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL stored procedure by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLStoredProc component, it must be connected to the database server. Therefore, the first step to take when working with TSQLStoredProc is to set the SQLConnection (see page 1603) property. Once the dataset is connected to a server, use the StoredProcName (see page 1603) property to specify the stored procedure to execute.

See Also

TSQLDataSet (see page 1556)

TSQLQuery (see page 1577)

TSQLTable (see page 1604)

TSQLConnection (see page 1510)

TClientDataSet

TDataSetProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Types of Unidirectional Datasets

24.1.15.1 TSQLStoredProc Members

The following tables list the members exposed by TSQLStoredProc.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.

	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1439)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
V	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1457)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1598)	Creates an instance of a TSQLStoredProc (see page 1594) component.
V	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	Destroy (see page 1426)	Destroys the TCustomSQLDataSet (see page 1421) instance.
V	ExecProc (see page 1599)	Executes the stored procedure when it does not return a cursor.
V	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
V	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	NextRecordSet (see page 1599)	Provides access to a secondary recordset.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
V	PrepareStatement (see page 1600)	Prepares the stored procedure for execution.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.

	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.
	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1453)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1600)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PackageName (see page 1601)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 1601)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1602)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	SQLConnection (see page 1603)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 1603)	Specifies the SQL statement to execute on the database server.

24.1.15.2 TSQLStoredProc.Create Constructor

Creates an instance of a TSQLStoredProc (see page 1594) component.

Pascal

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

C++

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

Description

Call Create to instantiate an SQL stored procedure component at runtime. SQL stored procedures placed in forms or data modules at design time are created automatically.

AOwner is the component, typically a form or data module, that is responsible for freeing the TSQLStoredProc (see page 1594) instance. It becomes the value of the Owner property.

See Also

Owner

Create

24.1.15.3 TSQLStoredProc Methods

The methods of the TSQLStoredProc class are listed here.

Public Methods

	Name	Description
	ExecProc (see page 1599)	Executes the stored procedure when it does not return a cursor.
	NextRecordSet (see page 1599)	Provides access to a secondary recordset.
	PrepareStatement (see page 1600)	Prepares the stored procedure for execution.

24.1.15.3.1 TSQLStoredProc.ExecProc Method

Executes the stored procedure when it does not return a cursor.

Pascal

```
function ExecProc: Integer; virtual;
```

C++

```
virtual __fastcall int ExecProc();
```

Description

Call ExecProc to execute the stored procedure specified by StoredProcName (see page 1603) if it does not return a cursor. ExecProc returns the number of rows affected by the stored procedure. This becomes the value of the RowsAffected (see page 1465) property.

To speed performance, an application should ordinarily prepare the stored procedure by setting the Prepared (see page 1464) property to true before calling ExecProc the first time.

Note: Do not use ExecProc for stored procedures that return a cursor. When the stored procedure returns data, use the Open method or set the Active (see page 1453) property to true.

See Also

StoredProcName (see page 1603)

RowsAffected (see page 1465)

Prepared (see page 1464)

Executing the Command

24.1.15.3.2 TSQLStoredProc.NextRecordSet Method

Provides access to a secondary recordset.

Pascal

```
function NextRecordSet: TCustomSQLDataSet;
```

C++

```
__fastcall TCustomSQLDataSet NextRecordSet();
```

Description

Call NextRecordSet to obtain secondary datasets when accessing the results of a stored procedure that returns multiple sets of records. Opening the dataset fetches the first set of records. NextRecordSet provides access to the subsequent sets.

NextRecordSet returns a newly created TCustomSQLDataSet (see page 1421) component that provides access to the next

set of records. The first time you call NextRecordSet, it returns a dataset for the second set of records. The next call returns a third dataset, and so on, until there are no more sets of records. When there are no additional datasets, NextRecordSet returns nil (Delphi) or NULL (C++).

See Also

Fetching the Data

Fetching the Data using TSQLDataSet

24.1.15.3.3 TSQLStoredProc.PrepareStatement Method

Prepares the stored procedure for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a stored procedure to be executed on the server. It ensures that dbExpress and the database server have allocated any resources they need for executing the stored procedure and binding its parameters.

Do not call PrepareStatement when preparing a stored procedure for execution. Instead, set the Prepared (see page 1464) property to true. When you use the Prepared (see page 1464) property instead, the stored procedure component keeps track of the fact that it has already been prepared and need not repeat the process every time it executes. In addition, when you set the Prepared (see page 1464) property, the component initializes internal properties that keep track of the number of rows affected by the stored procedure.

See Also

Prepared (see page 1464)

24.1.15.4 TSQLStoredProc Properties

The properties of the TSQLStoredProc class are listed here.

Published Properties

	Name	Description
	MaxBlobSize (see page 1600)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	PackageName (see page 1601)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 1601)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1602)	Represents the parameters for a query or stored procedure.
	SQLConnection (see page 1603)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 1603)	Specifies the SQL statement to execute on the database server.

24.1.15.4.1 TSQLStoredProc.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1546)

SQLConnection (see page 1467)

24.1.15.4.2 TSQLStoredProc.PackageName Property

Specifies the name of the Oracle package to which the stored procedure belongs.

Pascal

```
property PackageName: WideString;
```

C++

```
__property BSTR PackageName;
```

Description

Set PackageName to specify the name of the Oracle package that contains the stored procedure which the StoredProcName (see page 1603) property specifies. This property is only used when the stored procedure is defined on an Oracle server.

If the stored procedure does not belong to a package, the value of PackageName is an empty string.

See Also

StoredProcName (see page 1603)

GetPackageNames (see page 1522)

24.1.15.4.3 TSQLStoredProc.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1463)

CommandText (see page 1455)

SQL (see page 1587)

24.1.15.4.4 TSQLStoredProc.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1462)

SQL (see page 1587)

StoredProcName (see page 1603)

CommandText (see page 1455)

TParam

Value

DataType

DataSource (see page 1457)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

24.1.15.4.5 TSQLStoredProc.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1455)

Params (see page 1463)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

24.1.15.4.6 TSQLStoredProc.StoredProcName Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property StoredProcName: WideString;
```

C++

```
__property BSTR StoredProcName;
```

Description

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, an exception is raised.

When you set the StoredProcName property, TSQLStoredProc (see page 1594) queries the database server to ascertain the parameters of the specified stored procedure. It uses this information to initialize the Params (see page 1602) property. You can subsequently use the Params (see page 1602) property to adjust the properties of any parameters and supply values for the parameters. Some servers do not supply information about stored procedure parameters. In those cases, you must explicitly add the parameters.

See Also

Params (see page 1463)

PackageName (see page 1601)

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSQLStoredProc

24.1.16 TSQLTable Class

TSQLTable represents a database table that is accessed using dbExpress.

Class Hierarchy

Pascal

```
TSQLTable = class(TCustomSQLDataSet);
```

C++

```
class TSQLTable : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLTable to represent a table on a database server that is accessed via TSQLConnection (see page 1510). TSQLTable generates a query to fetch all of the rows and columns in a table you specify. You can add a TSQLTable component to a form or data module at design time, or create one dynamically at runtime.

TSQLTable is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL table by explicitly creating an SQL UPDATE command or by connecting the table to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLTable component, it must be connected to the database server. Therefore, the first step to take when working with TSQLTable is to set the SQLConnection (see page 1617) property. Once the dataset is connected to a server, use the TableName (see page 1617) property to specify the table this component represents.

See Also

TSQLDataSet (see page 1556)

TSQLQuery (see page 1577)

TSQLStoredProc (see page 1594)

TSQLConnection (see page 1510)

TClientDataSet

TDataSetProvider

Types of Datasets

Types of Unidirectional Datasets

24.1.16.1 TSQLTable Members

The following tables list the members exposed by TSQLTable.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1428)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1428)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	Check (see page 1428)	This is Check, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1428)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1429)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1429)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1429)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1430)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1430)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1430)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1431)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCursor (see page 1431)	This is FreeCursor, a member of class TCustomSQLDataSet.
	FreeStatement (see page 1431)	This is FreeStatement, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1432)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1432)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1434)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1434)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1435)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1435)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1435)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1435)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1436)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1436)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1436)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1436)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1436)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1437)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1437)	Initializes the component after the form file has been read into memory.

	LocateRecord (see page 1438)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1610)	This is OpenCursor, a member of class TSQLTable.
V	OpenSchema (see page 1439)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1440)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1440)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1441)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1441)	Executes a specified SQL command.
V	PSGetAttributes (see page 1442)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1442)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1443)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1443)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1443)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1444)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1444)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1445)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1445)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1445)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1445)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1446)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1446)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1447)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1447)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1448)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1448)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1448)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1449)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1449)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1449)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1450)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetIndexField (see page 1610)	This is SetIndexField, a member of class TSQLTable.
	SetMaxBlobSize (see page 1450)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1450)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1451)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SQLError (see page 1452)	This is SQLError, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1452)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1454)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1456)	Indicates the meaning of the CommandText (see page 1455) property.
	CurrentBlobSize (see page 1456)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1457)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1457)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 1459)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 1460)	Saves the error message for the last error returned by dbExpress.
R	MasterLink (see page 1615)	This is MasterLink, a member of class TSQLTable.
	NativeCommand (see page 1460)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1462)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1462)	Indicates the number of parameters listed by the Params (see page 1463) property.
	Params (see page 1463)	Represents the parameters for a query or stored procedure.

	ProcParams (see page 1464)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1465)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1466)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1466)	Indicates the sort order when CommandType (see page 1456) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1608)	Creates an instance of a TSQLTable (see page 1604) component.
V	CreateBlobStream (see page 1429)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	DeleteRecords (see page 1609)	Empties the underlying database table.
V	Destroy (see page 1608)	Destroys the TSQLTable (see page 1604) instance.
V	GetBlobFieldData (see page 1431)	Retrieves the current value of a BLOB field into a buffer.
V	GetDetailLinkFields (see page 1432)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1433)	This is the overview for the GetFieldData method overload.
	GetIndexNames (see page 1609)	Retrieves a list of available indexes defined for the table.
	GetKeyFieldNames (see page 1433)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1434)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1437)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1438)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1439)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1440)	This is ParamByName, a member of class TCustomSQLDataSet.
V	PrepareStatement (see page 1610)	Generates the query that TSQLTable (see page 1604) uses to fetch data from the database server.
	SetSchemaInfo (see page 1450)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1455)	Specifies the command the dataset executes.
	DesignerData (see page 1458)	Stores custom data.
	IndexDefs (see page 1459)	Contains the definitions of all indexes defined on the server for the dataset.
R	IndexFieldCount (see page 1613)	Indicates the number of fields that make up the current index.
	IndexFields (see page 1614)	Specifies the fields associated with the current index.
	Prepared (see page 1464)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1465)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1467)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1468)	Occurs after a data source connection is closed.
	AfterOpen (see page 1469)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1469)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1469)	Occurs after a result set is scrolled.
	BeforeClose (see page 1469)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1469)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1470)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1470)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1470)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1470)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1455) property.

	ParseInsertSql (see page 1470)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1455) property.
	ParseSelectSql (see page 1470)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1455) property.
	ParseUpdateSql (see page 1471)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1455) property.

Published Properties

	Name	Description
	Active (see page 1611)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1458)	Specifies whether the SQL dataset fetches metadata information along with the data.
	IndexFieldNames (see page 1613)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 1614)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 1615)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 1615)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 1616)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1461)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1461)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1466)	Specifies the name of the database schema.
	SQLConnection (see page 1617)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 1617)	Specifies the table on the database server represented by this component.

24.1.16.2 TSQLTable.Create Constructor

Creates an instance of a TSQLTable (see page 1604) component.

Pascal

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

C++

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

Description

Call Create to instantiate an SQL table at runtime. SQL tables placed in forms or data modules at design time are created automatically.

See Also

Create

24.1.16.3 TSQLTable.Destroy Destructor

Destroys the TSQLTable (see page 1604) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLTable();
```

Description

Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1608)

System::TObject::Free (see page 3382)

24.1.16.4 TSQLTable Methods

The methods of the TSQLTable class are listed here.

Protected Methods

	Name	Description
	OpenCursor (see page 1610)	This is OpenCursor, a member of class TSQLTable.
	SetIndexField (see page 1610)	This is SetIndexField, a member of class TSQLTable.

Public Methods

	Name	Description
	DeleteRecords (see page 1609)	Empties the underlying database table.
	GetIndexNames (see page 1609)	Retrieves a list of available indexes defined for the table.
	PrepareStatement (see page 1610)	Generates the query that TSQLTable (see page 1604) uses to fetch data from the database server.

24.1.16.4.1 TSQLTable.DeleteRecords Method

Empties the underlying database table.

Pascal

```
procedure DeleteRecords;
```

C++

```
__fastcall DeleteRecords();
```

Description

Call DeleteRecords to delete all the records in the underlying database table.

See Also

RecordCount

24.1.16.4.2 TSQLTable.GetIndexNames Method

Retrieves a list of available indexes defined for the table.

Pascal

```
procedure GetIndexNames(List: TWideStrings);
```

C++

```
__fastcall GetIndexNames(TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of all indexes on the server defined for the table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

See Also

IndexName (see page 1614)

IndexDefs (see page 1459)

Obtaining Information About Indexes

24.1.16.4.3 TSQLTable.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSQLTable.

24.1.16.4.4 TSQLTable.PrepareStatement Method

Generates the query that TSQLTable (see page 1604) uses to fetch data from the database server.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement generates a SELECT query to fetch the records from the table specified by TableName (see page 1617), ordering by the fields in IndexFieldNames (see page 1613) or the index in IndexName (see page 1614). If you have specified a master/detail relationship, PrepareStatement sets that up as well.

Do not call PrepareStatement in your applications. Instead, set the Prepared (see page 1464) property to true. When you use the Prepared (see page 1464) property instead, the SQL table keeps track of the fact that it has already been prepared and need not repeat the process every time it executes.

See Also

Prepared (see page 1464)

TableName (see page 1617)

IndexName (see page 1614)

IndexFieldNames (see page 1613)

24.1.16.4.5 TSQLTable.SetIndexField Method

Pascal

```
procedure SetIndexField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetIndexField(int Index, TField Value);
```

Description

This is SetIndexField, a member of class TSQLTable.

24.1.16.5 TSQLTable Properties

The properties of the TSQLTable class are listed here.

Protected Properties

	Name	Description
	MasterLink (see page 1615)	This is MasterLink, a member of class TSQLTable.

Public Properties

	Name	Description
	IndexFieldCount (see page 1613)	Indicates the number of fields that make up the current index.
	IndexFields (see page 1614)	Specifies the fields associated with the current index.

Published Properties

	Name	Description
	Active (see page 1611)	Specifies whether or not a dataset is open.
	IndexFieldNames (see page 1613)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 1614)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 1615)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 1615)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 1616)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	SQLConnection (see page 1617)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 1617)	Specifies the table on the database server represented by this component.

24.1.16.5.1 TSQLTable.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.

2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      // This path depends on the Demos directory of your RAD Studio installation.
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
end;
```

See Also

DataSource

State

AfterOpen

BeforeOpen

Close

Open

CanModify

BeforeClose

AfterClose

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

24.1.16.5.2 TSQLTable.IndexFieldCount Property

Indicates the number of fields that make up the current index.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Check (see page 1428) IndexFieldCount to determine how many fields are in the current index. If the IndexName (see page 1614) property is not set, IndexFieldCount is 0.

See Also

IndexFieldNames (see page 1613)

IndexDefs (see page 1459)

IndexName (see page 1614)

24.1.16.5.3 TSQLTable.IndexFieldNames Property

Lists the fields to use for sorting records and for linking in master/detail relationships.

Pascal

```
property IndexFieldNames: WideString;
```

C++

```
__property BSTR IndexFieldNames;
```

Description

IndexFieldNames is a semicolon-delimited set of field names, where each name specifies a field in the table.

Use IndexFieldNames to identify the fields in the SQL table that are used to sort records in the table. IndexFieldNames are used to generate the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexFieldNames is also used to link this table as the detail in a master/detail relationship. Each of these fields must be matched to a corresponding field from the MasterFields (see page 1615) property.

Instead of explicitly setting the sorting and linking fields using IndexFieldNames, you can indicate them using an existing index. To specify the sorting and linking fields implicitly through an index, use the IndexName (see page 1614) property instead.

Note: The IndexName (see page 1614) and IndexFieldNames properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexName (see page 1614)

MasterFields (see page 1615)

MasterSource (see page 1615)

Creating an Index with IndexFieldNames

Representing the Records in a Table

24.1.16.5.4 TSQLTable.IndexFields Property

Specifies the fields associated with the current index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexFields directly. Instead use the IndexFieldNames (see page 1613) property to specify the sort order for records in the table.

See Also

IndexDefs (see page 1459)

IndexFieldNames (see page 1613)

IndexName (see page 1614)

24.1.16.5.5 TSQLTable.IndexName Property

Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.

Pascal

```
property IndexName: WideString;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to specify an existing index that is defined for the underlying database table.

IndexName is used to sort records in the table by appearing in the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexName is also used to link this table as the detail in a master/detail relationship. Each of the fields in the index must be matched to a corresponding field from the MasterFields (see page 1615) property.

Note: The IndexName and IndexFieldNames (see page 1613) properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexFieldNames (see page 1613)

MasterFields (see page 1615)

MasterSource (see page 1615)

GetIndexNames (see page 1609)

Representing the Records in a Table

24.1.16.5.6 TSQLTable.MasterFields Property

Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterFields: WideString;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields to identify the fields in another (master) dataset that can be used as link fields in a master/detail relationship. MasterFields is a semicolon-delimited set of field names, where each name identifies a field in the master dataset. The master dataset is the dataset associated with the datasource specified by MasterSource (see page 1615). Each of the fields in MasterFields must be matched to a corresponding field from the IndexFieldNames (see page 1613) property.

At design-time, double-click on the MasterFields property in the Object Inspector to use the Field Link designer to help you define the master/detail relationship. You can only use the Field Link designer if you have already specified the IndexName (see page 1614) or IndexFieldNames (see page 1613) property and the MasterSource (see page 1615) property.

See Also

IndexFieldNames (see page 1613)

IndexName (see page 1614)

MasterSource (see page 1615)

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

24.1.16.5.7 TSQLTable.MasterLink Property

Pascal

```
property MasterLink: TMasterDataLink;
```

C++

```
__property TMasterDataLink MasterLink;
```

Description

This is MasterLink, a member of class TSQLTable.

24.1.16.5.8 TSQLTable.MasterSource Property

Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to indicate the dataset that serves as a master table when establishing a master/detail relationship. Specify a TDataSource component whose DataSet property identifies the desired master dataset.

A master/detail relationship is established when you set the MasterSource property to identify a master dataset, the MasterFields (see page 1615) property to identify the fields in that master dataset whose values must be matched by linking fields in this (detail) SQL table, and the IndexName (see page 1614) or IndexFieldNames (see page 1613) property to identify the linking fields in this SQL table.

MasterSource is the same as the DataSource (see page 1457) property, except that it is published.

See Also

IndexFieldNames (see page 1613)

IndexName (see page 1614)

MasterFields (see page 1615)

DataSource (see page 1457)

DataSet

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

24.1.16.5.9 TSQLTable.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1431)

Params (see page 1546)

SQLConnection (see page 1467)

24.1.16.5.10 TSQLTable.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1455)

Params (see page 1463)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

24.1.16.5.11 TSQLTable.TableName Property

Specifies the table on the database server represented by this component.

Pascal

```
property TableName: WideString;
```

C++

```
__property BSTR TableName;
```

Description

Set TableName to the name of a table on the database server. TSQLTable (see page 1604) automatically generates an SQL query to fetch all of the records and all of the fields in that table.

Note: In the automatically generated SQL query, the ORDER BY clause is determined by the IndexName (see page 1614) or IndexFieldNames (see page 1613) property.

See Also

IndexFieldNames (see page 1613)

IndexName (see page 1614)

Representing the Records in a Table

24.1.17 TTransactionItem Class

Class Hierarchy

Pascal

```
TTransactionItem = class;
```

C++

```
class TTransactionItem;
```

File

SqlExpr

Description

This is class SqlExpr.TTransactionItem.

24.1.17.1 TTransactionItem Members

The following tables list the members exposed by TTransactionItem.

Public Fields

	Name	Description
	FNext (see page 1618)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 1619)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 1619)	This is FTransactionDesc, a member of class TTransactionItem.

24.1.17.2 TTransactionItem Fields

The fields of the TTransactionItem class are listed here.

Public Fields

	Name	Description
	FNext (see page 1618)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 1619)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 1619)	This is FTransactionDesc, a member of class TTransactionItem.

24.1.17.2.1 TTransactionItem.FNext Field

Pascal

```
FNext: TTransactionItem;
```

C++

```
TTransactionItem FNext;
```

Description

This is FNext, a member of class TTransactionItem.

24.1.17.2.2 TTransactionItem.FTransaction Field

Pascal

```
FTransaction: TDBXTransaction;
```

C++

```
TDBXTransaction FTransaction;
```

Description

This is FTransaction, a member of class TTransactionItem.

24.1.17.2.3 TTransactionItem.FTransactionDesc Field

Pascal

```
FTransactionDesc: TTransactionDesc;
```

C++

```
TTransactionDesc FTransactionDesc;
```

Description

This is FTransactionDesc, a member of class TTransactionItem.

24.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeProcParams (see page 1620)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 1620)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 1621)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDelegateDriverNames (see page 1621)	This is function SqlExpr.GetDelegateDriverNames.
	GetDriverNames (see page 1621)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 1622)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 1622)	Fills a TParams object with information from a list of parameter descriptors.
	RegisterDbXpressLib (see page 1623)	This is function SqlExpr.RegisterDbXpressLib.

24.2.1 SqlExpr.FreeProcParams Function

Frees a list of procedure parameters that is allocated by GetProcedureParams.

Pascal

```
procedure FreeProcParams(var ProcParams: TList);
```

C++

```
FreeProcParams(TList ProcParams);
```

File

SqlExpr

Description

Use FreeProcParams to free a list of stored procedure parameters after using them. FreeProcParams takes a single parameter, ProcParams, which is a list of parameter information that was returned from a call to the GetProcedureParams method of TSQLConnection (see page 1510). It frees all of the parameter structures in the list, and then frees the list itself, changing the parameter that was passed in to nil (Delphi) or NULL (C++).

See Also

TSQLConnection (see page 1510)

LoadParamListItems (see page 1622)

24.2.2 SqlExpr.GetConnectionNames Function

Pascal

```
procedure GetConnectionNames(List: TStrings; Driver: string = ''; DesignMode: Boolean = True);
```

C++

```
GetConnectionNames(TStrings List, UnicodeString Driver = '', Boolean DesignMode = True);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionNames.

24.2.3 SqlExpr.GetConnectionRegistryFile Function

Pascal

```
function GetConnectionRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
UnicodeString GetConnectionRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionRegistryFile.

24.2.4 SqlExpr.GetDelegateDriverNames Function

Pascal

```
procedure GetDelegateDriverNames(List: TStrings; DesignMode: Boolean = True);
```

C++

```
GetDelegateDriverNames(TStrings List, Boolean DesignMode = True);
```

File

SqlExpr

Description

This is function SqlExpr.GetDelegateDriverNames.

24.2.5 SqlExpr.GetDriverNames Function

Populates a string list with the names of all available dbExpress drivers.

Pascal

```
procedure GetDriverNames(List: TStrings; DesignMode: Boolean = True);
```

C++

```
GetDriverNames(TStrings List, Boolean DesignMode = True);
```

File

SqlExpr

Description

GetDriverNames fills the string list specified by List with the names of all installed dbExpress drivers. GetDriverNames fetches the names of Installed drivers from the drivers.ini file.

List is a string list object, created and maintained by the application, into which to store the driver names.

DesignMode indicates whether the names are fetched at design time (true) or at runtime (false).

See Also

TSQLConnection (see page 1510)

24.2.6 SqlExpr.GetDriverRegistryFile Function

Pascal

```
function GetDriverRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
UnicodeString GetDriverRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetDriverRegistryFile.

24.2.7 SqlExpr.LoadParamListItems Function

Fills a TParams object with information from a list of parameter descriptors.

Pascal

```
procedure LoadParamListItems(Params: TParams; ProcParams: TList);
```

C++

```
LoadParamListItems(TParams Params, TList ProcParams);
```

File

SqlExpr

Description

Call LoadParamListItems to convert the list of parameter descriptors that is returned from the GetProcedureParams method of TSQLConnection (see page 1510) to a collection of TParam objects.

Params is an empty TParams collection that the procedure fills with information from the list of parameter descriptors.

ProcParams is the list of parameter descriptors that GetProcedureParams returns.

See Also

FreeProcParams (see page 1620)

TSQLConnection (see page 1510)

24.2.8 SqlExpr.RegisterDbXpressLib Function

Pascal

```
procedure RegisterDbXpressLib(GetClassProc: Pointer);
```

C++

```
RegisterDbXpressLib(void * GetClassProc);
```

File

SqlExpr

Description

This is function SqlExpr.RegisterDbXpressLib.

24.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	EConnectFlag (see page 1624)	This is record SqlExpr.EConnectFlag.
	TConnectionState (see page 1627)	TConnectionState describes the state of a TSQLConnection (see page 1510) component.
	TConnectionUserType (see page 1627)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 1628)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 1629)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TTableScope (see page 1630)	TTableScopes (see page 1640) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransIsolationLevel (see page 1631)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Records

	Name	Description
	SPParamDesc (see page 1625)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.
	SQLTRACEDesc25 (see page 1626)	This is record SqlExpr.SQLTRACEDesc25.
	SQLTRACEDesc30 (see page 1626)	This is record SqlExpr.SQLTRACEDesc30.
	TSQLSchemaInfo (see page 1630)	This is record SqlExpr.TSQLSchemaInfo.
	TTransactionDesc (see page 1631)	This is record SqlExpr.TTransactionDesc.

24.3.1 SqlExpr.EConnectFlag Enumeration

Pascal

```
EConnectFlag = (
  eConnect,
  eReconnect,
  eDisconnect
);
```

C++

```
enum EConnectFlag {
  eConnect,
  eReconnect,
  eDisconnect
};
```

File

SqlExpr

Description

This is record SqlExpr.EConnectFlag.

24.3.2 SqlExpr.SPParamDesc Record

SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.

Pascal

```
SPParamDesc = packed record
  iParamNum: Word;
  szName: WideString;
  iArgType: TParamType;
  iDataType: TFieldType;
  iUnits1: SmallInt;
  iUnits2: SmallInt;
  iLen: LongWord;
end;
```

C++

```
struct SPParamDesc {
  Word iParamNum;
  BSTR szName;
  TParamType iArgType;
  TFieldType iDataType;
  SmallInt iUnits1;
  SmallInt iUnits2;
  LongWord iLen;
};
```

File

SqlExpr

Description

PSPParamDesc (see page 1633) is a pointer to a parameter descriptor. SPParamDesc is a parameter descriptor that describes a parameter for a stored procedure. SPParamDesc includes the following members:

Member	Meaning
iParamNum	The ordinal position of the parameter.
szName	The name of the parameter
iArgType	The type of the parameter (input, output, and so on)
iDataType	The data type of the parameter
iUnits1	The size for the datatype (number of characters, digits, etc)
iUnits2	Number of decimal places if datatype is BCD
iLen	The number of bytes needed to store a parameter value

See Also

TSQLConnection

24.3.3 SqlExpr.SQLTRACEDesc25 Record

Pascal

```
SQLTRACEDesc25 = packed record
  pszTrace: array [0..1023] of AnsiChar;
  eTraceCat: TRACECat;
  ClientData: Integer;
  uTotalMsgLen: Word;
end;
```

C++

```
struct SQLTRACEDesc25 {
  array [0..1023] of AnsiChar pszTrace;
  TRACECat eTraceCat;
  int ClientData;
  Word uTotalMsgLen;
};
```

File

SqlExpr

Description

This is record SqlExpr.SQLTRACEDesc25.

24.3.4 SqlExpr.SQLTRACEDesc30 Record

Pascal

```
SQLTRACEDesc30 = packed record
  pszTrace: array [0..1023] of WideChar;
  eTraceCat: TRACECat;
  ClientData: Integer;
  uTotalMsgLen: Word;
end;
```

C++

```
struct SQLTRACEDesc30 {
  array [0..1023] of WideChar pszTrace;
  TRACECat eTraceCat;
  int ClientData;
  Word uTotalMsgLen;
};
```

File

SqlExpr

Description

This is record SqlExpr.SQLTRACEDesc30.

24.3.5 SqlExpr.TConnectionState Enumeration

TConnectionState describes the state of a TSQLConnection (see page 1510) component.

Pascal

```
TConnectionState = (
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
);
```

C++

```
enum TConnectionState {
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
};
```

File

SqlExpr

Description

TConnectionState can have any of the following values:

Value	Meaning
csStateClosed	The connection is closed. (The Connected property is false)
csStateOpen	The connection is open. (The Connected property is true)
csStateConnecting	The component is establishing a connection but has not completed the process.
csStateExecuting	The component has passed an SQL command to the server and it is currently executing.
csStateFetching	The component is fetching database information from the server.
csStateDisconnecting	The component is terminating a connection but has not completed the process.

24.3.6 SqlExpr.TConnectionUserType Enumeration

Pascal

```
TConnectionUserType = (
  eUserMonitor,
  eUserDataSet
);
```

C++

```
enum TConnectionUserType {
  eUserMonitor,
```

```
  eUserDataSet
};
```

File

SqlExpr

Description

This is record SqlExpr.TConnectionUserType.

24.3.7 SqlExpr.TSchemaType Enumeration

TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.

Pascal

```
TSchemaType = (
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
);
```

C++

```
enum TSchemaType {
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
};
```

File

SqlExpr

Description

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has three members: FType, ObjectName, and Pattern, which are described below.

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has four members: FType, ObjectName, Pattern, and PackageName, which are described below.

FType is of type TSchemaType. It identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. When FType is stNoSchema, the SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.

stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.
stPackages	Information about all of the packages defined on the database server (Oracle only).

ObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When FType is stColumns or stIndexes, ObjectName is the name of a table. When FType is stProcedureParams, ObjectName is the name of a stored procedure. When FType is stNone, stTables, stSysTables, or stProcedures, ObjectName is ignored.

Pattern is an SQL pattern mask that limits the items returned. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. Pattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package that contains the stored procedure about which you are requesting information. This member is used when FType is stProcedures or stProcedureParams. PackageName is ignored if the server is not an Oracle database.

24.3.8 SqlExpr.TSQLExceptionType Enumeration

TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.

Pascal

```
TSQLExceptionType = (
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
);
```

C++

```
enum TSQLExceptionType {
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
};
```

File

SqlExpr

Description

ESQLException objects use TSQLExceptionType to identify the dbExpress object from which an error code originated. By identifying the underlying dbExpress object, ESQLException identifies that object's interface, which it then uses to obtain an error message.

TSQLExceptionType includes the following values:

Value	Description
exceptConnection	Error (see page 3662) arose when establishing a connection to the database server, obtaining an interface for sending a command, fetching server metadata, or controlling a transaction.
exceptCommand	Error (see page 3662) occurred while preparing or executing an SQL command.
exceptCursor	Error (see page 3662) occurred while fetching data from a dbExpress cursor.
exceptMetaData	Error (see page 3662) occurred while trying to fetch metadata (schema information).
exceptUseLast	The error is cached by the TSQLConnection (see page 1510) component.

24.3.9 SqlExpr.TSQLSchemaInfo Record

Pascal

```
TSQLSchemaInfo = record
  FType: TSchemaType;
  ObjectName: WideString;
  Pattern: WideString;
  PackageName: WideString;
end;
```

C++

```
struct TSQLSchemaInfo {
  TSchemaType FType;
  BSTR ObjectName;
  BSTR Pattern;
  BSTR PackageName;
};
```

File

SqlExpr

Description

This is record SqlExpr.TSQLSchemaInfo.

24.3.10 SqlExpr.TTableScope Enumeration

TTableScopes (see page 1640) and TTableScopess indicate the types of tables for which to fetch metadata.

Pascal

```
TTableScope = (
  tsSynonym,
  tsSysTable,
  tsTable,
  tsView
);
```

C++

```
enum TTableScope {
  tsSynonym,
  tsSysTable,
```

```
  tsTable,
  tsView
};
```

File

SqlExpr

Description

TTableScopess is a set that indicates the types of tables for which to fetch metadata. It is a set of TTableScopes (see page 1640) values. The following table describes the possible values:

Value	Meaning
tsSynonym	synonyms
tsSysTable	system tables
tsTable	ordinary tables
tsView	views

24.3.11 SqlExpr.TTransactionDesc Record

Pascal

```
TTransactionDesc = packed record
  TransactionID: LongWord;
  GlobalID: LongWord;
  IsolationLevel: TTransIsolationLevel;
  CustomIsolation: LongWord;
end;
```

C++

```
struct TTransactionDesc {
  LongWord TransactionID;
  LongWord GlobalID;
  TTransIsolationLevel IsolationLevel;
  LongWord CustomIsolation;
};
```

File

SqlExpr

Description

This is record SqlExpr.TTransactionDesc.

24.3.12 SqlExpr.TTransIsolationLevel Enumeration

This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Pascal

```
TTransIsolationLevel = (
  xilREADCOMMITTED,
  xilREPEATABLEREAD,
  xilDIRTYREAD,
```

```
    xilCUSTOM
  );
```

C++

```
  enum TTransIsolationLevel {
    xilREADCOMMITTED,
    xilREPEATABLEREAD,
    xilDIRTYREAD,
    xilCUSTOM
  };
```

File

SqlExpr

Description

The following table describes the possible values.

Element	Description
xilDIRTYREAD	Current transaction sees all changes made by other transactions, even if they have not yet been committed. This option is not available for the Oracle driver.
xilREADCOMMITTED	Current transaction sees only changes that have been committed, but can receive an inconsistent view of the data if additional changes are committed before the transaction ends.
xilREPEATABLEREAD	Current transaction is guaranteed a consistent view of the data, which includes only changes committed by other transactions at the start of the transaction.
xilCUSTOM	Current transaction uses a database-specific isolation level. When the eConnTxnIsoLevel option is xilCUSTOM, the transaction descriptor gives the actual isolation level. This is not currently supported by any drivers, but allows future drivers the ability to expand the number of isolation levels.

24.4 Types

The following table lists types in this documentation.

Types

Name	Description
PSPParamDesc (see page 1633)	This is type SqlExpr.PSPParamDesc.
pSQLTRACEDesc (see page 1634)	This is type SqlExpr.pSQLTRACEDesc.
pSQLTRACEDesc25 (see page 1634)	This is type SqlExpr.pSQLTRACEDesc25.
pSQLTRACEDesc30 (see page 1634)	This is type SqlExpr.pSQLTRACEDesc30.
SQLResult (see page 1635)	SQLResult is the return type for dbExpress function calls.
SQLSPParamDesc (see page 1636)	This is type SqlExpr.SQLSPParamDesc.
SQLTRACEDesc (see page 1636)	This is type SqlExpr.SQLTRACEDesc.
TConnectChangeEvent (see page 1636)	This is type SqlExpr.TConnectChangeEvent.
TFieldDescList (see page 1637)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 1637)	This is type SqlExpr.TFieldList.
TFLDDescRef (see page 1637)	This is type SqlExpr.TFLDDescRef.
TGetDriverFunc (see page 1638)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 1638)	This is type SqlExpr.TLocale.
TLocaleCode (see page 1638)	This is type SqlExpr.TLocaleCode.
TParseInsertSqlEvent (see page 1639)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 1639)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 1639)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 1640)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 1640)	This is type SqlExpr.TTraceEvent.
TTraceEvent25 (see page 1640)	This is type SqlExpr.TTraceEvent25.
TTraceEvent30 (see page 1641)	This is type SqlExpr.TTraceEvent30.
TTraceLogEvent (see page 1641)	This is type SqlExpr.TTraceLogEvent.
TTraceLogEvent25 (see page 1641)	This is type SqlExpr.TTraceLogEvent25.
TTraceLogEvent30 (see page 1642)	This is type SqlExpr.TTraceLogEvent30.

24.4.1 SqlExpr.PSPParamDesc Type

Pascal

```
PSPParamDesc = ^SPParamDesc;
```

C++

```
^SPParamDesc PSPParamDesc;
```

File

SqlExpr

Description

This is type SqlExpr.PSPParamDesc.

24.4.2 SqlExpr.pSQLTRACEDesc Type

Pascal

```
pSQLTRACEDesc = pSQLTRACEDesc30;
```

C++

```
pSQLTRACEDesc30 pSQLTRACEDesc;
```

File

SqlExpr

Description

This is type SqlExpr.pSQLTRACEDesc.

24.4.3 SqlExpr.pSQLTRACEDesc25 Type

Pascal

```
pSQLTRACEDesc25 = ^SQLTRACEDesc25;
```

C++

```
^SQLTRACEDesc25 pSQLTRACEDesc25;
```

File

SqlExpr

Description

This is type SqlExpr.pSQLTRACEDesc25.

24.4.4 SqlExpr.pSQLTRACEDesc30 Type

Pascal

```
pSQLTRACEDesc30 = ^SQLTRACEDesc30;
```

C++

```
^SQLTRACEDesc30 pSQLTRACEDesc30;
```

File

SqlExpr

Description

This is type SqlExpr.pSQLTRACEDesc30.

24.4.5 SqlExpr.SQLResult Type

SQLResult is the return type for dbExpress function calls.

Pascal

```
SQLResult = TDBXErrorCode;
```

C++

```
TDBXErrorCode SQLResult;
```

File

SqlExpr

Description

SQLResult indicates the success or failure of a dbExpress function. The following table lists the error codes defined for dbExpress.

SQLResult Value	Meaning
SQL_SUCCESS	The function call was successful.
DBXERR_NONE	Same as SQL_SUCCESS.
DBXERR_NOMEMORY	Insufficient memory.
DBXERR_INVALIDFLDTYPE	Invalid field type.
DBXERR_INVALIDHNDL	Invalid handle.
DBXERR_INVALIDTIME	Invalid time value.
DBXERR_NOTSUPPORTED	Feature not supported.
DBXERR_INVALIDXLATION	Invalid translation.
DBXERR_INVALIDPARAM	Invalid parameter.
DBXERR_OUTOFRANGE	Value out of range.
DBXERR_SQLPARAMNOTSET	Parameter not set.
DBXERR_EOF	Unexpected end-of-file.
DBXERR_INVALIDUSRPASS	Invalid user password.
DBXERR_INVALIDPRECISION	Invalid precision on a BCD value.
DBXERR_INVALIDLEN	Invalid length.
DBXERR_INVALIDTXNISOLEVEL	Invalid transaction isolation level.
DBXERR_INVALIDTXNID	Invalid transaction id.
DBXERR_DUPLICATETXNID	Duplicate transaction id.
DBXERR_DRIVERRESTRICTED	Driver restricted.
DBXERR_LOCALTRANSACTIVE	An attempt to start multiple local transactions (only 1 allowed).

24.4.6 SqlExpr.SQLSPParamDesc Type

Pascal

```
SQLSPParamDesc = SPParamDesc;
```

C++

```
SPParamDesc SQLSPParamDesc;
```

File

SqlExpr

Description

This is type SqlExpr.SQLSPParamDesc.

24.4.7 SqlExpr.SQLTRACEDesc Type

Pascal

```
SQLTRACEDesc = SQLTRACEDesc30;
```

C++

```
SQLTRACEDesc30 SQLTRACEDesc;
```

File

SqlExpr

Description

This is type SqlExpr.SQLTRACEDesc.

24.4.8 SqlExpr.TConnectChangeEvent Type

Pascal

```
TConnectChangeEvent = procedure (Sender: TObject; Connecting: Boolean) of object;
```

C++

```
(Sender: TObject; Connecting: Boolean) ( TConnectChangeEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TConnectChangeEvent.

24.4.9 SqlExpr.TFieldDescList Type

Pascal

```
TFieldDescList = array of TFLDDesc;
```

C++

```
array of TFLDDesc TFieldDescList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldDescList.

24.4.10 SqlExpr.TFieldList Type

Pascal

```
TFieldList = TList;
```

C++

```
TList TFieldList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldList.

24.4.11 SqlExpr.TFLDDescRef Type

Pascal

```
TFLDDescRef = class of TFLDDesc;
```

C++

```
TFLDDesc TFLDDescRef;
```

File

SqlExpr

Description

This is type SqlExpr.TFLDDescRef.

24.4.12 SqlExpr.TGetDriverFunc Type

Pascal

```
TGetDriverFunc = function (SVendorLib, SResourceFile: PChar; out Obj): SQLResult;
```

C++

```
(SVendorLib, SResourceFile: PChar; out Obj): SQLResult ( TGetDriverFunc)();
```

File

SqlExpr

Description

This is type SqlExpr.TGetDriverFunc.

24.4.13 SqlExpr.TLocale Type

Pascal

```
TLocale = Pointer;
```

C++

```
void * TLocale;
```

File

SqlExpr

Description

This is type SqlExpr.TLocale.

24.4.14 SqlExpr.TLocaleCode Type

Pascal

```
TLocaleCode = Integer;
```

C++

```
int TLocaleCode;
```

File

SqlExpr

Description

This is type SqlExpr.TLocaleCode.

24.4.15 SqlExpr.TParseInsertSqlEvent Type

Pascal

```
TParseInsertSqlEvent = procedure (var FieldNames: TWideStrings; SQL: WideString; var
BindAllFields: Boolean; var TableName: WideString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: WideString; var BindAllFields: Boolean; var TableName:
WideString) ( TParseInsertSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseInsertSqlEvent.

24.4.16 SqlExpr.TParseSqlEvent Type

Pascal

```
TParseSqlEvent = procedure (var FieldNames: TWideStrings; SQL: WideString; var TableName:
WideString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: WideString; var TableName: WideString) ( TParseSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseSqlEvent.

24.4.17 SqlExpr.TSQLConnectionLoginEvent Type

Pascal

```
TSQLConnectionLoginEvent = procedure (Database: TSQLConnection; LoginParams: TWideStrings) of
object;
```

C++

```
(Database: TSQLConnection; LoginParams: TWideStrings) ( TSQLConnectionLoginEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TSQLConnectionLoginEvent.

24.4.18 SqlExpr.TTableScopes Type

Pascal

```
TTableScopes = set of TTableScope;
```

C++

```
TTableScope TTableScopes;
```

File

SqlExpr

Description

This is type SqlExpr.TTableScopes.

24.4.19 SqlExpr.TTraceEvent Type

Pascal

```
TTraceEvent = TTraceEvent30;
```

C++

```
TTraceEvent30 TTraceEvent;
```

File

SqlExpr

Description

This is type SqlExpr.TTraceEvent.

24.4.20 SqlExpr.TTraceEvent25 Type

Pascal

```
TTraceEvent25 = procedure (Sender: TObject; CBInfo: pSQLTRACEDesc25; var LogTrace: Boolean) of
object;
```

C++

```
(Sender: TObject; CBInfo: pSQLTRACEDesc25; var LogTrace: Boolean) ( TTraceEvent25)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceEvent25.

24.4.21 SqlExpr.TTraceEvent30 Type

Pascal

```
TTraceEvent30 = procedure (Sender: TObject; CBInfo: pSQLTRACEDesc30; var LogTrace: Boolean) of object;
```

C++

```
(Sender: TObject; CBInfo: pSQLTRACEDesc30; var LogTrace: Boolean) ( TTraceEvent30)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceEvent30.

24.4.22 SqlExpr.TTraceLogEvent Type

Pascal

```
TTraceLogEvent = TTraceLogEvent30;
```

C++

```
TTraceLogEvent30 TTraceLogEvent;
```

File

SqlExpr

Description

This is type SqlExpr.TTraceLogEvent.

24.4.23 SqlExpr.TTraceLogEvent25 Type

Pascal

```
TTraceLogEvent25 = procedure (Sender: TObject; CBInfo: pSQLTRACEDesc25) of object;
```

C++

```
(Sender: TObject; CBInfo: pSQLTRACEDesc25) ( TTraceLogEvent25)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceLogEvent25.

24.4.24 SqlExpr.TTraceLogEvent30 Type

Pascal

```
TTraceLogEvent30 = procedure (Sender: TObject; CBInfo: pSQLTRACEDesc30) of object;
```

C++

```
(Sender: TObject; CBInfo: pSQLTRACEDesc30) ( TTraceLogEvent30)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceLogEvent30.

24.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DllGetClassObject (see page 1643)	This is variable SqlExpr.DllGetClassObject.

24.5.1 SqlExpr.DllGetClassObject Variable

Pascal

```
DllGetClassObject: function(const CLSID, IID: TGUID; var Obj): HResult;
```

C++

```
function(const CLSID, IID: TGUID; var Obj): HResult DllGetClassObject;
```

File

SqlExpr

Description

This is variable SqlExpr.DllGetClassObject.

24.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BlobTypeMap (see page 1644)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 1645)	This is constant SqlExpr.DataTypeMap.
DefaultCursor (see page 1645)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 1646)	This is constant SqlExpr.DefaultMaxBlobSize.
DefaultRowsetSize (see page 1646)	This is constant SqlExpr.DefaultRowsetSize.
eSQLAutoIncr (see page 1646)	This is constant SqlExpr.eSQLAutoIncr.
eSQLDefault (see page 1647)	This is constant SqlExpr.eSQLDefault.
eSQLFunction (see page 1647)	This is constant SqlExpr.eSQLFunction.
eSQLLocal (see page 1647)	This is constant SqlExpr.eSQLLocal.
eSQLNonUnique (see page 1648)	This is constant SqlExpr.eSQLNonUnique.
eSQLPackage (see page 1648)	This is constant SqlExpr.eSQLPackage.
eSQLPrimaryKey (see page 1648)	This is constant SqlExpr.eSQLPrimaryKey.
eSQLProcedure (see page 1649)	This is constant SqlExpr.eSQLProcedure.
eSQLRowId (see page 1649)	This is constant SqlExpr.eSQLRowId.
eSQLRowVersion (see page 1649)	This is constant SqlExpr.eSQLRowVersion.
eSQLSynonym (see page 1650)	This is constant SqlExpr.eSQLSynonym.
eSQLSysProcedure (see page 1650)	This is constant SqlExpr.eSQLSysProcedure.
eSQLSystemTable (see page 1650)	This is constant SqlExpr.eSQLSystemTable.
eSQLTable (see page 1651)	This is constant SqlExpr.eSQLTable.
eSQLTempTable (see page 1651)	This is constant SqlExpr.eSQLTempTable.
eSQLUnique (see page 1651)	This is constant SqlExpr.eSQLUnique.
eSQLView (see page 1652)	This is constant SqlExpr.eSQLView.
FldSubTypeMap (see page 1652)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 1652)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 1653)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 1653)	This is constant SqlExpr.SAnd.
SOrderBy (see page 1653)	This is constant SqlExpr.SOrderBy.
SParam (see page 1654)	This is constant SqlExpr.SParam.
SSelect (see page 1654)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 1654)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 1655)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 1655)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 1655)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 1656)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 1656)	This is constant SqlExpr.TErrorMessageSize.

24.6.1 SqlExpr.BlobTypeMap Constant

Pascal

```
BlobTypeMap: array[fldstMEMO..fldstBFILE] of TFieldType = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftWideMemo,
ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

C++

```
array[fldstMEMO..fldstBFILE] of TFieldType BlobTypeMap = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftWideMemo,
ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

File

SqlExpr

Description

This is constant SqlExpr.BlobTypeMap.

24.6.2 SqlExpr.DataTypeMap Constant

Pascal

```
DataTypeMap: array[0..MAXLOGFLDTYPES - 1] of TFieldType = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt, ftLargeInt, ftADT, ftArray,
ftReference, ftDataSet, ftTimeStamp, ftFMTBCD, ftWideString);
```

C++

```
array[0..MAXLOGFLDTYPES - 1] of TFieldType DataTypeMap = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt, ftLargeInt, ftADT, ftArray,
ftReference, ftDataSet, ftTimeStamp, ftFMTBCD, ftWideString);
```

File

SqlExpr

Description

This is constant SqlExpr.DataTypeMap.

24.6.3 SqlExpr.DefaultCursor Constant

Pascal

```
DefaultCursor = 0;
```

C++

```
DefaultCursor = 0;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultCursor.

24.6.4 SqlExpr.DefaultMaxBlobSize Constant

Pascal

```
DefaultMaxBlobSize = -1;
```

C++

```
DefaultMaxBlobSize = -1;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultMaxBlobSize.

24.6.5 SqlExpr.DefaultRowsetSize Constant

Pascal

```
DefaultRowsetSize = 20;
```

C++

```
DefaultRowsetSize = 20;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultRowsetSize.

24.6.6 SqlExpr.eSQLAutoIncr Constant

Pascal

```
eSQLAutoIncr = $0004;
```

C++

```
eSQLAutoIncr = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLAutoIncr.

24.6.7 SqlExpr.eSQLDefault Constant

Pascal

```
eSQLDefault = $0008;
```

C++

```
eSQLDefault = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLDefault.

24.6.8 SqlExpr.eSQLFunction Constant

Pascal

```
eSQLFunction = $0002;
```

C++

```
eSQLFunction = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLFunction.

24.6.9 SqlExpr.eSQLLocal Constant

Pascal

```
eSQLLocal = $0020;
```

C++

```
eSQLLocal = 0x0020;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLLocal.

24.6.10 SqlExpr.eSQLNonUnique Constant

Pascal

```
eSQLNonUnique = $0001;
```

C++

```
eSQLNonUnique = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLNonUnique.

24.6.11 SqlExpr.eSQLPackage Constant

Pascal

```
eSQLPackage = $0004;
```

C++

```
eSQLPackage = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLPackage.

24.6.12 SqlExpr.eSQLPrimaryKey Constant

Pascal

```
eSQLPrimaryKey = $0004;
```

C++

```
eSQLPrimaryKey = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLPrimaryKey.

24.6.13 SqlExpr.eSQLProcedure Constant

Pascal

```
eSQLProcedure = $0001;
```

C++

```
eSQLProcedure = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLProcedure.

24.6.14 SqlExpr.eSQLRowId Constant

Pascal

```
eSQLRowId = $0001;
```

C++

```
eSQLRowId = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLRowId.

24.6.15 SqlExpr.eSQLRowVersion Constant

Pascal

```
eSQLRowVersion = $0002;
```

C++

```
eSQLRowVersion = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLRowVersion.

24.6.16 SqlExpr.eSQLSynonym Constant

Pascal

```
eSQLSynonym = $0008;
```

C++

```
eSQLSynonym = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSynonym.

24.6.17 SqlExpr.eSQLSysProcedure Constant

Pascal

```
eSQLSysProcedure = $0008;
```

C++

```
eSQLSysProcedure = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSysProcedure.

24.6.18 SqlExpr.eSQLSystemTable Constant

Pascal

```
eSQLSystemTable = $0004;
```

C++

```
eSQLSystemTable = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSystemTable.

24.6.19 SqlExpr.eSQLTable Constant

Pascal

```
eSQLTable = $0001;
```

C++

```
eSQLTable = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLTable.

24.6.20 SqlExpr.eSQLTempTable Constant

Pascal

```
eSQLTempTable = $0010;
```

C++

```
eSQLTempTable = 0x0010;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLTempTable.

24.6.21 SqlExpr.eSQLUnique Constant

Pascal

```
eSQLUnique = $0002;
```

C++

```
eSQLUnique = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLUnique.

24.6.22 SqlExpr.eSQLView Constant

Pascal

```
eSQLView = $0002;
```

C++

```
eSQLView = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLView.

24.6.23 SqlExpr.FldSubTypeMap Constant

Pascal

```
FldSubTypeMap: array[TFieldType] of Word = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0,
0, fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ,
fldstDBSOLEOBJ, fldstTYPEDBINARY, 0, fldstFIXED, 0, 0, 0, 0, 0, 0, fldstHBINARY, fldstHMEMO,
0, 0, 0, 0, 0, 0, fldstFIXED, fldstWIDEMEMO, fldstORATIMESTAMP, fldstORAINTERVAL);
```

C++

```
array[TFieldType] of Word FldSubTypeMap = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0, 0,
fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ, fldstDBSOLEOBJ,
fldstTYPEDBINARY, 0, fldstFIXED, 0, 0, 0, 0, 0, 0, fldstHBINARY, fldstHMEMO, 0, 0, 0, 0, 0, 0,
fldstFIXED, fldstWIDEMEMO, fldstORATIMESTAMP, fldstORAINTERVAL);
```

File

SqlExpr

Description

This is constant SqlExpr.FldSubTypeMap.

24.6.24 SqlExpr.FldTypeMap Constant

Pascal

```
FldTypeMap: TFieldMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
fldWIDESTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldBCD, fldWIDESTRING, fldBLOB, fldDATETIME,
fldZSTRING);
```

C++

```
TFieldMap FldTypeMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
```

```
fldWIDESTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldBCD, fldWIDESTRING, fldBLOB, fldDATETIME,
fldZSTRING);
```

File

SqlExpr

Description

This is constant SqlExpr.FldTypeMap.

24.6.25 SqlExpr.HourGlassCursor Constant

Pascal

```
HourGlassCursor = -11;
```

C++

```
HourGlassCursor = -11;
```

File

SqlExpr

Description

This is constant SqlExpr.HourGlassCursor.

24.6.26 SqlExpr.SAnd Constant

Pascal

```
SAnd = ' and ';
```

C++

```
SAnd = ' and ';
```

File

SqlExpr

Description

This is constant SqlExpr.SAnd.

24.6.27 SqlExpr.SOrderBy Constant

Pascal

```
SOrderBy = ' order by ';
```

C++

```
SOrderBy = ' order by ';
```

File

SqlExpr

Description

This is constant SqlExpr.SOrderBy.

24.6.28 SqlExpr.SParam Constant

Pascal

```
SParam = '?';
```

C++

```
SParam = '?';
```

File

SqlExpr

Description

This is constant SqlExpr.SParam.

24.6.29 SqlExpr.SSelect Constant

Pascal

```
SSelect = 'select';
```

C++

```
SSelect = 'select';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelect.

24.6.30 SqlExpr.SSelectSpaces Constant

Pascal

```
SSelectSpaces = ' select ';
```

C++

```
SSelectSpaces = ' select ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectSpaces.

24.6.31 SqlExpr.SSelectStar Constant

Pascal

```
SSelectStar = ' select * ';
```

C++

```
SSelectStar = ' select * ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStar.

24.6.32 SqlExpr.SSelectStarFrom Constant

Pascal

```
SSelectStarFrom = ' select * from ';
```

C++

```
SSelectStarFrom = ' select * from ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStarFrom.

24.6.33 SqlExpr.SUB_TYPE_MEMO Constant

Pascal

```
SUB_TYPE_MEMO = TDBXDataTypes.MemoSubType;
```

C++

```
SUB_TYPE_MEMO = TDBXDataTypes.MemoSubType;
```

File

SqlExpr

Description

This is constant SqlExpr.SUB_TYPE_MEMO.

24.6.34 SqlExpr.SWhere Constant

Pascal

```
SWhere = ' where ';
```

C++

```
SWhere = ' where ';
```

File

SqlExpr

Description

This is constant SqlExpr.SWhere.

24.6.35 SqlExpr.TErrorMessageSize Constant

Pascal

```
TErrorMessageSize = 2048;
```

C++

```
TErrorMessageSize = 2048;
```

File

SqlExpr

Description

This is constant SqlExpr.TErrorMessageSize.

25 SqlTimSt Namespace

This is namespace SqlTimSt.

Topics

Name	Description
Classes (see page 1659)	The following table lists classes in this documentation.
Functions (see page 1661)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1669)	The following table lists structs, records, enums in this documentation.
Types (see page 1671)	The following table lists types in this documentation.
Constants (see page 1672)	The following table lists constants in this documentation.

Classes

	Name	Description
	TTimeZone (see page 1659)	This is class SqlTimSt.TTimeZone.

Constants

Name	Description
NullSQLTimeStamp (see page 1672)	This is constant SqlTimSt.NullSQLTimeStamp.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 1661)	Checks whether a TSQLTimeStamp (see page 1669) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 1662)	Converts a TDateTime (see page 3335) value to a TSQLTimeStamp (see page 1669) value.
	LocalToUTC (see page 1662)	This is function SqlTimSt.LocalToUTC.
	SQLDayOfWeek (see page 1662)	Indicates the day of the week when a specified TSQLTimeStamp (see page 1669) value occurs.
	SQLTimeStampToDateTime (see page 1663)	Converts a TSQLTimeStamp (see page 1669) value to a TDateTime (see page 3335) value.
	SQLTimeStampToStr (see page 1663)	Converts a TSQLTimeStamp (see page 1669) value to a string.
	StrToSQLTimeStamp (see page 1664)	Converts a string to a TSQLTimeStamp (see page 1669) value.
	TryStrToSQLTimeStamp (see page 1665)	Converts a string to a TSQLTimeStamp (see page 1669) value.
	UTCToLocal (see page 1665)	This is function SqlTimSt.UTCToLocal.
	VarIsSQLTimeStamp (see page 1666)	Indicates whether a specified Variant (see page 3430) represents a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStamp (see page 1666)	Returns the TVarType constant that identifies the internal type of a Variant (see page 3430) as a TSQLTimeStamp (see page 1669) value.

	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarToSQLTimeStamp (see page 1668)	Converts a Variant (see page 3430) that represents a TSQLTimeStamp (see page 1669) value to the corresponding TSQLTimeStamp (see page 1669).

Structs, Records, Enums

	Name	Description
	TSQLTimeStamp (see page 1669)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

Types

Name	Description
PSQLTimeStamp (see page 1671)	This is type SqlTimSt.PSQLTimeStamp.

25.1 Classes

The following table lists classes in this documentation.

Records

	Name	Description
	TTimeZone (see page 1659)	This is class SqlTimSt.TTimeZone.

25.1.1 TTimeZone Record

Pascal

```
TTimeZone = record
  FUsesDayligtSavings: Boolean;
  FInfo: TTimeZoneInformation;
end;
```

C++

```
struct TTimeZone {
  Boolean FUsesDayligtSavings;
  TTimeZoneInformation FInfo;
};
```

File

SqlTimSt

Description

This is class SqlTimSt.TTimeZone.

25.1.1.1 TTimeZone Members

The following tables list the members exposed by TTimeZone.

Public Fields

	Name	Description
	FInfo (see page 1660)	This is FInfo, a member of class TTimeZone.
	FUsesDayligtSavings (see page 1660)	This is FUsesDayligtSavings, a member of class TTimeZone.

Public Methods

	Name	Description
	GetTimeZone (see page 1660)	This is GetTimeZone, a member of class TTimeZone.

25.1.1.2 TTimeZone Fields

The fields of the TTimeZone class are listed here.

Public Fields

	Name	Description
	FInfo (see page 1660)	This is FInfo, a member of class TTimeZone.
	FUsesDayligtSavings (see page 1660)	This is FUsesDayligtSavings, a member of class TTimeZone.

25.1.1.2.1 TTimeZone.FInfo Field

Pascal

```
FInfo: TTimeZoneInformation;
```

C++

```
TTimeZoneInformation FInfo;
```

Description

This is FInfo, a member of class TTimeZone.

25.1.1.2.2 TTimeZone.FUsesDayligtSavings Field

Pascal

```
FUsesDayligtSavings: Boolean;
```

C++

```
Boolean FUsesDayligtSavings;
```

Description

This is FUsesDayligtSavings, a member of class TTimeZone.

25.1.1.3 TTimeZone Methods

The methods of the TTimeZone class are listed here.

Public Methods

	Name	Description
	GetTimeZone (see page 1660)	This is GetTimeZone, a member of class TTimeZone.

25.1.1.3.1 TTimeZone.GetTimeZone Method

Pascal

```
class function GetTimeZone: TTimeZone; static;
```

C++

```
static __fastcall TTimeZone GetTimeZone();
```

Description

This is GetTimeZone, a member of class TTimeZone.

25.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 1661)	Checks whether a TSQLTimeStamp (see page 1669) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 1662)	Converts a TDateTime (see page 3335) value to a TSQLTimeStamp (see page 1669) value.
	LocalToUTC (see page 1662)	This is function SqlTimSt.LocalToUTC.
	SQLDayOfWeek (see page 1662)	Indicates the day of the week when a specified TSQLTimeStamp (see page 1669) value occurs.
	SQLTimeStampToDateTime (see page 1663)	Converts a TSQLTimeStamp (see page 1669) value to a TDateTime (see page 3335) value.
	SQLTimeStampToStr (see page 1663)	Converts a TSQLTimeStamp (see page 1669) value to a string.
	StrToSQLTimeStamp (see page 1664)	Converts a string to a TSQLTimeStamp (see page 1669) value.
	TryStrToSQLTimeStamp (see page 1665)	Converts a string to a TSQLTimeStamp (see page 1669) value.
	UTCToLocal (see page 1665)	This is function SqlTimSt.UTCToLocal.
	VarIsSQLTimeStamp (see page 1666)	Indicates whether a specified Variant (see page 3430) represents a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStamp (see page 1666)	Returns the TVarType constant that identifies the internal type of a Variant (see page 3430) as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarSQLTimeStampCreate (see page 1667)	Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.
	VarToSQLTimeStamp (see page 1668)	Converts a Variant (see page 3430) that represents a TSQLTimeStamp (see page 1669) value to the corresponding TSQLTimeStamp (see page 1669).

25.2.1 SqlTimSt.CheckSqlTimeStamp Function

Checks whether a TSQLTimeStamp (see page 1669) value represents a valid date and time.

Pascal

```
procedure CheckSqlTimeStamp(const ASQLTimeStamp: TSQLTimeStamp);
```

C++

```
CheckSqlTimeStamp(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

CheckSqlTimeStamp checks that the specified TSQLTimeStamp (see page 1669) value represents a valid date and time. If the fields of the ASQLTimeStamp parameter are out of the valid range, CheckSqlTimeStamp raises an EConversionError

exception.

See Also

EConvertError

25.2.2 SqlTimSt.DateTimeToSQLTimeStamp Function

Converts a TDateTime (see page 3335) value to a TSQLTimeStamp (see page 1669) value.

Pascal

```
function DateTimeToSQLTimeStamp(const DateTime: TDateTime): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp DateTimeToSQLTimeStamp(const TDateTime DateTime);
```

File

SqlTimSt

Description

Use DateTimeToSQLTimeStamp to convert the TDateTime (see page 3335) value specified by DateTime to a TSQLTimeStamp (see page 1669) value.

See Also

SQLTimeStampToDateTime (see page 1663)

SQLTimeStampToStr (see page 1663)

StrToSQLTimeStamp (see page 1664)

TSQLTimeStampField

25.2.3 SqlTimSt.LocalToUTC Function

Pascal

```
function LocalToUTC(var TZInfo: TTimeZone; var Value: TSQLTimeStamp): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp LocalToUTC(TTimeZone TZInfo, TSQLTimeStamp Value);
```

File

SqlTimSt

Description

This is function SqlTimSt.LocalToUTC.

25.2.4 SqlTimSt.SQLDayOfWeek Function

Indicates the day of the week when a specified TSQLTimeStamp (see page 1669) value occurs.

Pascal

```
function SQLDayOfWeek(const DateTime: TSQLTimeStamp): integer;
```

C++

```
int SQLDayOfWeek(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

SQLDayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

See Also

DayOfWeek

25.2.5 SqlTimSt.SQLTimeStampToDateTime Function

Converts a TSQLTimeStamp (see page 1669) value to a TDateTime (see page 3335) value.

Pascal

```
function SQLTimeStampToDateTime(const DateTime: TSQLTimeStamp): TDateTime;
```

C++

```
TDateTime SQLTimeStampToDateTime(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToDateTime to convert the TSQLTimeStamp (see page 1669) value specified by DateTime to a TDateTime (see page 3335) value.

See Also

DateTimeToSQLTimeStamp (see page 1662)

SQLTimeStampToStr (see page 1663)

StrToSQLTimeStamp (see page 1664)

TSQLTimeStampField

25.2.6 SqlTimSt.SQLTimeStampToStr Function

Converts a TSQLTimeStamp (see page 1669) value to a string.

Pascal

```
function SQLTimeStampToStr(const Format: string; DateTime: TSQLTimeStamp): string;
```

C++

```
UnicodeString SQLTimeStampToStr(const UnicodeString Format, TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToStr to convert a TSQLTimeStamp (see page 1669) value to a string an AnsiString.

DateTimeToStr converts the TSQLTimeStamp (see page 1669) value given by DateTime using the format given by Format and returns the resulting string.

If the string specified by the Format parameter is empty, the TSQLTimeStamp (see page 1669) value is formatted using the ShortDateFormat global variable followed by the LongTimeFormat global variable. The time is not displayed if all time fields in the DateTime parameter are 0.

To create a format string for the Format parameter, use the standard date-time format specifiers:

See Also

DateTimeToSQLTimeStamp (see page 1662)

SQLTimeStampToDateTime (see page 1663)

StrToSQLTimeStamp (see page 1664)

TSQLTimeStampField

CurrencyAndDatetimeFormatting

TryStrToSQLTimeStamp (see page 1665)

25.2.7 SqlTimSt.StrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 1669) value.

Pascal

```
function StrToSQLTimeStamp(const S: string): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp StrToSQLTimeStamp(const UnicodeString S);
```

File

SqlTimSt

Description

Use StrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 1669) value. If S does not represent a valid date-and-time value, StrToSQLTimeStamp raises an EConvertError exception.

Tip: To convert strings to TSQLTimeStamp (see page 1669) values without raising an exception when the string is not a valid date-and-time, use TryStrToSQLTimeStamp (see page 1665) instead.

See Also

DateTimeToSQLTimeStamp (see page 1662)

SQLTimeStampToStr (see page 1663)

SQLTimeStampToDateTime (see page 1663)

TSQLTimeStampField

EConvertError

TryStrToSQLTimeStamp (see page 1665)

25.2.8 SqlTimSt.TryStrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 1669) value.

Pascal

```
function TryStrToSQLTimeStamp(const S: string; var TimeStamp: TSQLTimeStamp): Boolean;
```

C++

```
Boolean TryStrToSQLTimeStamp(const UnicodeString S, TSQLTimeStamp TimeStamp);
```

File

SqlTimSt

Description

Use TryStrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 1669) value.

If S can be converted to a TSQLTimeStamp (see page 1669) value, the result is returned as the TimeStamp parameter and TryStrToSQLTimeStamp returns true.

If S can't be converted to a TSQLTimeStamp (see page 1669) value, TimeStamp is set to NullSQLTimeStamp (see page 1672) and TryStrToSQLTimeStamp returns false.

See Also

DateTimeToSQLTimeStamp (see page 1662)

SQLTimeStampToStr (see page 1663)

SQLTimeStampToDateTime (see page 1663)

NullSQLTimeStamp (see page 1672)

StrToSQLTimeStamp (see page 1664)

25.2.9 SqlTimSt.UTCToLocal Function

Pascal

```
function UTCToLocal(var TZInfo: TTimeZone; var Value: TSQLTimeStamp): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp UTCToLocal(TTimeZone TZInfo, TSQLTimeStamp Value);
```

File

SqlTimSt

Description

This is function SqlTimSt.UTCToLocal.

25.2.10 SqlTimSt.VarIsSQLTimeStamp Function

Indicates whether a specified Variant (see page 3430) represents a TSQLTimeStamp (see page 1669) value.

Pascal

```
function VarIsSQLTimeStamp(const aValue: Variant): Boolean; overload;
```

C++

```
Boolean VarIsSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

VarIsSQLTimeStamp returns true if the Variant (see page 3430) aValue stores its value internally as a TSQLTimeStamp (see page 1669) value. VarIsSQLTimeStamp returns false otherwise.

See Also

VarToSQLTimeStamp (see page 1668)

TSQLTimeStamp (see page 1669)

25.2.11 SqlTimSt.VarSQLTimeStamp Function

Returns the TVarType constant that identifies the internal type of a Variant (see page 3430) as a TSQLTimeStamp (see page 1669) value.

Pascal

```
function VarSQLTimeStamp: TVarType;
```

C++

```
TVarType VarSQLTimeStamp();
```

File

SqlTimSt

Description

VarSQLTimeStamp returns the TVarType constant that identifies a TSQLTimeStamp (see page 1669)-valued Variant (see page 3430). This is the value that the VarType (see page 5101) function returns when the Variant (see page 3430) stores its value as a TSQLTimeStamp (see page 1669) value.

The Variant (see page 3430) type codes for custom Variants (see page 4997) are dynamically allocated when the application loads the unit that implements them. VarSQLTimeStamp allows the application to identify the Variant (see page 3430) type code that is currently used for custom Variants (see page 4997) that represent an SQL time stamp. This value, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3430) types.

See Also

VarIsSQLTimeStamp (see page 1666)

VarType

TSQLTimeStamp (see page 1669)

VarToSQLTimeStamp (see page 1668)

25.2.12 SqlTimSt.VarSQLTimeStampCreate Function

Creates a new Variant (see page 3430) that stores its value internally as a TSQLTimeStamp (see page 1669) value.

Pascal

```
procedure VarSQLTimeStampCreate(var aDest: Variant; const ASQLTimeStamp: TSQLTimeStamp);
overload;
function VarSQLTimeStampCreate: Variant; overload;
function VarSQLTimeStampCreate(const AValue: string): Variant; overload;
function VarSQLTimeStampCreate(const AValue: TDateTime): Variant; overload;
function VarSQLTimeStampCreate(const ASQLTimeStamp: TSQLTimeStamp): Variant; overload;
```

C++

```
VarSQLTimeStampCreate(Variant aDest, const TSQLTimeStamp ASQLTimeStamp);
Variant VarSQLTimeStampCreate();
Variant VarSQLTimeStampCreate(const UnicodeString AValue);
Variant VarSQLTimeStampCreate(const TDateTime AValue);
Variant VarSQLTimeStampCreate(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

Call VarSQLTimeStampCreate to obtain a Variant (see page 3430) that stores its value as a TSQLTimeStamp (see page 1669) value.

Using the first syntax, the Variant (see page 3430) returned as aDest has the same value as ASQLTimeStamp.

Using the second syntax (passing no arguments), the returned Variant (see page 3430) has a value of NullSQLTimeStamp (see page 1672).

Using the third syntax, the returned Variant (see page 3430) has the same value as ASQLTimeStamp.

Using the fourth syntax, the returned Variant (see page 3430) represents the same date and time as the TDateTime (see page 3335) value AValue.

See Also

VarIsSQLTimeStamp (see page 1666)

VarToSQLTimeStamp (see page 1668)

NullSQLTimeStamp (see page 1672)

25.2.13 SqlTimSt.VarToSQLTimeStamp Function

Converts a Variant (see page 3430) that represents a TSQLTimeStamp (see page 1669) value to the corresponding TSQLTimeStamp (see page 1669).

Pascal

```
function VarToSQLTimeStamp(const aValue: Variant): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp VarToSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

Use VarToSQLTimeStamp to obtain a TSQLTimeStamp (see page 1669) from a variant whose value represents a date-and-time value. If the Variant (see page 3430) is a Null (see page 5075) Variant (see page 3430), VarToSQLTimeStamp returns NullSQLTimeStamp (see page 1672).

See Also

VarIsSQLTimeStamp (see page 1666)

TSQLTimeStamp (see page 1669)

NullSQLTimeStamp (see page 1672)

25.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TSQLTimeStamp (see page 1669)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

25.3.1 SqlTimSt.TSQLTimeStamp Record

TSQLTimeStamp represents time and date values with a high degree of accuracy.

Pascal

```
TSQLTimeStamp = packed record
  Year: SmallInt;
  Month: Word;
  Day: Word;
  Hour: Word;
  Minute: Word;
  Second: Word;
  Fractions: LongWord;
end;
```

C++

```
struct TSQLTimeStamp {
  SmallInt Year;
  Word Month;
  Word Day;
  Word Hour;
  Word Minute;
  Word Second;
  LongWord Fractions;
};
```

File

SqlTimSt

Description

The dbExpress database drivers use TSQLTimeStamp when working with date and time information. Year specifies a year from 1 to 9999. Month specifies a month from 1 to 12. Day specifies a day of the month, from 1 through 28, 29, 30, or 31, depending on the Month value. Hour specifies the hour from 0 to 23. Minute and Second can range from 0 to 59. Fractions specifies milliseconds from 0 to 999.

You can create a Variant (see page 3430) that represents a TSQLTimeStamp value using the VarSQLTimeStampCreate (see page 1667) function. In fact, the easiest way to manipulate TSQLTimeStamp values is to create such a Variant (see page 3430) and use the built-in operators provided by the Variant (see page 3430) type.

See Also

TDateTime

TTimeStamp

TSQLTimeStampField

SQLTimeStampTODateTime

SQLTimeStampToStr

DateTimeToSQLTimeStamp

SQLDayOfWeek

25.4 Types

The following table lists types in this documentation.

Types

Name	Description
PSQLTimeStamp (see page 1671)	This is type SqlTimSt.PSQLTimeStamp.

25.4.1 SqlTimSt.PSQLTimeStamp Type

Pascal

```
PSQLTimeStamp = ^TSQLTimeStamp;
```

C++

```
^TSQLTimeStamp PSQLTimeStamp;
```

File

SqlTimSt

Description

This is type SqlTimSt.PSQLTimeStamp.

25.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NullSQLTimeStamp (see page 1672)	This is constant SqlTimSt.NullSQLTimeStamp.

25.5.1 SqlTimSt.NullSQLTimeStamp Constant

Pascal

```
NullSQLTimeStamp: TSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

C++

```
TSQLTimeStamp NullSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

File

SqlTimSt

Description

This is constant SqlTimSt.NullSQLTimeStamp.

26 StdActnMenus Namespace

This is namespace StdActnMenus.

Topics

Name	Description
Classes (see page 1674)	The following table lists classes in this documentation.
Functions (see page 1692)	The following table lists functions in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 1674)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 1676)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 1677)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 1679)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 1680)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 1682)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 1683)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 1688)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 1690)	This is class StdActnMenus.TStandardToolScrollBtn.

Functions

	Name	Description
	RegisterStandardMenus (see page 1692)	This is function StdActnMenus.RegisterStandardMenus.

26.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 1674)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 1676)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 1677)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 1679)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 1680)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 1682)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 1683)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 1688)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 1690)	This is class StdActnMenus.TStandardToolScrollBtn.

26.1.1 TStandardAddRemoveItem Class

TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.

Class Hierarchy

Pascal

```
TStandardAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TStandardAddRemoveItem : public TCustomAddRemoveItem;
```

File

StdActnMenus

Description

TStandardAddRemoveItem is the standard ActionBand menu for customizing action bands at runtime.

Do not create instances of TStandardAddRemoveItem as this will be done for you automatically if the class is the default customization menu. If you would like to replace the standard customization menu, use the class TCustomAddRemoveItem as a parent class for your replacement class. To make the change for your application, set the value of MenuAddRemoveItemClass (a global variable) to your own subclass of TCustomAddRemoveItem instead of TStandardAddRemoveItem.

See Also

TCustomAddRemoveItem

26.1.1.1 TStandardAddRemoveItem Members

The following tables list the members exposed by TStandardAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 1675)	Draws a background image on the customization menu.
	DrawGlyph (see page 1675)	Draws a symbol on the control.

26.1.1.2 TStandardAddRemoveItem Methods

The methods of the TStandardAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1675)	Draws a background image on the customization menu.
	DrawGlyph (see page 1675)	Draws a symbol on the control.

26.1.1.2.1 TStandardAddRemoveItem.DrawBackground Method

Draws a background image on the customization menu.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is largely the same as the inherited method, with some modifications to account for the method of selection. See the inherited method for more details.

See Also

DrawBackground

DrawGlyph (see page 1675)

DrawText

26.1.1.2.2 TStandardAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph performs the same function as the method it inherits from, and it does call the inherited method. The override sets many of the draw parameters based on the present state of the menu and how it has been selected.

See Also

DrawGlyph

DrawBackground (see page 1675)

DrawText

26.1.2 TStandardButtonControl Class

Class Hierarchy

Pascal

```
TStandardButtonControl = class(TCustomButtonControl);
```

C++

```
class TStandardButtonControl : public TCustomButtonControl;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardButtonControl.

26.1.2.1 TStandardButtonControl Members

The following tables list the members exposed by TStandardButtonControl.

Protected Methods

	Name	Description
	DrawFrame (see page 1676)	This is DrawFrame, a member of class TStandardButtonControl.
	DrawGlyph (see page 1677)	Draws a symbol on the control.
	DrawText (see page 1677)	This is DrawText, a member of class TStandardButtonControl.

26.1.2.2 TStandardButtonControl Methods

The methods of the TStandardButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1676)	This is DrawFrame, a member of class TStandardButtonControl.
	DrawGlyph (see page 1677)	Draws a symbol on the control.
	DrawText (see page 1677)	This is DrawText, a member of class TStandardButtonControl.

26.1.2.2.1 TStandardButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardButtonControl.

26.1.2.2.2 TStandardButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

26.1.2.2.3 TStandardButtonControl.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TStandardButtonControl.

26.1.3 TStandardCustomizePopup Class

Class Hierarchy

Pascal

```
TStandardCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TStandardCustomizePopup : public TCustomizeActionToolBar;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardCustomizePopup.

26.1.3.1 TStandardCustomizePopup Members

The following tables list the members exposed by TStandardCustomizePopup.

Protected Methods

	Name	Description
	CanAutoSize (see page 1678)	Indicates how the control can be resized.
	NCPaint (see page 1678)	This is NCPaint, a member of class TStandardCustomizePopup.
	PositionPopup (see page 1679)	This is PositionPopup, a member of class TStandardCustomizePopup.

26.1.3.2 TStandardCustomizePopup Methods

The methods of the TStandardCustomizePopup class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 1678)	Indicates how the control can be resized.
	NCPaint (see page 1678)	This is NCPaint, a member of class TStandardCustomizePopup.
	PositionPopup (see page 1679)	This is PositionPopup, a member of class TStandardCustomizePopup.

26.1.3.2.1 TStandardCustomizePopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

26.1.3.2.2 TStandardCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardCustomizePopup.

26.1.3.2.3 TStandardCustomizePopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardCustomizePopup.

26.1.4 TStandardDropDownButton Class

Class Hierarchy

Pascal

```
TStandardDropDownButton = class(TCustomDropDownButton);
```

C++

```
class TStandardDropDownButton : public TCustomDropDownButton;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardDropDownButton.

26.1.4.1 TStandardDropDownButton Members

The following tables list the members exposed by TStandardDropDownButton.

Protected Methods

	Name	Description
	DrawFrame (see page 1680)	This is DrawFrame, a member of class TStandardDropDownButton.
	GetPopupClass (see page 1680)	This is GetPopupClass, a member of class TStandardDropDownButton.

26.1.4.2 TStandardDropDownButton Methods

The methods of the TStandardDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1680)	This is DrawFrame, a member of class TStandardDropDownButton.
	GetPopupClass (see page 1680)	This is GetPopupClass, a member of class TStandardDropDownButton.

26.1.4.2.1 TStandardDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardDropDownButton.

26.1.4.2.2 TStandardDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TStandardDropDownButton.

26.1.5 TStandardMenuButton Class

TStandardMenuButton is the standard ActionBand menu button class.

Class Hierarchy

Pascal

```
TStandardMenuButton = class(TCustomMenuButton);
```

C++

```
class TStandardMenuButton : public TCustomMenuButton;
```

File

StdActnMenus

Description

TStandardMenuButton is the standard menu button class. If you would like to replace the standard menu button behaviors, use the class TCustomMenuButton as a parent class for your replacement class. To make the change for your application, set the value of MenuButtonControlClass (a global variable) to your own subclass of TCustomMenuButton instead of TStandardMenuButton.

See Also

TCustomMenuButton

26.1.5.1 TStandardMenuButton Members

The following tables list the members exposed by TStandardMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 1681)	Draws a background image on the control.
	DrawDesignFocus (see page 1681)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawFrame (see page 1682)	This is DrawFrame, a member of class TStandardMenuButton.
	DrawText (see page 1682)	This is DrawText, a member of class TStandardMenuButton.

26.1.5.2 TStandardMenuButton Methods

The methods of the TStandardMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1681)	Draws a background image on the control.
	DrawDesignFocus (see page 1681)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawFrame (see page 1682)	This is DrawFrame, a member of class TStandardMenuButton.
	DrawText (see page 1682)	This is DrawText, a member of class TStandardMenuButton.

26.1.5.2.1 TStandardMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

26.1.5.2.2 TStandardMenuButton.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuButton.

26.1.5.2.3 TStandardMenuButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardMenuButton.

26.1.5.2.4 TStandardMenuButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TStandardMenuButton.

26.1.6 TStandardMenuExpandBtn Class

Class Hierarchy

Pascal

```
TStandardMenuExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TStandardMenuExpandBtn : public TCustomMenuExpandBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardMenuExpandBtn.

26.1.6.1 TStandardMenuExpandBtn Members

The following tables list the members exposed by TStandardMenuExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 1683)	Draws a background image on the control.
	DrawFrame (see page 1683)	This is DrawFrame, a member of class TStandardMenuExpandBtn.

26.1.6.2 TStandardMenuExpandBtn Methods

The methods of the TStandardMenuExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1683)	Draws a background image on the control.
	DrawFrame (see page 1683)	This is DrawFrame, a member of class TStandardMenuExpandBtn.

26.1.6.2.1 TStandardMenuExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

26.1.6.2.2 TStandardMenuExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

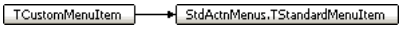
Description

This is DrawFrame, a member of class TStandardMenuExpandBtn.

26.1.7 TStandardMenuItem Class

TStandardMenuItem is the default class for menu items in an action band.

Class Hierarchy



Pascal

```
TStandardMenuItem = class(TCustomMenuItem);
```

C++

```
class TStandardMenuItem : public TCustomMenuItem;
```

File

StdActnMenus

Description

TStandardMenuItem is the class that action bands use to generate child controls that are menu items. Do not create instances of TStandardMenuItem. The action manager automatically handles the creating of TStandardMenuItem instances to represent the actions it contains.

If you want to replace the class used for menu items, derive your own replacement class from TCustomMenuItem. To substitute your replacement class, set the value of the global MenuItemControlClass variable to your TCustomMenuItem descendant.

See Also

TStandardMenuButton (see page 1680)

TStandardMenuPopup (see page 1688)

MenuItemControlClass

26.1.7.1 TStandardMenuItem Members

The following tables list the members exposed by TStandardMenuItem.

Protected Methods

	Name	Description
	CMTextchanged (see page 1685)	Responds when this menu item's caption changes.
V	DrawBackground (see page 1685)	Draws the background of the menu item.
V	DrawDesignFocus (see page 1685)	This is DrawDesignFocus, a member of class TStandardMenuItem.
V	DrawEdge (see page 1686)	This is DrawEdge, a member of class TStandardMenuItem.
V	DrawGlyph (see page 1686)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 1686)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
V	DrawLargeGlyph (see page 1687)	This is DrawLargeGlyph, a member of class TStandardMenuItem.
V	DrawSeparator (see page 1687)	Draws a separator.
V	DrawText (see page 1687)	Draws the menu item's text.

26.1.7.2 TStandardMenuItem Methods

The methods of the TStandardMenuItem class are listed here.

Protected Methods

	Name	Description
	CMTextchanged (see page 1685)	Responds when this menu item's caption changes.
V	DrawBackground (see page 1685)	Draws the background of the menu item.
V	DrawDesignFocus (see page 1685)	This is DrawDesignFocus, a member of class TStandardMenuItem.
V	DrawEdge (see page 1686)	This is DrawEdge, a member of class TStandardMenuItem.
V	DrawGlyph (see page 1686)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 1686)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
V	DrawLargeGlyph (see page 1687)	This is DrawLargeGlyph, a member of class TStandardMenuItem.

	DrawSeparator (see page 1687)	Draws a separator.
	DrawText (see page 1687)	Draws the menu item's text.

26.1.7.2.1 TStandardMenuItem.CMTextchanged Method

Responds when this menu item's caption changes.

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This method is called automatically in response to Windows messages containing a notification that the text of this menu item has changed.

Message represents the Windows message that notifies this menu item of a change to its text.

CMTextchanged recalculates the string it displays when the menu item is not underlining its accelerator key.

See Also

PersistentHotKeys

26.1.7.2.2 TStandardMenuItem.DrawBackground Method

Draws the background of the menu item.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is called automatically when the menu item needs to render its background (everything except the text and glyph).

PaintRect indicates the dimensions of the menu item on its canvas.

DrawBackground paints the background of the menu item so that its color reflects whether the item currently has focus, adjusting PaintRect to accommodate any glyph or checkmark that appears on the menu item.

See Also

DrawBackground

DrawGlyph (see page 1686)

DrawText (see page 1687)

26.1.7.2.3 TStandardMenuItem.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuItem.

26.1.7.2.4 TStandardMenuItem.DrawEdge Method

Pascal

```
procedure DrawEdge(Edges: TMenuEdges); override;
```

C++

```
virtual __fastcall DrawEdge(TMenuEdges Edges);
```

Description

This is DrawEdge, a member of class TStandardMenuItem.

26.1.7.2.5 TStandardMenuItem.DrawGlyph Method

Draws a symbol on the menu item.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph is called automatically when the menu item needs to render its glyph. This occurs if the menu item is checked or if the associated action client item specifies a glyph for this menu item using its ImageIndex property.

Location is the position on this menu item's canvas for the upper left corner of the glyph.

See Also

DrawGlyph

DrawBackground (see page 1685)

DrawText (see page 1687)

ImageIndex

26.1.7.2.6 TStandardMenuItem.DrawGlyphFrame Method

Pascal

```
procedure DrawGlyphFrame(const Location: TPoint);
```

C++

```
__fastcall DrawGlyphFrame(const TPoint Location);
```

Description

This is DrawGlyphFrame, a member of class TStandardMenuItem.

26.1.7.2.7 TStandardMenuItem.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TStandardMenuItem.

26.1.7.2.8 TStandardMenuItem.DrawSeparator Method

Draws a separator.

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

DrawSeparator is called automatically when the menu item represents a separator bar in the menu and it needs to paint itself. This method paints the separator bar.

Offset indicates the offset of the separator, in pixels, from the top of the menu item. DrawSeparator ignores this value and always uses a fixed value for menu items.

26.1.7.2.9 TStandardMenuItem.DrawText Method

Draws the menu item's text.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

DrawText is called automatically when the menu item needs to render its text.

ARect indicates the rectangle on the menu item's canvas where the text label should appear.

Flags supplies the value to use for the TextFlags property of the canvas.

Text is the text of the menu item.

DrawText checks whether the menu item should display underlines on accelerator keys, and draws the text accordingly, using a font color that reflects whether the menu item is selected and whether it is enabled.

See Also

DrawText

DrawBackground (see page 1685)

DrawGlyph (see page 1686)

PersistentHotKeys

TextFlags

26.1.8 TStandardMenuPopup Class

TStandardMenuPopup is the default menu popup style.

Class Hierarchy

Pascal

```
TStandardMenuPopup = class(TCustomActionPopupMenu);
```

C++

```
class TStandardMenuPopup : public TCustomActionPopupMenu;
```

File

StdActnMenus

Description

TStandardMenuPopup is the default menu popup style. If you would like to replace it with a style of your own, subclass TCustomActionPopupMenu to provide your own behaviors. To make the change for your application, set the value of MenuPopupClass (a global variable) to your own subclass of TCustomActionPopupMenu instead of TStandardMenuPopup.

See Also

TActionMainMenuBar

TStandardMenuButton (see page 1680)

TStandardMenuItem (see page 1683)

26.1.8.1 TStandardMenuPopup Members

The following tables list the members exposed by TStandardMenuPopup.

Protected Methods

	Name	Description
	CanAutoSize (see page 1689)	Indicates how the control can be resized.
	GetExpandBtnClass (see page 1689)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
	NCPaint (see page 1689)	This is NCPaint, a member of class TStandardMenuPopup.
	PositionPopup (see page 1690)	This is PositionPopup, a member of class TStandardMenuPopup.

26.1.8.2 TStandardMenuPopup Methods

The methods of the TStandardMenuPopup class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 1689)	Indicates how the control can be resized.

	GetExpandBtnClass (see page 1689)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
	NCPaint (see page 1689)	This is NCPaint, a member of class TStandardMenuPopup.
	PositionPopup (see page 1690)	This is PositionPopup, a member of class TStandardMenuPopup.

26.1.8.2.1 TStandardMenuPopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

26.1.8.2.2 TStandardMenuPopup.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TStandardMenuPopup.

26.1.8.2.3 TStandardMenuPopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardMenuPopup.

26.1.8.2.4 TStandardMenuPopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardMenuPopup.

26.1.9 TStandardToolScrollBtn Class

Class Hierarchy

Pascal

```
TStandardToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TStandardToolScrollBtn : public TCustomToolScrollBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardToolScrollBtn.

26.1.9.1 TStandardToolScrollBtn Members

The following tables list the members exposed by TStandardToolScrollBtn.

Protected Methods

	Name	Description
	DrawFrame (see page 1691)	This is DrawFrame, a member of class TStandardToolScrollBtn.

26.1.9.2 TStandardToolScrollBtn Methods

The methods of the TStandardToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1691)	This is DrawFrame, a member of class TStandardToolScrollBtn.

26.1.9.2.1 TStandardToolScrollBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardToolScrollBtn.

26.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterStandardMenus (see page 1692)	This is function StdActnMenus.RegisterStandardMenus.

26.2.1 StdActnMenus.RegisterStandardMenus Function

Pascal

```
procedure RegisterStandardMenus; deprecated;
```

C++

```
RegisterStandardMenus();
```

File

StdActnMenus

Description

This is function StdActnMenus.RegisterStandardMenus.

27 StdActns Namespace

This is namespace StdActns.

Topics

Name	Description
Classes (see page 1695)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 1875)	The following table lists structs, records, enums in this documentation.
Types (see page 1877)	The following table lists types in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 1696)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 1706)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 1715)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 1722)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 1726)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 1728)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 1730)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 1732)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 1734)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 1736)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 1739)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 1741)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 1749)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 1760)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 1767)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 1776)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 1785)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 1794)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 1803)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 1805)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 1806)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 1808)	THelpOnHelp brings up a help topic on how to use Help.

	THelpTopicSearch (see page 1810)	THelpTopicSearch brings up the help index.
	THintAction (see page 1811)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 1814)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 1823)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 1828)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 1838)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 1840)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 1850)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 1860)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 1864)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 1865)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 1867)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 1869)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 1871)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 1873)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Structs, Records, Enums

	Name	Description
	TBrowseForFolderOption (see page 1875)	This is record StdActns.TBrowseForFolderOption.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 1877)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 1877)	This is type StdActns.TBrowseOptions.
TCommonDialogClass (see page 1878)	This is type StdActns.TCommonDialogClass.

27.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 1696)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 1706)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 1715)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 1722)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 1726)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 1728)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 1730)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 1732)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 1734)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 1736)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 1739)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 1741)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 1749)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 1760)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 1767)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 1776)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 1785)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 1794)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 1803)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 1805)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 1806)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 1808)	THelpOnHelp brings up a help topic on how to use Help.
	THelpTopicSearch (see page 1810)	THelpTopicSearch brings up the help index.
	THintAction (see page 1811)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 1814)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 1823)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 1828)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 1838)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 1840)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 1850)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 1860)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 1864)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 1865)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 1867)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 1869)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 1871)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 1873)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

27.1.1 TBrowseForFolder Class

Class Hierarchy

Pascal

```
TBrowseForFolder = class(TCommonDialogAction);
```

C++

```
class TBrowseForFolder : public TCommonDialogAction;
```

File

StdActns

Description

This is class StdActns.TBrowseForFolder.

27.1.1.1 TBrowseForFolder Members

The following tables list the members exposed by TBrowseForFolder.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCallback (see page 1697)	This is DoCallback, a member of class TBrowseForFolder.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	DoValidate (see page 1697)	This is DoValidate, a member of class TBrowseForFolder.
V	GetDialogClass (see page 1698)	This is GetDialogClass, a member of class TBrowseForFolder.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1716)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1698)	Executes the associated dialog and generates an OnAccept or OnCancel event.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.
	Folder (see page 1700)	This is Folder, a member of class TBrowseForFolder.

Published Events

	Name	Description
	BeforeExecute (see page 1704)	Occurs immediately before the dialog appears.

	OnAccept (see page 1704)	This is OnAccept, a member of class TBrowseForFolder.
	OnCallback (see page 1705)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 1705)	This is OnCancel, a member of class TBrowseForFolder.
	OnHint (see page 1705)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 1705)	This is OnValidate, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 1699)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 1699)	Represents the caption of client controls and menu items.
	DialogCaption (see page 1700)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 1700)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1700)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1701)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1701)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1702)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1702)	Indicates the ImageIndex property for client controls and menu items.
	RootDir (see page 1702)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 1703)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1703)	Specifies the ShortCut property for client menu items.
	Visible (see page 1703)	Specifies the Visible property for client controls and menu items.

27.1.1.2 TBrowseForFolder Methods

The methods of the TBrowseForFolder class are listed here.

Protected Methods

	Name	Description
	DoCallback (see page 1697)	This is DoCallback, a member of class TBrowseForFolder.
	DoValidate (see page 1697)	This is DoValidate, a member of class TBrowseForFolder.
	GetDialogClass (see page 1698)	This is GetDialogClass, a member of class TBrowseForFolder.

Public Methods

	Name	Description
	ExecuteTarget (see page 1698)	Executes the associated dialog and generates an OnAccept or OnCancel event.

27.1.1.2.1 TBrowseForFolder.DoCallback Method

Pascal

```
function DoCallback(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoCallback(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoCallback, a member of class TBrowseForFolder.

27.1.1.2.2 TBrowseForFolder.DoValidate Method

Pascal

```
function DoValidate(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoValidate(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoValidate, a member of class TBrowseForFolder.

27.1.1.2.3 TBrowseForFolder.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

BeforeExecute (see page 1721)

27.1.1.2.4 TBrowseForFolder.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TBrowseForFolder.

27.1.1.3 TBrowseForFolder Properties

The properties of the TBrowseForFolder class are listed here.

Public Properties

	Name	Description
	Folder (see page 1700)	This is Folder, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 1699)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 1699)	Represents the caption of client controls and menu items.
	DialogCaption (see page 1700)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 1700)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1700)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1701)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1701)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1702)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1702)	Indicates the ImageIndex property for client controls and menu items.
	RootDir (see page 1702)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 1703)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1703)	Specifies the ShortCut property for client menu items.
	Visible (see page 1703)	Specifies the Visible property for client controls and menu items.

27.1.1.3.1 TBrowseForFolder.BrowseOptions Property

Pascal

```
property BrowseOptions: TBrowseOptions;
```

C++

```
__property TBrowseOptions BrowseOptions;
```

Description

This is BrowseOptions, a member of class TBrowseForFolder.

27.1.1.3.2 TBrowseForFolder.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.1.3.3 TBrowseForFolder.DialogCaption Property

Pascal

```
property DialogCaption: string;
```

C++

```
__property UnicodeString DialogCaption;
```

Description

This is DialogCaption, a member of class TBrowseForFolder.

27.1.1.3.4 TBrowseForFolder.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.1.3.5 TBrowseForFolder.Folder Property

Pascal

```
property Folder: string;
```

C++

```
__property UnicodeString Folder;
```

Description

This is Folder, a member of class TBrowseForFolder.

27.1.1.3.6 TBrowseForFolder.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.1.3.7 TBrowseForFolder.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.1.3.8 TBrowseForFolder.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property

for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.1.3.9 TBrowseForFolder.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.1.3.10 TBrowseForFolder.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.1.3.11 TBrowseForFolder.RootDir Property

Pascal

```
property RootDir: string;
```

C++

```
__property UnicodeString RootDir;
```

Description

This is RootDir, a member of class TBrowseForFolder.

27.1.1.3.12 TBrowseForFolder.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.1.3.13 TBrowseForFolder.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.1.3.14 TBrowseForFolder.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.1.4 TBrowseForFolder Events

The events of the TBrowseForFolder class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1704)	Occurs immediately before the dialog appears.
	OnAccept (see page 1704)	This is OnAccept, a member of class TBrowseForFolder.
	OnCallback (see page 1705)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 1705)	This is OnCancel, a member of class TBrowseForFolder.
	OnHint (see page 1705)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 1705)	This is OnValidate, a member of class TBrowseForFolder.

27.1.1.4.1 TBrowseForFolder.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.1.4.2 TBrowseForFolder.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TBrowseForFolder.

27.1.1.4.3 TBrowseForFolder.OnCallback Event

Pascal

```
property OnCallback: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnCallback;
```

Description

This is OnCallback, a member of class TBrowseForFolder.

27.1.1.4.4 TBrowseForFolder.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TBrowseForFolder.

27.1.1.4.5 TBrowseForFolder.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.1.4.6 TBrowseForFolder.OnValidate Event

Pascal

```
property OnValidate: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnValidate;
```

Description

This is OnValidate, a member of class TBrowseForFolder.

27.1.2 TColorSelect Class

TColorSelect is the standard action for displaying a color selection dialog.

Class Hierarchy

Pascal

```
TColorSelect = class(TCommonDialogAction);
```

C++

```
class TColorSelect : public TCommonDialogAction;
```

File

StdActns

Description

Add TColorSelect to an action list to add a modal color selection dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the color selection dialog when invoked. Use the OnAccept (see page 1713) event to respond after the user selects a color using the dialog or the OnCancel (see page 1713) event to respond if the user cancels from the dialog. In the event handler for these events, you can read the color from the dialog specified by the Dialog (see page 1708) property.

See Also

TColorDialog

27.1.2.1 TColorSelect Members

The following tables list the members exposed by TColorSelect.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1707)	This is GetDialogClass, a member of class TColorSelect.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1712)	Occurs immediately before the dialog appears.
	OnAccept (see page 1713)	This is OnAccept, a member of class TColorSelect.
	OnCancel (see page 1713)	This is OnCancel, a member of class TColorSelect.
	OnHint (see page 1713)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1714)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1708)	Represents the caption of client controls and menu items.
	Dialog (see page 1708)	Provides access the color selection dialog that the action executes.
	Enabled (see page 1709)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1709)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1709)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1710)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1710)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1711)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1711)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1712)	Specifies the ShortCut property for client menu items.
	Visible (see page 1712)	Specifies the Visible property for client controls and menu items.

27.1.2.2 TColorSelect Methods

The methods of the TColorSelect class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1707)	This is GetDialogClass, a member of class TColorSelect.

27.1.2.2.1 TColorSelect.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TColorSelect.

27.1.2.3 TColorSelect Properties

The properties of the TColorSelect class are listed here.

Published Properties

	Name	Description
	Caption (see page 1708)	Represents the caption of client controls and menu items.
R	Dialog (see page 1708)	Provides access the color selection dialog that the action executes.
	Enabled (see page 1709)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1709)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1709)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1710)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1710)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1711)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1711)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1712)	Specifies the ShortCut property for client menu items.
	Visible (see page 1712)	Specifies the Visible property for client controls and menu items.

27.1.2.3.1 TColorSelect.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.2.3.2 TColorSelect.Dialog Property

Provides access the color selection dialog that the action executes.

Pascal

```
property Dialog: TColorDialog;
```

C++

```
__property TColorDialog Dialog;
```

Description

Use Dialog to configure the color selection dialog before the action fires or to read the selected color after the action executes the dialog. For example, you can set the initially selected color in the dialog at design time using the Object Inspector or read the dialog's Color property in an OnAccept (see page 1713) event handler to make use of the selected color.

When using a color select action, you do not place a TColorDialog component on the form or data module at design time. Instead, you add the TColorSelect (see page 1706) to your action list and then use the Dialog property. The color selection dialog component is created by the TColorSelect (see page 1706) constructor.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Color

27.1.2.3.3 TColorSelect.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.2.3.4 TColorSelect.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.2.3.5 TColorSelect.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.2.3.6 TColorSelect.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.2.3.7 TColorSelect.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.2.3.8 TColorSelect.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.2.3.9 TColorSelect.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.2.3.10 TColorSelect.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.2.3.11 TColorSelect.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.2.4 TColorSelect Events

The events of the TColorSelect class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1712)	Occurs immediately before the dialog appears.
	OnAccept (see page 1713)	This is OnAccept, a member of class TColorSelect.
	OnCancel (see page 1713)	This is OnCancel, a member of class TColorSelect.
	OnHint (see page 1713)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1714)	Occurs when the application is idle or when the action list updates.

27.1.2.4.1 TColorSelect.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.2.4.2 TColorSelect.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TColorSelect.

27.1.2.4.3 TColorSelect.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TColorSelect.

27.1.2.4.4 TColorSelect.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.2.4.5 TColorSelect.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.3 TCommonDialogAction Class

TCommonDialogAction is the base class for action objects that display a dialog.

Class Hierarchy

Pascal

```
TCommonDialogAction = class(TCustomAction);
```

C++

```
class TCommonDialogAction : public TCustomAction;
```

File

StdActns

Description

TCommonDialogAction implements the common behavior for actions that display a dialog when executed. Each descendant class represents a specific dialog. Do not use TCommonDialogAction directly in an application. Instead, use the specific TCommonDialogAction descendant that represents the type of dialog you want to execute. The various TCommonDialogAction descendants are listed in the See also list.

TCommonDialogAction objects do not require a specific type of control to be active: they handle any target. When executed, the dialog action displays its dialog, and then generates an OnAccept (see page 1721) or OnCancel (see page 1722) event, which you can use to process the results from the dialog.

When using TCommonDialogAction desendants, there is no need to place the dialog component in your form or data module. The action object creates its own dialog component.

See Also

TColorSelect (see page 1706)

TFileOpen (see page 1749)

TFilePrintSetup (see page 1776)

TFileSaveAs (see page 1785)

TSearchFind (see page 1828)

TSearchReplace (see page 1850)

TFontEdit (see page 1794)

TOpenPicture

TSavePicture

27.1.3.1 TCommonDialogAction Members

The following tables list the members exposed by TCommonDialogAction.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1721)	Occurs immediately before the dialog appears.
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

27.1.3.2 TCommonDialogAction.Create Constructor

Creates an instance of TCommonDialogAction.

Pascal

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

C++

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

Description

Most applications do not need to call the TCommonDialogAction constructor because actions are added at design time to an action list or action manager.

The TCommonDialogAction constructor creates an instance of its associated dialog, which descendant classes make available through the Dialog property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCommonDialogAction instance.

See Also

Owner

Actions

27.1.3.3 TCommonDialogAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCommonDialogAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

27.1.3.4 TCommonDialogAction Fields

The fields of the TCommonDialogAction class are listed here.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

27.1.3.4.1 TCommonDialogAction.FDialog Field

Pascal

```
FDialog: TCommonDialog;
```

C++

```
TCommonDialog FDialog;
```

Description

This is FDialog, a member of class TCommonDialogAction.

27.1.3.5 TCommonDialogAction Methods

The methods of the TCommonDialogAction class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

27.1.3.5.1 TCommonDialogAction.DoAccept Method

Pascal

```
procedure DoAccept; virtual;
```

C++

```
virtual __fastcall DoAccept();
```

Description

This is DoAccept, a member of class TCommonDialogAction.

27.1.3.5.2 TCommonDialogAction.DoCancel Method

Pascal

```
procedure DoCancel; virtual;
```

C++

```
virtual __fastcall DoCancel();
```

Description

This is DoCancel, a member of class TCommonDialogAction.

27.1.3.5.3 TCommonDialogAction.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute (see page 1721) event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept (see page 1721) event or an OnCancel (see page 1722) event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

BeforeExecute (see page 1721)

27.1.3.5.4 TCommonDialogAction.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; virtual;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TCommonDialogAction.

27.1.3.5.5 TCommonDialogAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because the execution of the dialog does not depend on any target object.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

27.1.3.5.6 TCommonDialogAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

27.1.3.5.7 TCommonDialogAction.SetupDialog Method

Pascal

```
procedure SetupDialog; virtual;
```

C++

```
virtual __fastcall SetupDialog();
```

Description

This is SetupDialog, a member of class TCommonDialogAction.

27.1.3.6 TCommonDialogAction Properties

The properties of the TCommonDialogAction class are listed here.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

27.1.3.6.1 TCommonDialogAction.ExecuteResult Property

Indicates the result of executing the associated dialog.

Pascal

```
property ExecuteResult: Boolean;
```

C++

```
__property Boolean ExecuteResult;
```

Description

When the dialog action fires, it executes the dialog specified by the CommonDialog property and sets ExecuteResult the value returned by that dialog's Execute method. Thus, ExecuteResult is false if the dialog has not executed or the user canceled when

the dialog executed, and true if the user exited the dialog by clicking OK.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

27.1.3.7 TCommonDialogAction Events

The events of the TCommonDialogAction class are listed here.

Public Events

	Name	Description
	BeforeExecute (see page 1721)	Occurs immediately before the dialog appears.
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

27.1.3.7.1 TCommonDialogAction.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write (see page 3789) a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.3.7.2 TCommonDialogAction.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TCommonDialogAction.

27.1.3.7.3 TCommonDialogAction.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TCommonDialogAction.

27.1.4 TEditAction Class

TEditAction is the base class for action objects that perform standard edits on an edit control.

Class Hierarchy

Pascal

```
TEditAction = class(TAction);
```

C++

```
class TEditAction : public TAction;
```

File

StdActns

Description

TEditAction is the common base class for actions that work with edit controls. Each descendant class performs a specific type of edit on the target edit control. Do not use TEditAction directly in an application. Instead, use a TEditAction descendant that performs a specific function (for example, Cut, Copy (see page 3648), Paste, Delete (see page 3653), Select All, or Undo). The various TEditAction descendants are listed in the See also list.

See Also

TEditCopy (see page 1726)

TEditCut (see page 1728)

TEditDelete (see page 1730)

TEditPaste (see page 1732)

TEditSelectAll (see page 1734)

TEditUndo (see page 1736)

TEditRedo

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TRichEditAlignCenter

TRIchEditAlignLeft

TRichEditAlignRight

TRichEditBullets

TRichEditStrikeOut

27.1.4.1 TEditAction Members

The following tables list the members exposed by TEditAction.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1725)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.4.2 TEditAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEditAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

27.1.4.3 TEditAction Methods

The methods of the TEditAction class are listed here.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1725)	Enables or disables the action, depending on the currently active control in the form.

27.1.4.3.1 TEditAction.GetControl Method

Returns the target of the edit action.

Pascal

```
function GetControl(Target: TObject): TCustomEdit; virtual;
```

C++

```
virtual __fastcall TCustomEdit GetControl(TObject * Target);
```

Description

TEditAction (see page 1722) uses the protected GetControl method internally to obtain the target of the action.

Target is the currently active object, or the value of the Control (see page 1726) property.

GetControl returns the target object as an edit control.

See Also

ExecuteTarget

Control (see page 1726)

27.1.4.3.2 TEditAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target of is a TCustomEdit (see page 2263) descendant (or that the EditControl (see page 1726) property is set if Target is nil (Delphi) or NULL (C++)). HandlesTarget returns true if there is an edit control to act as target, false otherwise.

See Also

ExecuteTarget

TCustomEdit (see page 2263)

EditControl

27.1.4.3.3 TEditAction.Notification Method

Responds when objects are added or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TEditAction (see page 1722) responds to notifications that the target edit control (as specified by the Control (see page 1726) property) is about to be freed by setting Control (see page 1726) to nil (Delphi) or NULL (C++).

See Also

ExecuteTarget

Control (see page 1726)

27.1.4.3.4 TEditAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TEditAction (see page 1722), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control and, if the action's function requires it, that some of its text is selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

27.1.4.4 TEditAction Properties

The properties of the TEditAction class are listed here.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.4.4.1 TEditAction.Control Property

Provides access to the target of the action.

Pascal

```
property Control: TCustomEdit;
```

C++

```
__property TCustomEdit Control;
```

Description

EditControl is the edit control on which the action performs its function. Set the value of Control to limit the action so that it only operates on a specific edit control. If Control is unassigned, the action dynamically locates the target edit control when it executes.

See Also

ExecuteTarget

27.1.5 TEditCopy Class

TEditCopy copies the selected text in a target edit control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCopy = class(TEditAction);
```

C++

```
class TEditCopy : public TEditAction;
```

File

StdActns

Description

Add TEditCopy to an action list to allow users to copy selected text to the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to copy the currently selected text in an active text control to the clipboard.

TEditCopy is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have any text selected.

See Also

TEditCut (see page 1728)

TEditDelete (see page 1730)

TEditPaste (see page 1732)

TEditSelectAll (see page 1734)

TEditUndo (see page 1736)

TEditRedo

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.5.1 TEditCopy Members

The following tables list the members exposed by TEditCopy.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1727)	Copies selected text in the target control to the Windows clipboard.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1725)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.5.2 TEditCopy Methods

The methods of the TEditCopy class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1727)	Copies selected text in the target control to the Windows clipboard.

27.1.5.2.1 TEditCopy.ExecuteTarget Method

Copies selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEdit (see page 2263) descendant), and if so, calls its CopyToClipboard method, copying the selected text to the clipboard.

See Also

TCustomEdit (see page 2263)

CopyToClipboard (see page 2268)

27.1.6 TEditCut Class

TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCut = class(TEditAction);
```

C++

```
class TEditCut : public TEditAction;
```

File

StdActns

Description

Add TEditCut to an action list to allow users to cut selected text to the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to cut the currently selected text in an active text control to the clipboard.

TEditCut is designed to work with an edit control (TCustomEdit (see page 2263) or TCustomMemo (see page 2361) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have a selection to cut.

See Also

TEditCopy (see page 1726)

TEditDelete (see page 1730)

TEditPaste (see page 1732)

TEditSelectAll (see page 1734)

TEditUndo (see page 1736)

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.6.1 TEditCut Members

The following tables list the members exposed by TEditCut.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1729)	Cuts selected text in the target control to the Windows clipboard.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1725)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.6.2 TEditCut Methods

The methods of the TEditCut class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1729)	Cuts selected text in the target control to the Windows clipboard.

27.1.6.2.1 TEditCut.ExecuteTarget Method

Cuts selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEditControl descendant), and if so, calls its CutToClipboard method, cutting the selected text to the clipboard.

See Also

TCustomEdit (see page 2263)

CutToClipboard (see page 2271)

27.1.7 TEditDelete Class

TEditDelete deletes the selected text in a target edit or memo control.

Class Hierarchy

Pascal

```
TEditDelete = class(TEditAction);
```

C++

```
class TEditDelete : public TEditAction;
```

File

StdActns

Description

Add TEditDelete to an action list to let users delete selected text using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to delete the currently selected text in an active text control.

TEditDelete is designed to work with an edit control (TCustomEdit (see page 2263) or TCustomMemo (see page 2361) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the active edit control does not have any selected text.

See Also

TEditCopy (see page 1726)

TEditCut (see page 1728)

TEditPaste (see page 1732)

TEditSelectAll (see page 1734)

TEditUndo (see page 1736)

TCustomEdit (see page 2263)

TCustomMemo (see page 2361)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.7.1 TEditDelete Members

The following tables list the members exposed by TEditDelete.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.

	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1731)	Deletes selected text in the target control.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1731)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.7.2 TEditDelete Methods

The methods of the TEditDelete class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1731)	Deletes selected text in the target control.
	UpdateTarget (see page 1731)	Enables or disables the action, depending on the currently active control in the form.

27.1.7.2.1 TEditDelete.ExecuteTarget Method

Deletes selected text in the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2263) descendant, and if so, calls its ClearSelection method, deleting the selected text.

See Also

TCustomEdit (see page 2263)

EditControl

MemoControl

ClearSelection (see page 2267)

27.1.7.2.2 TEditDelete.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TEditDelete (see page 1730), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that has some of its text selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

27.1.8 TEditPaste Class

TEditPaste pastes text from the clipboard into the active edit or memo control.

Class Hierarchy

Pascal

```
TEditPaste = class(TEditAction);
```

C++

```
class TEditPaste : public TEditAction;
```

File

StdActns

Description

Add TEditPaste to an action list to enable users to paste text from the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to paste text from the clipboard into an active text control.

TEditPaste is designed to work with an edit control (TCustomEdit (see page 2263) or TCustomMemo (see page 2361) descendant) as a target. It automatically disables itself if the clipboard does not contain text.

See Also

TEditCopy (see page 1726)

TEditCut (see page 1728)

TEditDelete (see page 1730)

TEditSelectAll (see page 1734)

TEditUndo (see page 1736)

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.8.1 TEditPaste Members

The following tables list the members exposed by TEditPaste.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1733)	Pastes text from the clipboard into the target edit control.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1734)	Enables or disables the action, depending on the contents of the clipboard.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.8.2 TEditPaste Methods

The methods of the TEditPaste class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1733)	Pastes text from the clipboard into the target edit control.
	UpdateTarget (see page 1734)	Enables or disables the action, depending on the contents of the clipboard.

27.1.8.2.1 TEditPaste.ExecuteTarget Method

Pastes text from the clipboard into the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2263) descendant, and if so, calls its PasteFromClipboard

method, pasting in any text on the clipboard.

See Also

TCustomEdit (see page 2263)

EditControl

MemoControl

PasteFromClipboard (see page 2276)

27.1.8.2.2 TEditPaste.UpdateTarget Method

Enables or disables the action, depending on the contents of the clipboard.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditPaste (see page 1732), UpdateTarget checks whether the clipboard contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

27.1.9 TEditSelectAll Class

TEditSelectAll selects all text in an active text control.

Class Hierarchy

Pascal

```
TEditSelectAll = class(TEditAction);
```

C++

```
class TEditSelectAll : public TEditAction;
```

File

StdActns

Description

Add TEditSelectAll to an action list to allow users to select all the text in a text control using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to select all text in an active text control.

TEditSelectAll is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the edit control does not have any text.

See Also

TEditCopy (see page 1726)

TEditCut (see page 1728)

TEditDelete (see page 1730)

TEditPaste (see page 1732)

TEditUndo (see page 1736)

TEditRedo

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.9.1 TEditSelectAll Members

The following tables list the members exposed by TEditSelectAll.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1736)	Selects all text in the target edit control.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1736)	Enables or disables the action, depending on the contents of the target edit control.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.9.2 TEditSelectAll Methods

The methods of the TEditSelectAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1736)	Selects all text in the target edit control.
	UpdateTarget (see page 1736)	Enables or disables the action, depending on the contents of the target edit control.

27.1.9.2.1 TEditSelectAll.ExecuteTarget Method

Selects all text in the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2263) descendant, and if so, calls its SelectAll method, selecting all of its text.

See Also

TCustomEdit (see page 2263)

EditControl

MemoControl

SelectAll

27.1.9.2.2 TEditSelectAll.UpdateTarget Method

Enables or disables the action, depending on the contents of the target edit control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditSelectAll (see page 1734), UpdateTarget checks whether Target is a text control that contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

27.1.10 TEditUndo Class

TEditUndo backs out the last change in an edit control.

Class Hierarchy

Pascal

```
TEditUndo = class(TEditAction);
```

C++

```
class TEditUndo : public TEditAction;
```

File

StdActns

Description

Add TEditUndo to an action list to allow users to back out edits using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to undo the last edit in an active text control.

TEditUndo is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control that can back out the last edit.

See Also

TEditCopy (see page 1726)

TEditCut (see page 1728)

TEditPaste (see page 1732)

TEditSelectAll (see page 1734)

TEditDelete (see page 1730)

TCustomEdit (see page 2263)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1850)

27.1.10.1 TEditUndo Members

The following tables list the members exposed by TEditUndo.

Protected Methods

	Name	Description
	GetControl (see page 1724)	Returns the target of the edit action.
	Notification (see page 1725)	Responds when objects are added or removed.

Public Methods

	Name	Description
	Destroy (see page 1723)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1738)	Backs out the last edit from the target control.
	HandlesTarget (see page 1724)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1738)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1726)	Provides access to the target of the action.

27.1.10.2 TEditUndo Methods

The methods of the TEditUndo class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1738)	Backs out the last edit from the target control.
	UpdateTarget (see page 1738)	Enables or disables the action, depending on the currently active control in the form.

27.1.10.2.1 TEditUndo.ExecuteTarget Method

Backs out the last edit from the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2263) descendant, and if so, calls its Undo method, backing out the last change.

See Also

TCustomEdit (see page 2263)

Undo (see page 2282)

27.1.10.2.2 TEditUndo.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditUndo (see page 1736), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that is in a state where it can back out changes. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget

sets Enabled to false.

See Also

Enabled

CanUndo (see page 2285)

27.1.11 TFileAction Class

TFileAction is the base class for action objects that display a file selection dialog.

Class Hierarchy

Pascal

```
TFileAction = class(TCommonDialogAction);
```

C++

```
class TFileAction : public TCommonDialogAction;
```

File

StdActns

Description

TFileAction implements the FileName (see page 1741) property that is common to action objects that display a file selection object.

See Also

TFileOpen (see page 1749)

TFileSaveAs (see page 1785)

TOpenDialog

27.1.11.1 TFileAction Members

The following tables list the members exposed by TFileAction.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1740)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1741)	Specifies the file name for the file selection dialog.

Public Events

	Name	Description
	BeforeExecute (see page 1721)	Occurs immediately before the dialog appears.
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

27.1.11.2 TFileAction Methods

The methods of the TFileAction class are listed here.

Protected Methods

	Name	Description
	GetDialog (see page 1740)	This is GetDialog, a member of class TFileAction.

27.1.11.2.1 TFileAction.GetDialog Method

Pascal

```
function GetDialog: TOpenDialog;
```

C++

```
__fastcall TOpenDialog GetDialog();
```

Description

This is GetDialog, a member of class TFileAction.

27.1.11.3 TFileAction Properties

The properties of the TFileAction class are listed here.

Protected Properties

	Name	Description
	FileName (see page 1741)	Specifies the file name for the file selection dialog.

27.1.11.3.1 TFileAction.FileName Property

Specifies the file name for the file selection dialog.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Applications can't use the protected FileName property. To access the name of the file selected by the user, use the Dialog property of a TFileAction (see page 1739) descendant.

FileName is the same as the FileName property of the associated dialog.

See Also

FileName

onaccept (see page 1721)

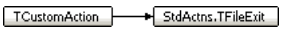
Dialog (see page 1753)

Dialog (see page 1788)

27.1.12 TFileExit Class

TFileExit is the standard action for shutting down an application.

Class Hierarchy



Pascal

```
TFileExit = class(TCustomAction);
```

C++

```
class TFileExit : public TCustomAction;
```

File

StdActns

Description

Add TFileExit to an action list to let users shut down the application using a menu or tool bar. Controls linked to this action cause the application to shut down by closing the main form.

See Also

TCustomForm

TAPplication

27.1.12.1 TFileExit Members

The following tables list the members exposed by TFileExit.

Public Methods

	Name	Description
	ExecuteTarget (see page 1742)	Shuts down the main form of the application.
	HandlesTarget (see page 1743)	Indicates whether the action can execute.

Published Events

	Name	Description
	OnHint (see page 1747)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1748)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1743)	Represents the caption of client controls and menu items.
	Enabled (see page 1744)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1744)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1745)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1745)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1746)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1746)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1746)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1747)	Specifies the ShortCut property for client menu items.
	Visible (see page 1747)	Specifies the Visible property for client controls and menu items.

27.1.12.2 TFileExit Methods

The methods of the TFileExit class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1742)	Shuts down the main form of the application.
	HandlesTarget (see page 1743)	Indicates whether the action can execute.

27.1.12.2.1 TFileExit.ExecuteTarget Method

Shuts down the main form of the application.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and calls the Close (see page 3643) method of the application's main form. This causes the application to shut down.

See Also

MainForm

Close

27.1.12.2.2 TFileExit.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because shutting down the application does not depend on any target object.

27.1.12.3 TFileExit Properties

The properties of the TFileExit class are listed here.

Published Properties

	Name	Description
	Caption (see page 1743)	Represents the caption of client controls and menu items.
	Enabled (see page 1744)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1744)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1745)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1745)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1746)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1746)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1746)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1747)	Specifies the ShortCut property for client menu items.
	Visible (see page 1747)	Specifies the Visible property for client controls and menu items.

27.1.12.3.1 TFileExit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.12.3.2 TFileExit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.12.3.3 TFileExit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.12.3.4 TFileExit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.12.3.5 TFileExit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.12.3.6 TFileExit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.12.3.7 TFileExit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.12.3.8 TFileExit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.12.3.9 TFileExit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.12.3.10 TFileExit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.12.4 TFileExit Events

The events of the TFileExit class are listed here.

Published Events

	Name	Description
	OnHint (see page 1747)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1748)	Occurs when the application is idle or when the action list updates.

27.1.12.4.1 TFileExit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.12.4.2 TFileExit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

27.1.13 TFileOpen Class

TFileOpen is the standard action for displaying a file open dialog.

Class Hierarchy

Pascal

```
TFileOpen = class(TFileAction);
```

C++

```
class TFileOpen : public TFileAction;
```

File

StdActns

Description

Add TFileOpen to an action list to add a common file open dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a file open dialog when invoked. Use the OnAccept (see page 1758) event to respond after the user selects a file name in the file open dialog or the OnCancel (see page 1758) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog specified by the Dialog (see page 1753) property.

See Also

TFileOpenWith (see page 1760)

TFileSaveAs (see page 1785)

TOpenDialog

TOpenPicture

27.1.13.1 TFileOpen Members

The following tables list the members exposed by TFileOpen.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1740)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 1752)	This is GetDialogClass, a member of class TFileOpen.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1741)	Specifies the file name for the file selection dialog.

Public Methods

	Name	Description
V	Create (see page 1751)	Creates an instance of TFileOpen (see page 1749).
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1751)	Executes the associated dialog and generates an OnAccept or OnCancel event.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1758)	Occurs immediately before the dialog appears.
	OnAccept (see page 1758)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 1758)	This is OnCancel, a member of class TFileOpen.
	OnHint (see page 1758)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1759)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1753)	Represents the caption of client controls and menu items.
R	Dialog (see page 1753)	Provides access the open dialog that the action executes.
	Enabled (see page 1753)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1754)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1754)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1755)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1755)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1755)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1756)	Specifies the short cuts (in addition to ShortCut) for triggering clients.

	ShortCut (see page 1756)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1757)	Use the default application to open the file.
	Visible (see page 1757)	Specifies the Visible property for client controls and menu items.

27.1.13.2 TFileOpen.Create Constructor

Creates an instance of TFileOpen (see page 1749).

Pascal

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

C++

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

Description

Most applications do not need to call the TFileOpen (see page 1749) constructor because actions are added at design time to an action manager or action list.

The TFileOpen (see page 1749) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1753) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileOpen (see page 1749) instance.

See Also

Dialog (see page 1753)

Owner

27.1.13.3 TFileOpen Methods

The methods of the TFileOpen class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1752)	This is GetDialogClass, a member of class TFileOpen.

Public Methods

	Name	Description
	ExecuteTarget (see page 1751)	Executes the associated dialog and generates an OnAccept or OnCancel event.

27.1.13.3.1 TFileOpen.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for

example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

BeforeExecute (see page 1721)

27.1.13.3.2 TFileOpen.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileOpen.

27.1.13.4 TFileOpen Properties

The properties of the TFileOpen class are listed here.

Published Properties

	Name	Description
	Caption (see page 1753)	Represents the caption of client controls and menu items.
R	Dialog (see page 1753)	Provides access the open dialog that the action executes.
	Enabled (see page 1753)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1754)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1754)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1755)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1755)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1755)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1756)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1756)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1757)	Use the default application to open the file.
	Visible (see page 1757)	Specifies the Visible property for client controls and menu items.

27.1.13.4.1 TFileOpen.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.13.4.2 TFileOpen.Dialog Property

Provides access the open dialog that the action executes.

Pascal

```
property Dialog: TOpenDialog;
```

C++

```
__property TOpenDialog Dialog;
```

Description

Use Dialog to configure the open dialog before the action fires. For example, you can set the open dialog's default extension or initially selected file at design time using the Object Inspector. From an OnAccept (see page 1758) event handler, use Dialog to read the selected file name.

When using a file open action, you do not place a TOpenDialog component on the form or data module at design time. Instead, you add the TFileOpen (see page 1749) action to your action list and then use the Dialog property. The open dialog component is created by the TFileOpen (see page 1749) constructor.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

FileName

27.1.13.4.3 TFileOpen.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.13.4.4 TFileOpen.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.13.4.5 TFileOpen.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.13.4.6 TFileOpen.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.13.4.7 TFileOpen.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.13.4.8 TFileOpen.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.13.4.9 TFileOpen.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.13.4.10 TFileOpen.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.13.4.11 TFileOpen.UseDefaultApp Property

Use the default application to open the file.

Pascal

```
property UseDefaultApp: Boolean;
```

C++

```
__property Boolean UseDefaultApp;
```

Description

When set to True, UseDefaultApp automatically uses the default application as specified in the registry to open the file. Otherwise, TFileOpen (see page 1749) displays the FileOpen (see page 4200) dialog. The default is False.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

FileName

27.1.13.4.12 TFileOpen.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.13.5 TFileOpen Events

The events of the TFileOpen class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1758)	Occurs immediately before the dialog appears.
	OnAccept (see page 1758)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 1758)	This is OnCancel, a member of class TFileOpen.
	OnHint (see page 1758)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1759)	Occurs when the application is idle or when the action list updates.

27.1.13.5.1 TFileOpen.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.13.5.2 TFileOpen.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFileOpen.

27.1.13.5.3 TFileOpen.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFileOpen.

27.1.13.5.4 TFileOpen.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.13.5.5 TFileOpen.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
```

```
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

27.1.14 TFileOpenWith Class

TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.

Class Hierarchy

Pascal

```
TFileOpenWith = class(TFileOpen);
```

C++

```
class TFileOpenWith : public TFileOpen;
```

File

StdActns

Description

Add TFileOpenWith to an action list to add the Open With dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the Open With dialog and then use the chosen application to open the file specified by the FileName (see page 1763) property. If the FileName (see page 1763) property was not previously set, TFileOpenWith first displays an Open dialog in which the user can select a file name before displaying the Open With dialog for the user to specify what application to use when opening that file.

Note: TFileOpenWith always displays the Open With dialog, even if the specified file already has an extension with a registered association. To let users open a file by letting Windows choose the application based on the extension (or bring up the Open With dialog automatically when there is no association for the extension), use the TFileRun action instead.

You do not need to do anything to launch the selected application. TFileOpenWith automatically opens the file using the application that the user selects.

Note: The Dialog (see page 1753) property of TFileOpenWith represents the Open dialog in which a user selects the file to open, not the Open With dialog in which they choose the application to user for opening that file. Similarly, the OnAccept (see page 1758) and OnCancel (see page 1758) events do not occur after the action displays the Open With dialog – rather, they occur after the Open dialog that appears when the FileName (see page 1763) property is not preset.

See Also

TFileSaveAs (see page 1785)

TFileOpen (see page 1749)

TOpenPicture

TFileRun

27.1.14.1 TFileOpenWith Members

The following tables list the members exposed by TFileOpenWith.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1740)	This is GetDialog, a member of class TFileAction.
	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
	GetDialogClass (see page 1752)	This is GetDialogClass, a member of class TFileOpen.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Create (see page 1751)	Creates an instance of TFileOpen (see page 1749).
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1762)	Displays the Open With dialog and then opens a file with the application the user selects.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	AfterOpen (see page 1765)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 1765)	Occurs immediately before the dialog appears.
	OnAccept (see page 1758)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 1758)	This is OnCancel, a member of class TFileOpen.
	OnHint (see page 1758)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1766)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1753)	Represents the caption of client controls and menu items.
R	Dialog (see page 1753)	Provides access the open dialog that the action executes.
	Enabled (see page 1753)	Indicates whether client controls and menu items are enabled.
	FileName (see page 1763)	Specifies the file to be opened when the action executes.
	HelpContext (see page 1763)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1764)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1764)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1755)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1755)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1756)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1756)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1757)	Use the default application to open the file.
	Visible (see page 1757)	Specifies the Visible property for client controls and menu items.

27.1.14.2 TFileOpenWith Methods

The methods of the TFileOpenWith class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 1762)	Displays the Open With dialog and then opens a file with the application the user selects.

27.1.14.2.1 TFileOpenWith.ExecuteTarget Method

Displays the Open With dialog and then opens a file with the application the user selects.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and does the following:

1. If the FileName (see page 1763) property is not set, it displays the Open dialog (specified by the Dialog (see page 1753) property) to let the user select a file to open.
2. If it displays the Open dialog, it generates an OnAccept (see page 1758) or OnCancel (see page 1758) event, depending on whether the user exited the Open dialog by clicking OK or Cancel.
3. It displays the Open With dialog for the user to choose an application with which to view the file specified by FileName (see page 1763) or selected in the Open dialog.
4. It launches the selected application, displaying the specified File.

See Also

FileName (see page 1763)

Dialog (see page 1753)

onaccept (see page 1721)

oncancel (see page 1722)

27.1.14.3 TFileOpenWith Properties

The properties of the TFileOpenWith class are listed here.

Published Properties

	Name	Description
	FileName (see page 1763)	Specifies the file to be opened when the action executes.
	HelpContext (see page 1763)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1764)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1764)	Indicates the mechanism for client controls and menu items to use when invoking help.

27.1.14.3.1 TFileOpenWith.FileName Property

Specifies the file to be opened when the action executes.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file to open with the application that the user selects in the Open With dialog. If the FileName property is not set, then when the action executes, it displays an Open dialog (specified by the Dialog (see page 1753) property) in which the user can select a file before displaying the Open With dialog.

Note: When the action displays the Open dialog, it uses the selected file to set FileName. Thus, if you want to let the user select a file the next time the action executes, you must clear the FileName property in an OnAccept (see page 1758) event handler.

See Also

Dialog (see page 1753)

ExecuteTarget (see page 1762)

onaccept (see page 1721)

27.1.14.3.2 TFileOpenWith.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.14.3.3 TFileOpenWith.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.14.3.4 TFileOpenWith.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.14.4 TFileOpenWith Events

The events of the TFileOpenWith class are listed here.

Published Events

	Name	Description
	AfterOpen (see page 1765)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 1765)	Occurs immediately before the dialog appears.
	OnUpdate (see page 1766)	Occurs when the application is idle or when the action list updates.

27.1.14.4.1 TFileOpenWith.AfterOpen Event

Occurs after an application completes opening an Open With dialog.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write (see page 3789) an AfterOpen event handler to take specific action immediately after an application opens the Open With dialog, but before the user responds to the dialog.

27.1.14.4.2 TFileOpenWith.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.14.4.3 TFileOpenWith.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
  disabling, checking and unchecking, and so on. For example,
  the following code illustrates the OnUpdate event handler
  for an action that is "checked" when the toolbar is visible.
  This example requires a TActionList (with one TAction) and a
  TToolBar on a form. This example requires a TActionList
  (with one TAction) and a TToolBar on a form.  Double click
  on the TActionList and create an TAction named Action1.  The
  TAction has an OnUpdate event handler called Action1Update.
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.15 TFilePageSetup Class

TFilePageSetup is the standard action for displaying a page setup dialog.

Class Hierarchy

Pascal

```
TFilePageSetup = class(TCommonDialogAction);
```

C++

```
class TFilePageSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePageSetup to an action list to add a common page setup dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the page setup dialog when invoked. Because the page setup dialog automatically configures the pages as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 1774) or OnCancel (see page 1774) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog

TPrintDlg (see page 1814)

27.1.15.1 TFilePageSetup Members

The following tables list the members exposed by TFilePageSetup.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1769)	This is GetDialogClass, a member of class TFilePageSetup.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1716)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1774)	Occurs immediately before the dialog appears.
	OnAccept (see page 1774)	This is OnAccept, a member of class TFilePageSetup.
	OnCancel (see page 1774)	This is OnCancel, a member of class TFilePageSetup.
	OnHint (see page 1775)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1775)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1769)	Represents the caption of client controls and menu items.
R	Dialog (see page 1770)	Provides access the page setup dialog that the action executes.
	Enabled (see page 1770)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1770)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1771)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1771)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1772)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1772)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1772)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1773)	Specifies the ShortCut property for client menu items.
	Visible (see page 1773)	Specifies the Visible property for client controls and menu items.

27.1.15.2 TFilePageSetup Methods

The methods of the TFilePageSetup class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1769)	This is GetDialogClass, a member of class TFilePageSetup.

27.1.15.2.1 TFilePageSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePageSetup.

27.1.15.3 TFilePageSetup Properties

The properties of the TFilePageSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 1769)	Represents the caption of client controls and menu items.
	Dialog (see page 1770)	Provides access the page setup dialog that the action executes.
	Enabled (see page 1770)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1770)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1771)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1771)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1772)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1772)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1772)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1773)	Specifies the ShortCut property for client menu items.
	Visible (see page 1773)	Specifies the Visible property for client controls and menu items.

27.1.15.3.1 TFilePageSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.15.3.2 TFilePageSetup.Dialog Property

Provides access the page setup dialog that the action executes.

Pascal

```
property Dialog: TPageSetupDialog;
```

C++

```
__property TPageSetupDialog Dialog;
```

Description

When using a file page setup action, you do not place a TPageSetupDialog component on the form or data module at design time. Instead, you add the TFilePageSetup (see page 1767) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The page setup dialog component is created by the TFilePageSetup (see page 1767) constructor.

See Also

Create (see page 1716)

Create

27.1.15.3.3 TFilePageSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.15.3.4 TFilePageSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value.

Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.15.3.5 TFilePageSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.15.3.6 TFilePageSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.15.3.7 TFilePageSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.15.3.8 TFilePageSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.15.3.9 TFilePageSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.15.3.10 TFilePageSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.15.3.11 TFilePageSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.15.4 TFilePageSetup Events

The events of the TFilePageSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1774)	Occurs immediately before the dialog appears.
	OnAccept (see page 1774)	This is OnAccept, a member of class TFilePageSetup.
	OnCancel (see page 1774)	This is OnCancel, a member of class TFilePageSetup.
	OnHint (see page 1775)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1775)	Occurs when the application is idle or when the action list updates.

27.1.15.4.1 TFilePageSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.15.4.2 TFilePageSetup.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFilePageSetup.

27.1.15.4.3 TFilePageSetup.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFilePageSetup.

27.1.15.4.4 TFilePageSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.15.4.5 TFilePageSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
```

```
    }
//    pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

27.1.16 TFilePrintSetup Class

TFilePrintSetup is the standard action for displaying a printer setup dialog.

Class Hierarchy

Pascal

```
TFilePrintSetup = class(TCommonDialogAction);
```

C++

```
class TFilePrintSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePrintSetup to an action list to add a common printer setup dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the printer setup dialog when invoked. Because the printer setup dialog automatically configures the printers as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 1783) or OnCancel (see page 1783) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog

TPrintDlg (see page 1814)

27.1.16.1 TFilePrintSetup Members

The following tables list the members exposed by TFilePrintSetup.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1778)	This is GetDialogClass, a member of class TFilePrintSetup.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1716)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1783)	Occurs immediately before the dialog appears.
	OnAccept (see page 1783)	This is OnAccept, a member of class TFilePrintSetup.
	OnCancel (see page 1783)	This is OnCancel, a member of class TFilePrintSetup.
	OnHint (see page 1784)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1784)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1778)	Represents the caption of client controls and menu items.
R	Dialog (see page 1779)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 1779)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1779)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1780)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1780)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1781)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1781)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1781)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1782)	Specifies the ShortCut property for client menu items.
	Visible (see page 1782)	Specifies the Visible property for client controls and menu items.

27.1.16.2 TFilePrintSetup Methods

The methods of the TFilePrintSetup class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1778)	This is GetDialogClass, a member of class TFilePrintSetup.

27.1.16.2.1 TFilePrintSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePrintSetup.

27.1.16.3 TFilePrintSetup Properties

The properties of the TFilePrintSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 1778)	Represents the caption of client controls and menu items.
R	Dialog (see page 1779)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 1779)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1779)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1780)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1780)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1781)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1781)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1781)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1782)	Specifies the ShortCut property for client menu items.
	Visible (see page 1782)	Specifies the Visible property for client controls and menu items.

27.1.16.3.1 TFilePrintSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.16.3.2 TFilePrintSetup.Dialog Property

Provides access the printer setup dialog that the action executes.

Pascal

```
property Dialog: TPrinterSetupDialog;
```

C++

```
__property TPrinterSetupDialog Dialog;
```

Description

When using a file print setup action, you do not place a TPrinterSetupDialog component on the form or data module at design time. Instead, you add the TFilePrintSetup (see page 1776) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The printer setup dialog component is created by the TFilePrintSetup (see page 1776) constructor.

See Also

Create (see page 1716)

Create_c

27.1.16.3.3 TFilePrintSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.16.3.4 TFilePrintSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.16.3.5 TFilePrintSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.16.3.6 TFilePrintSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.16.3.7 TFilePrintSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.16.3.8 TFilePrintSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.16.3.9 TFilePrintSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.16.3.10 TFilePrintSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.16.3.11 TFilePrintSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.16.4 TFilePrintSetup Events

The events of the TFilePrintSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1783)	Occurs immediately before the dialog appears.
	OnAccept (see page 1783)	This is OnAccept, a member of class TFilePrintSetup.
	OnCancel (see page 1783)	This is OnCancel, a member of class TFilePrintSetup.
	OnHint (see page 1784)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1784)	Occurs when the application is idle or when the action list updates.

27.1.16.4.1 TFilePrintSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.16.4.2 TFilePrintSetup.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFilePrintSetup.

27.1.16.4.3 TFilePrintSetup.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFilePrintSetup.

27.1.16.4.4 TFilePrintSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.16.4.5 TFilePrintSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
```

```
      {
        lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
      }
      checked = ToolBar1->Visible;
    }
//    pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

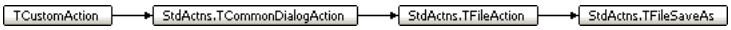
See Also

UpdateAction

27.1.17 TFileSaveAs Class

TFileSaveAs is the standard action for displaying a file save as dialog.

Class Hierarchy



Pascal

```
TFileSaveAs = class(TFileAction);
```

C++

```
class TFileSaveAs : public TFileAction;
```

File

StdActns

Description

Add TFileSaveAs to an action list to add a common file save dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a file save dialog when invoked. Use the OnAccept (see page 1792) event to respond after the user selects a file name in the file save dialog or the OnCancel (see page 1792) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog

specified by the Dialog (see page 1788) property.

See Also

TFileOpen (see page 1749)

TSaveDialog

TSavePicture

27.1.17.1 TFileSaveAs Members

The following tables list the members exposed by TFileSaveAs.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1740)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 1787)	This is GetDialogClass, a member of class TFileSaveAs.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1741)	Specifies the file name for the file selection dialog.

Public Methods

	Name	Description
V	Create (see page 1787)	Creates an instance of TFileSaveAs (see page 1785).
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1791)	Occurs immediately before the dialog appears.
	OnAccept (see page 1792)	This is OnAccept, a member of class TFileSaveAs.
	OnCancel (see page 1792)	This is OnCancel, a member of class TFileSaveAs.
	OnHint (see page 1792)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1793)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1788)	Represents the caption of client controls and menu items.
R	Dialog (see page 1788)	Provides access the save dialog that the action executes.
	Enabled (see page 1789)	Indicates whether client controls and menu items are enabled.

	HelpContext (see page 1789)	Indicates the help context ID for client controls and menu items.
	Hint (see page 1789)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1790)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1790)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1791)	Specifies the ShortCut property for client menu items.
	Visible (see page 1791)	Specifies the Visible property for client controls and menu items.

27.1.17.2 TFileSaveAs.Create Constructor

Creates an instance of TFileSaveAs (see page 1785).

Pascal

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

C++

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

Description

Most applications do not need to call the TFileSaveAs (see page 1785) constructor because actions are added at design time to an action manager or action list.

The TFileSaveAs (see page 1785) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1788) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileSaveAs (see page 1785) instance.

See Also

Dialog (see page 1788)

Owner

27.1.17.3 TFileSaveAs Methods

The methods of the TFileSaveAs class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1787)	This is GetDialogClass, a member of class TFileSaveAs.

27.1.17.3.1 TFileSaveAs.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileSaveAs.

27.1.17.4 TFileSaveAs Properties

The properties of the TFileSaveAs class are listed here.

Published Properties

	Name	Description
	Caption (see page 1788)	Represents the caption of client controls and menu items.
R	Dialog (see page 1788)	Provides access the save dialog that the action executes.
	Enabled (see page 1789)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1789)	Indicates the help context ID for client controls and menu items.
	Hint (see page 1789)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1790)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1790)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1791)	Specifies the ShortCut property for client menu items.
	Visible (see page 1791)	Specifies the Visible property for client controls and menu items.

27.1.17.4.1 TFileSaveAs.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.17.4.2 TFileSaveAs.Dialog Property

Provides access the save dialog that the action executes.

Pascal

```
property Dialog: TSaveDialog;
```

C++

```
__property TSaveDialog Dialog;
```

Description

Use Dialog to configure the save as dialog before the action fires. For example, you can set the save as dialog's default extension at design time using the Object Inspector. Dialog is also useful for reading the selected file name from an OnAccept (see page 1792) event handler.

When using a file save as action, you do not place a TSaveDialog component on the form or data module at design time. Instead, you add the TFileSaveAs (see page 1785) to your action list and then use the Dialog property. The save dialog component is created by the TFileSaveAs (see page 1785) constructor.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

27.1.17.4.3 TFileSaveAs.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.17.4.4 TFileSaveAs.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.17.4.5 TFileSaveAs.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.17.4.6 TFileSaveAs.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.17.4.7 TFileSaveAs.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.17.4.8 TFileSaveAs.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.17.4.9 TFileSaveAs.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.17.5 TFileSaveAs Events

The events of the TFileSaveAs class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1791)	Occurs immediately before the dialog appears.
	OnAccept (see page 1792)	This is OnAccept, a member of class TFileSaveAs.
	OnCancel (see page 1792)	This is OnCancel, a member of class TFileSaveAs.
	OnHint (see page 1792)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1793)	Occurs when the application is idle or when the action list updates.

27.1.17.5.1 TFileSaveAs.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.17.5.2 TFileSaveAs.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFileSaveAs.

27.1.17.5.3 TFileSaveAs.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFileSaveAs.

27.1.17.5.4 TFileSaveAs.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.17.5.5 TFileSaveAs.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.18 TFontEdit Class

TFontEdit is the standard action for displaying a font selection dialog.

Class Hierarchy

Pascal

```
TFontEdit = class(TCommonDialogAction);
```

C++

```
class TFontEdit : public TCommonDialogAction;
```

File

StdActns

Description

Add TFontEdit to an action list to add a modal font selection dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a font dialog when invoked. Use the OnAccept (see page 1801) event to respond after the user selects a font in the dialog or the OnCancel (see page 1801) event to respond when the user cancels from the dialog. In the event handler, you can read the selected font from the dialog specified by the Dialog (see page 1796) property.

See Also

TFontDialog

27.1.18.1 TFontEdit Members

The following tables list the members exposed by TFontEdit.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.

	GetDialogClass (see page 1795)	This is GetDialogClass, a member of class TFontEdit.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1801)	Occurs immediately before the dialog appears.
	OnAccept (see page 1801)	This is OnAccept, a member of class TFontEdit.
	OnCancel (see page 1801)	This is OnCancel, a member of class TFontEdit.
	OnHint (see page 1801)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1802)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1796)	Represents the caption of client controls and menu items.
R	Dialog (see page 1796)	Provides access the font selection dialog that the action executes.
	Enabled (see page 1797)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1797)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1798)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1798)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1799)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1799)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1799)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1800)	Specifies the ShortCut property for client menu items.
	Visible (see page 1800)	Specifies the Visible property for client controls and menu items.

27.1.18.2 TFontEdit Methods

The methods of the TFontEdit class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1795)	This is GetDialogClass, a member of class TFontEdit.

27.1.18.2.1 TFontEdit.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFontEdit.

27.1.18.3 TFontEdit Properties

The properties of the TFontEdit class are listed here.

Published Properties

	Name	Description
	Caption (see page 1796)	Represents the caption of client controls and menu items.
R	Dialog (see page 1796)	Provides access the font selection dialog that the action executes.
	Enabled (see page 1797)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1797)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1798)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1798)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1799)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1799)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1799)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1800)	Specifies the ShortCut property for client menu items.
	Visible (see page 1800)	Specifies the Visible property for client controls and menu items.

27.1.18.3.1 TFontEdit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.18.3.2 TFontEdit.Dialog Property

Provides access the font selection dialog that the action executes.

Pascal

```
property Dialog: TFontDialog;
```

C++

```
__property TFontDialog Dialog;
```

Description

Use Dialog to configure the font selection dialog before the action fires or to read the selected font after the action executes the

dialog. For example, you can set the font dialog's target device or limit the available font sizes at design time using the Object Inspector or read the Font property in an OnAccept (see page 1801) event handler to make use of the font that the user selects.

When using a font edit action, you do not place a TFontDialog component on the form or data module at design time. Instead, you add the TFontEdit (see page 1794) to your action list and then use the Dialog property. The font selection dialog component is created by the TFontEdit (see page 1794) constructor.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Font

27.1.18.3.3 TFontEdit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.18.3.4 TFontEdit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.18.3.5 TFontEdit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.18.3.6 TFontEdit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.18.3.7 TFontEdit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.18.3.8 TFontEdit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.18.3.9 TFontEdit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.18.3.10 TFontEdit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.18.3.11 TFontEdit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.18.4 TFontEdit Events

The events of the TFontEdit class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1801)	Occurs immediately before the dialog appears.
	OnAccept (see page 1801)	This is OnAccept, a member of class TFontEdit.
	OnCancel (see page 1801)	This is OnCancel, a member of class TFontEdit.
	OnHint (see page 1801)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1802)	Occurs when the application is idle or when the action list updates.

27.1.18.4.1 TFontEdit.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.18.4.2 TFontEdit.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFontEdit.

27.1.18.4.3 TFontEdit.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFontEdit.

27.1.18.4.4 TFontEdit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.18.4.5 TFontEdit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
```

```
  for an action that is "checked" when the toolbar is visible.
  This example requires a TActionList (with one TAction) and a
  TToolBar on a form. This example requires a TActionList
  (with one TAction) and a TToolBar on a form.  Double click
  on the TActionList and create an TAction named Action1.  The
  TAction has an OnUpdate event handler called Action1Update.
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.19 THelpAction Class

THelpAction is the base class for action objects that interact with the help system.

Class Hierarchy

Pascal

```
THelpAction = class(TAction);
```

C++

```
class THelpAction : public TAction;
```

File

StdActns

Description

THelpAction implements the common behavior for actions that work the help system. Each descendant class performs a specific help function. Do not use THelpAction directly in an application. Instead, use a THelpAction descendant such as THelpContents (see page 1805), THelpContextAction (see page 1806), THelpOnHelp (see page 1808), or THelpTopicSearch (see page 1810).

THelpAction ensures that the action is enabled as long as there is an application object to pass commands on to a registered help system. THelpAction descendants override the ExecuteTarget method to execute their specific help command.

See Also

THelpContents (see page 1805)

THelpContextAction (see page 1806)

THelpOnHelp (see page 1808)

THelpTopicSearch (see page 1810)

TApplication

27.1.19.1 THelpAction Members

The following tables list the members exposed by THelpAction.

Public Methods

	Name	Description
	Create (see page 1804)	Creates and initializes a TAction object.
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the availability of the Application object.

27.1.19.2 THelpAction.Create Constructor

Creates and initializes a TAction object.

Pascal

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

C++

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

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList

27.1.19.3 THelpAction Methods

The methods of the THelpAction class are listed here.

Public Methods

	Name	Description
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the availability of the Application object.

27.1.19.3.1 THelpAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool

button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in THelpAction (see page 1803), HandlesTarget ignores the Target parameter and always returns true, because the help actions do not rely on a specific target object.

27.1.19.3.2 THelpAction.UpdateTarget Method

Enables or disables the action, depending on the availability of the Application object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpAction (see page 1803), UpdateTarget checks whether the global Application variable is assigned, because this object is required to pass help commands on to WinHelp HyperHelp. If Application is assigned, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

Application

27.1.20 THelpContents Class

THelpContents brings up the contents for the default help system.

Class Hierarchy

Pascal

```
THelpContents = class(THelpAction);
```

C++

```
class THelpContents : public THelpAction;
```

File

StdActns

Description

Add THelpContents to an action list to allow users to bring up the help contents by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to show the help contents.

On WinHelp, THelpContents displays the Help Topics dialog on the tab (Contents, Index or Find) that was last used.

See Also

THelpContextAction (see page 1806)

THelpOnHelp (see page 1808)

THelpTopicSearch (see page 1810)

27.1.20.1 THelpContents Members

The following tables list the members exposed by THelpContents.

Public Methods

	Name	Description
	Create (see page 1804)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1806)	Displays a dialog that lists the help topics.
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the availability of the Application object.

27.1.20.2 THelpContents Methods

The methods of the THelpContents class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1806)	Displays a dialog that lists the help topics.

27.1.20.2.1 THelpContents.ExecuteTarget Method

Displays a dialog that lists the help topics.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to list the help topics.

27.1.21 THelpContextAction Class

THelpContextAction brings up the help topic for the active control.

Class Hierarchy

Pascal

```
THelpContextAction = class(THelpAction);
```

C++

```
class THelpContextAction : public THelpAction;
```

File

StdActns

Description

Add THelpContextAction to an action list to allow users to bring up the help for an active control. Controls such as menu items and tool buttons linked to this action cause the application to send the help engine a command to display the help topic specified by the active control's HelpContext property.

See Also

THelpContents (see page 1805)

THelpTopicSearch (see page 1810)

THelpOnHelp (see page 1808)

TControl

TScreen

27.1.21.1 THelpContextAction Members

The following tables list the members exposed by THelpContextAction.

Public Methods

	Name	Description
	Create (see page 1804)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1807)	Displays the help topic associated with the active control's HelpContext property.
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1808)	Enables or disables the action, depending on whether there is an active object.

27.1.21.2 THelpContextAction Methods

The methods of the THelpContextAction class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1807)	Displays the help topic associated with the active control's HelpContext property.
	UpdateTarget (see page 1808)	Enables or disables the action, depending on whether there is an active object.

27.1.21.2.1 THelpContextAction.ExecuteTarget Method

Displays the help topic associated with the active control's HelpContext property.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the topic specified by the active control's HelpContext property. ExecuteTarget gets the active control from the global Screen object rather than using the Target parameter.

See Also

HelpCommand

ActiveControl

HelpContext

27.1.21.2.2 THelpContextAction.UpdateTarget Method

Enables or disables the action, depending on whether there is an active object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpContextAction (see page 1806), UpdateTarget checks whether the global Screen object's ActiveConrol property is assigned. If there is an active control, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

ActiveControl

27.1.22 THelpOnHelp Class

THelpOnHelp brings up a help topic on how to use Help.

Class Hierarchy

Pascal

```
THelpOnHelp = class(THelpAction);
```

C++

```
class THelpOnHelp : public THelpAction;
```

File

StdActns

Description

Add THelpOnHelp to an action list to allow users to bring up the help system that describes how to use help. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display its help.

Note that on recent versions of Windows with a default help system of WinHelp, this brings up an HTML help file that does not describe the WinHelp system.

See Also

THelpContents (see page 1805)

THelpContextAction (see page 1806)

THelpTopicSearch (see page 1810)

27.1.22.1 THelpOnHelp Members

The following tables list the members exposed by THelpOnHelp.

Public Methods

	Name	Description
	Create (see page 1804)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1809)	Brings up the help system's help file.
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the availability of the Application object.

27.1.22.2 THelpOnHelp Methods

The methods of the THelpOnHelp class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1809)	Brings up the help system's help file.

27.1.22.2.1 THelpOnHelp.ExecuteTarget Method

Brings up the help system's help file.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a

menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the Help system's own help file.

See Also

HelpCommand

27.1.23 THelpTopicSearch Class

THelpTopicSearch brings up the help index.

Class Hierarchy

Pascal

```
THelpTopicSearch = class(THelpAction);
```

C++

```
class THelpTopicSearch : public THelpAction;
```

File

StdActns

Description

Add THelpTopicSearch to an action list to allow users to bring up the help index. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display the help index.

See Also

THelpTopicSearch

THelpContextAction (see page 1806)

THelpOnHelp (see page 1808)

THelpTopicSearch

27.1.23.1 THelpTopicSearch Members

The following tables list the members exposed by THelpTopicSearch.

Public Methods

	Name	Description
	Create (see page 1804)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1811)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 1804)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the availability of the Application object.

27.1.23.2 THelpTopicSearch Methods

The methods of the THelpTopicSearch class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1811)	Introduces an interface for invoking an action on a target client component or control.

27.1.23.2.1 THelpTopicSearch.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

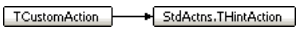
OnExecute

HandlesTarget

27.1.24 THintAction Class

THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.

Class Hierarchy



Pascal

```
THintAction = class(TCustomAction);
```

C++

```
class THintAction : public TCustomAction;
```

File

StdActns

Description

Do not use THintAction in an action manager or action list the way you would use other predefined actions. THintAction is not intended to work with a client control. Rather, it is used by the application object.

When the user pauses the mouse over a control or menu item that defines its Hint (see page 1813) property, the application responds in one of two ways:

If the application has an OnHint event handler, it generates an OnHint event.

If there is no OnHint event handler, the application executes a THintAction object.

When THintAction executes, it calls the ExecuteAction method of potential target controls until a control is found to respond to the action. Some controls, such as TStatusBar, override the ExecuteAction method to respond to THintAction.

Component writers can override the ExecuteAction method of a new component to respond to THintAction by displaying hint messages. Such an ExecuteAction method might look like the following:

```
function TMyControl.ExecuteAction(Action: TBasicAction): Boolean;
var
  HintText: string;
begin
if Action is THintAction then
begin
    HintText := THintAction(Action).Hint;
    { do something to display the hint text }
    Result := True;
end
else Result := inherited ExecuteAction(Action);
end;
bool __fastcall TMyControl::ExecuteAction(TBasicAction *Action)
{
  THintAction *pHint = dynamic_cast<THintAction *>(Action);
if (pHint)
  {
    // do something to display pHint->Hint
returntrue;
  }
return TComponent::ExecuteAction(Action);
}
```

See Also

TControl

TMenuItem

TComponent

TApplication

27.1.24.1 THintAction Members

The following tables list the members exposed by THintAction.

Public Methods

	Name	Description
	Create (see page 1812)	Creates an instance of THintAction (see page 1811).

Published Properties

	Name	Description
	Hint (see page 1813)	Indicates the Help hint for client controls and menu items.

27.1.24.2 THintAction.Create Constructor

Creates an instance of THintAction (see page 1811).

Pascal

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

C++

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

Description

Typically, you do not need to create THintAction (see page 1811) instances. THintAction (see page 1811) is created by the application object when it detects the need to display a hint.

You can, however, create a THintAction (see page 1811) object, set its Hint (see page 1813) property, and then call its Execute method to cause other controls in the application (such as status bars) to display a message for some application-generated condition (such as a status message).

AOwner is the component, typically the application object, that is responsible for freeing the THintAction (see page 1811) object.

See Also

Owner

Hint

Execute

27.1.24.3 THintAction Properties

The properties of the THintAction class are listed here.

Published Properties

	Name	Description
	Hint (see page 1813)	Indicates the Help hint for client controls and menu items.

27.1.24.3.1 THintAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.25 TPrintDlg Class

TPrintDlg is the standard action for displaying a print dialog.

Class Hierarchy

Pascal

```
TPrintDlg = class(TCommonDialogAction);
```

C++

```
class TPrintDlg : public TCommonDialogAction;
```

File

StdActns

Description

Add TPrintDlg to an action list to add a print dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the print dialog (TPrintDialog) when invoked. Write (see page 3789) an OnAccept (see page 1821) event handler to perform the actual printing when the user clicks OK. You can read details about the user's selections in the dialog from the Dialog (see page 1816) property.

See Also

TPrintDialog

TFilePrintSetup (see page 1776)

27.1.25.1 TPrintDlg Members

The following tables list the members exposed by TPrintDlg.

Protected Fields

	Name	Description
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1815)	This is GetDialogClass, a member of class TPrintDlg.
	Notification (see page 1720)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1821)	Occurs immediately before the dialog appears.
	OnAccept (see page 1821)	This is OnAccept, a member of class TPrintDlg.
	OnCancel (see page 1821)	This is OnCancel, a member of class TPrintDlg.
	OnHint (see page 1821)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1822)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1816)	Represents the caption of client controls and menu items.
	Dialog (see page 1816)	Provides access the print dialog that the action executes.
	Enabled (see page 1817)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1817)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1818)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1818)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1819)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1819)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1819)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1820)	Specifies the ShortCut property for client menu items.
	Visible (see page 1820)	Specifies the Visible property for client controls and menu items.

27.1.25.2 TPrintDlg Methods

The methods of the TPrintDlg class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1815)	This is GetDialogClass, a member of class TPrintDlg.

27.1.25.2.1 TPrintDlg.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TPrintDlg.

27.1.25.3 TPrintDlg Properties

The properties of the TPrintDlg class are listed here.

Published Properties

	Name	Description
	Caption (see page 1816)	Represents the caption of client controls and menu items.
R	Dialog (see page 1816)	Provides access the print dialog that the action executes.
	Enabled (see page 1817)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1817)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1818)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1818)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1819)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1819)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1819)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1820)	Specifies the ShortCut property for client menu items.
	Visible (see page 1820)	Specifies the Visible property for client controls and menu items.

27.1.25.3.1 TPrintDlg.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.25.3.2 TPrintDlg.Dialog Property

Provides access the print dialog that the action executes.

Pascal

```
property Dialog: TPrintDialog;
```

C++

```
__property TPrintDialog Dialog;
```

Description

When using a print dialog action, you do not place a TPrintDialog component on the form or data module at design time. Instead, you add the TPrintDlg (see page 1814) action to your action list. If you want to configure the dialog, for example to set the Options that are available to users, you can access the dialog component using the Dialog property. From an OnAccept (see page 1821) event handler, use Dialog to read the choices that the user made in the print dialog and then perform the printing based on those choices.

The print dialog component is created by the TPrintDlg (see page 1814) constructor.

See Also

Create_d

Create_c

Create

Options

onaccept (see page 1721)

27.1.25.3.3 TPrintDlg.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.25.3.4 TPrintDlg.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.25.3.5 TPrintDlg.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.25.3.6 TPrintDlg.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.25.3.7 TPrintDlg.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.25.3.8 TPrintDlg.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.25.3.9 TPrintDlg.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.25.3.10 TPrintDlg.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.25.3.11 TPrintDlg.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.25.4 TPrintDlg Events

The events of the TPrintDlg class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1821)	Occurs immediately before the dialog appears.
	OnAccept (see page 1821)	This is OnAccept, a member of class TPrintDlg.
	OnCancel (see page 1821)	This is OnCancel, a member of class TPrintDlg.
	OnHint (see page 1821)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1822)	Occurs when the application is idle or when the action list updates.

27.1.25.4.1 TPrintDlg.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.25.4.2 TPrintDlg.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TPrintDlg.

27.1.25.4.3 TPrintDlg.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TPrintDlg.

27.1.25.4.4 TPrintDlg.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.25.4.5 TPrintDlg.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
```

```
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

27.1.26 TSearchAction Class

TSearchAction is the base class for action objects that display a search or search and replace dialog.

Class Hierarchy

Pascal

```
TSearchAction = class(TCommonDialogAction);
```

C++

```
class TSearchAction : public TCommonDialogAction;
```

File

StdActns

Description

TSearchAction implements the common behavior for actions that display a modeless dialog where the user can enter a search string for searching an edit control. Do not use TSearchAction directly in an application. Instead, use one of the TSearchAction descendants: TSearchFind (see page 1828) or TSearchFindFirst (see page 1838) for a simple locate dialog, TSearchReplace (see page 1850) for a locate-and-replace dialog.

TSearchAction descendants are designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. They automatically disable themselves if the active control in a form is not an edit control.

Unlike other dialog actions, search actions represent a modeless dialog. As a result, the OnAccept (see page 1721) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for or replace a string, the search action automatically performs that search, selecting or replacing text in the target edit control.

See Also

TSearchFind (see page 1828)

TSearchReplace (see page 1850)

TSearchFindNext (see page 1840)

TCustomEdit (see page 2263)

TSearchFindFirst (see page 1838)

27.1.26.1 TSearchAction Members

The following tables list the members exposed by TSearchAction.

Protected Fields

	Name	Description
	FControl (see page 1825)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1826)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1827)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1721)	Occurs immediately before the dialog appears.
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1824)	Creates and initializes an instance of TSearchAction (see page 1823).
	Destroy (see page 1825)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1826)	Launches the associated dialog.
	HandlesTarget (see page 1827)	Indicates whether the action can execute its dialog.
	Search (see page 1827)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1828)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

27.1.26.2 TSearchAction.Create Constructor

Creates and initializes an instance of TSearchAction (see page 1823).

Pascal

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

C++

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

Description

Most applications do not need to call the TSearchAction (see page 1823) constructor because actions are added at design

time to action lists or action managers.

The TSearchAction (see page 1823) constructor creates an instance of its associated dialog, which descendant classes surface as the Dialog property. It then assigns the Search (see page 1827) method as an event handler for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchAction (see page 1823) instance.

See Also

Dialog (see page 1831)

Dialog (see page 1853)

Owner

Search (see page 1827)

OnFind

27.1.26.3 TSearchAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSearchAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

27.1.26.4 TSearchAction Fields

The fields of the TSearchAction class are listed here.

Protected Fields

	Name	Description
	FControl (see page 1825)	This is FControl, a member of class TSearchAction.
	FFindFirst (see page 1826)	This is FFindFirst, a member of class TSearchAction.

27.1.26.4.1 TSearchAction.FControl Field

Pascal

```
FControl: TCustomEdit;
```

C++

```
TCustomEdit FControl;
```

Description

This is FControl, a member of class TSearchAction.

27.1.26.4.2 TSearchAction.FFindFirst Field

Pascal

```
FFindFirst: Boolean;
```

C++

```
Boolean FFindFirst;
```

Description

This is FFindFirst, a member of class TSearchAction.

27.1.26.5 TSearchAction Methods

The methods of the TSearchAction class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1827)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	ExecuteTarget (see page 1826)	Launches the associated dialog.
	HandlesTarget (see page 1827)	Indicates whether the action can execute its dialog.
	Search (see page 1827)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1828)	Enables or disables the action, depending on the currently active control in the form.

27.1.26.5.1 TSearchAction.ExecuteTarget Method

Launches the associated dialog.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches its search or search and replace dialog. This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 1720) property, and an OnAccept (see page 1721) or OnCancel (see page 1722) event occurs, depending on whether the dialog was brought into focus.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Execute

ExecuteResult (see page 1720)

27.1.26.5.2 TSearchAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget returns true if the target is an edit control that has keyboard focus.

27.1.26.5.3 TSearchAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

27.1.26.5.4 TSearchAction.Search Method

Selects the next occurrence of the search string in the target edit control.

Pascal

```
procedure Search(Sender: TObject); virtual;
```

C++

```
virtual __fastcall Search(TObject * Sender);
```

Description

Most applications have no need to call the Search method: the TSearchAction (see page 1823) constructor automatically assigns Search as the OnFind event handler of its associated dialog. The dialog's OnFind event occurs when the user directs the dialog to find a specified string.

Search searches the target edit control for the string that is the FindText property of the search dialog, using the options specified by the dialog's Options property. If it finds the specified string, it selects it in the edit control. Otherwise, it displays a message to the user indicating that the string was not found.

Note: If the search action is an instance of TSearchFind (see page 1828), Search looks for the first occurrence of FindText after the current selection in the edit control. If the search action is an instance of TSearchFindFirst (see page 1838), Search looks for the first occurrence of FindText, regardless of the current selection.

See Also

OnFind

Options

FindText

27.1.26.5.5 TSearchAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TSearchAction (see page 1823), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

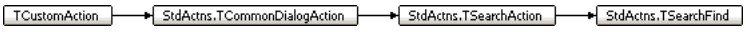
See Also

Enabled

27.1.27 TSearchFind Class

TSearchFind is the standard action for displaying a find dialog.

Class Hierarchy



Pascal

```
TSearchFind = class(TSearchAction);
```

C++

```
class TSearchFind : public TSearchAction;
```

File

StdActns

Description

Add TSearchFind to an action list to add a modeless find dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFind is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1827) actions represent a modeless dialog. As a result, the OnAccept (see page 1836) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFind automatically performs that search, selecting the specified text in the target edit control.

See Also

TFindDialog

TSearchReplace (see page 1850)

TSearchFindNext (see page 1840)

TCustomEdit (see page 2263)

TSearchFindFirst (see page 1838)

27.1.27.1 TSearchFind Members

The following tables list the members exposed by TSearchFind.

Protected Fields

	Name	Description
	FControl (see page 1825)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1826)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1830)	This is GetDialogClass, a member of class TSearchFind.
	Notification (see page 1720)	Responds when components are created or destroyed.
	Notification (see page 1827)	Responds when components are created or destroyed.
	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1716)	Creates an instance of TCommonDialogAction.
	Create (see page 1824)	Creates and initializes an instance of TSearchAction (see page 1823).
	Destroy (see page 1717)	Frees an instance of TCustomAction.
	Destroy (see page 1825)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
	ExecuteTarget (see page 1826)	Launches the associated dialog.
	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 1827)	Indicates whether the action can execute its dialog.

	Search (see page 1827)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1828)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1835)	Occurs immediately before the dialog appears.
	OnAccept (see page 1836)	This is OnAccept, a member of class TSearchFind.
	OnCancel (see page 1836)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 1836)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1837)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1831)	Represents the caption of client controls and menu items.
	Dialog (see page 1831)	Provides access the find dialog that the action executes.
	Enabled (see page 1832)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1832)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1832)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1833)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1833)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1834)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1834)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1835)	Specifies the ShortCut property for client menu items.
	Visible (see page 1835)	Specifies the Visible property for client controls and menu items.

27.1.27.2 TSearchFind Methods

The methods of the TSearchFind class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1830)	This is GetDialogClass, a member of class TSearchFind.

27.1.27.2.1 TSearchFind.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchFind.

27.1.27.3 TSearchFind Properties

The properties of the TSearchFind class are listed here.

Published Properties

	Name	Description
	Caption (see page 1831)	Represents the caption of client controls and menu items.
R	Dialog (see page 1831)	Provides access the find dialog that the action executes.
	Enabled (see page 1832)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1832)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1832)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1833)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1833)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1834)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1834)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1835)	Specifies the ShortCut property for client menu items.
	Visible (see page 1835)	Specifies the Visible property for client controls and menu items.

27.1.27.3.1 TSearchFind.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.27.3.2 TSearchFind.Dialog Property

Provides access the find dialog that the action executes.

Pascal

```
property Dialog: TFindDialog;
```

C++

```
__property TFindDialog Dialog;
```

Description

Use Dialog to configure the find dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the find dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to close it.

When using a search-find action, you do not place a TFindDialog component on the form or data module at design time. Instead, you add the TSearchFind (see page 1828) action to your action list and then use the Dialog property. The find dialog

component is created by the TSearchFind (see page 1828) constructor.

See Also

ExecuteTarget (see page 1826)

CloseDialog

27.1.27.3.3 TSearchFind.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.27.3.4 TSearchFind.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.27.3.5 TSearchFind.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.27.3.6 TSearchFind.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.27.3.7 TSearchFind.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.27.3.8 TSearchFind.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.27.3.9 TSearchFind.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.27.3.10 TSearchFind.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.27.3.11 TSearchFind.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.27.4 TSearchFind Events

The events of the TSearchFind class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1835)	Occurs immediately before the dialog appears.
	OnAccept (see page 1836)	This is OnAccept, a member of class TSearchFind.
	OnCancel (see page 1836)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 1836)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1837)	Occurs when the application is idle or when the action list updates.

27.1.27.4.1 TSearchFind.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.27.4.2 TSearchFind.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TSearchFind.

27.1.27.4.3 TSearchFind.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TSearchFind.

27.1.27.4.4 TSearchFind.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.27.4.5 TSearchFind.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.28 TSearchFindFirst Class

TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.

Class Hierarchy

Pascal

```
TSearchFindFirst = class(TSearchFind);
```

C++

```
class TSearchFindFirst : public TSearchFind;
```

File

StdActns

Description

Add TSearchFindFirst to an action list to add a modeless find dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFindFirst is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1827) actions represent a modeless dialog. As a result, the OnAccept (see page 1836) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFindFirst automatically performs that search, selecting the specified text in the target edit control. Unlike the TSearchFind (see page 1828) action, which locates the first occurrence of the search string following the cursor position in the edit control, TSearchFindFirst locates the first occurrence of the search string, regardless of the cursor position.

See Also

TFindDialog

TSearchReplace (see page 1850)

TSearchFindNext (see page 1840)

TCustomEdit (see page 2263)

TSearchFind (see page 1828)

27.1.28.1 TSearchFindFirst Members

The following tables list the members exposed by TSearchFindFirst.

Protected Fields

	Name	Description
	FControl (see page 1825)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1826)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1719)	This is GetDialogClass, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1830)	This is GetDialogClass, a member of class TSearchFind.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	Notification (see page 1827)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1721)	Occurs immediately before the dialog appears.
	OnAccept (see page 1721)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 1722)	This is OnCancel, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1840)	Creates and initializes an instance of TSearchFindFirst (see page 1838).
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	Destroy (see page 1825)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1718)	Executes the associated dialog and generates an OnAccept (see page 1721) or OnCancel (see page 1722) event.
V	ExecuteTarget (see page 1826)	Launches the associated dialog.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.
V	HandlesTarget (see page 1827)	Indicates whether the action can execute its dialog.
V	Search (see page 1827)	Selects the next occurrence of the search string in the target edit control.
V	UpdateTarget (see page 1828)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1835)	Occurs immediately before the dialog appears.
	OnAccept (see page 1836)	This is OnAccept, a member of class TSearchFind.
	OnCancel (see page 1836)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 1836)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1837)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1831)	Represents the caption of client controls and menu items.
R	Dialog (see page 1831)	Provides access the find dialog that the action executes.
	Enabled (see page 1832)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1832)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1832)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1833)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1833)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1834)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1834)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1835)	Specifies the ShortCut property for client menu items.
	Visible (see page 1835)	Specifies the Visible property for client controls and menu items.

27.1.28.2 TSearchFindFirst.Create Constructor

Creates and initializes an instance of TSearchFindFirst (see page 1838).

Pascal

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

C++

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

Description

Most applications do not need to call the TSearchFindFirst (see page 1838) constructor because actions are added at design time to an action manager or action list.

The TSearchFindFirst (see page 1838) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1831) property. It then assigns the Search (see page 1827) method as an event handler for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindFirst (see page 1838) instance.

See Also

Dialog (see page 1831)

Owner

Search (see page 1827)

OnFind

27.1.29 TSearchFindNext Class

TSearchFindNext is the standard action for locating the next instance of a string in an edit control.

Class Hierarchy

Pascal

```
TSearchFindNext = class(TCustomAction);
```

C++

```
class TSearchFindNext : public TCustomAction;
```

File

StdActns

Description

Add TSearchFindNext to an action list to add the ability to repeat a search initiated by a TSearchFind (see page 1828) or TSearchFindFirst (see page 1838) action. Controls such as menu items and tool buttons linked to this action cause the application to repeat the search initiated by the TSearchFind (see page 1828) action's dialog.

TSearchFindNext objects must be associated with a TSearchFind (see page 1828) action by setting the SearchFind (see page 1847) property.

See Also

TSearchFind (see page 1828)

TSearchFindFirst (see page 1838)

27.1.29.1 TSearchFindNext Members

The following tables list the members exposed by TSearchFindNext.

Public Methods

	Name	Description
	Create (see page 1842)	Creates an instance of TSearchFindNext (see page 1840).
	ExecuteTarget (see page 1842)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 1843)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	UpdateTarget (see page 1843)	Enables or disables the action, depending on the currently active control in the form.

Published Events

	Name	Description
	OnHint (see page 1848)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1849)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1844)	Represents the caption of client controls and menu items.
	Enabled (see page 1844)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1844)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1845)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1845)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1846)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1846)	Indicates the ImageIndex property for client controls and menu items.
	SearchFind (see page 1847)	Associates the TSearchFindNext (see page 1840) action with a TSearchFind (see page 1828) action that initiates the search.
	SecondaryShortCuts (see page 1847)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1847)	Specifies the ShortCut property for client menu items.

	Visible (see page 1848)	Specifies the Visible property for client controls and menu items.

27.1.29.2 TSearchFindNext.Create Constructor

Creates an instance of TSearchFindNext (see page 1840).

Pascal

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

C++

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

Description

Most applications do not need to call the TSearchFindNext (see page 1840) constructor because actions are added at design time to an action manager or action list.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindNext (see page 1840) instance.

See Also

Owner

Actions

27.1.29.3 TSearchFindNext Methods

The methods of the TSearchFindNext class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1842)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 1843)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	UpdateTarget (see page 1843)	Enables or disables the action, depending on the currently active control in the form.

27.1.29.3.1 TSearchFindNext.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute

HandlesTarget

27.1.29.3.2 TSearchFindNext.HandlesTarget Method

Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

HandlesTarget does nothing in TBasicAction. HandlesTarget was introduced in TBasicAction so that descendants can override it to check the type and state of a target. Handles target can verify any information about a target that is relevant for the action. HandlesTarget returns true if the target meets the specified criteria, false otherwise.

See Also

ExecuteTarget

27.1.29.3.3 TSearchFindNext.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TSearchFindNext (see page 1840), UpdateTarget ignores the current target object (specified by the Target parameter). Instead, it checks the associated TSearchFind (see page 1828) action (the SearchFind (see page 1847) property), and sets Enabled (see page 1844) to true if the search dialog has already specified a search string. If SearchFind (see page 1847) is not set, or if there is no search string in its associated dialog, UpdateTarget sets Enabled (see page 1844) to false.

See Also

Enabled

SearchFind (see page 1847)

FindText

27.1.29.4 TSearchFindNext Properties

The properties of the TSearchFindNext class are listed here.

Published Properties

	Name	Description
	Caption (see page 1844)	Represents the caption of client controls and menu items.

	Enabled (see page 1844)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1844)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1845)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1845)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1846)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1846)	Indicates the ImageIndex property for client controls and menu items.
	SearchFind (see page 1847)	Associates the TSearchFindNext (see page 1840) action with a TSearchFind (see page 1828) action that initiates the search.
	SecondaryShortCuts (see page 1847)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1847)	Specifies the ShortCut property for client menu items.
	Visible (see page 1848)	Specifies the Visible property for client controls and menu items.

27.1.29.4.1 TSearchFindNext.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.29.4.2 TSearchFindNext.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.29.4.3 TSearchFindNext.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.29.4.4 TSearchFindNext.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.29.4.5 TSearchFindNext.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.29.4.6 TSearchFindNext.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.29.4.7 TSearchFindNext.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.29.4.8 TSearchFindNext.SearchFind Property

Associates the TSearchFindNext (see page 1840) action with a TSearchFind (see page 1828) action that initiates the search.

Pascal

```
property SearchFind: TSearchFind;
```

C++

```
__property TSearchFind SearchFind;
```

Description

SearchFind is the TSearchFind (see page 1828) action that displays the modeless search dialog whose current search should be repeated. If SearchFind is not set, then the TSearchFindNext (see page 1840) action remains disabled.

TSearchFindNext (see page 1840) uses the SearchFind property to obtain a search string from its associated dialog and to search its associated control.

See Also

executetarget (see page 1842)

Enabled

Search (see page 1827)

27.1.29.4.9 TSearchFindNext.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.29.4.10 TSearchFindNext.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.29.4.11 TSearchFindNext.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.29.5 TSearchFindNext Events

The events of the TSearchFindNext class are listed here.

Published Events

	Name	Description
	OnHint (see page 1848)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1849)	Occurs when the application is idle or when the action list updates.

27.1.29.5.1 TSearchFindNext.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.29.5.2 TSearchFindNext.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

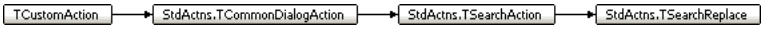
See Also

UpdateAction

27.1.30 TSearchReplace Class

TSearchReplace is the standard action for displaying a find-and-replace dialog.

Class Hierarchy



Pascal

```
TSearchReplace = class(TSearchAction);
```

C++

```
class TSearchReplace : public TSearchAction;
```

File

StdActns

Description

Add TSearchReplace to an action list to add a modeless find-and-replace dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find-and-replace dialog or bring it to the fore.

TSearchReplace is designed to work with an edit control (TCustomEdit (see page 2263) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1827)/replace actions represent a modeless dialog. As a result, the OnAccept (see page 1858) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to replace one or all instances of a string, TSearchReplace automatically performs the search and replaces the specified text in the target edit control.

See Also

TReplaceDialog

TSearchFind (see page 1828)

TCustomEdit (see page 2263)

TSearchFindFirst (see page 1838)

27.1.30.1 TSearchReplace Members

The following tables list the members exposed by TSearchReplace.

Protected Fields

	Name	Description
	FControl (see page 1825)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1717)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1826)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1718)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1718)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1852)	This is GetDialogClass, a member of class TSearchReplace.
V	Notification (see page 1720)	Responds when components are created or destroyed.
V	Notification (see page 1827)	Responds when components are created or destroyed.
V	SetupDialog (see page 1720)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1716)	Creates an instance of TCommonDialogAction.
V	Create (see page 1824)	Creates and initializes an instance of TSearchAction (see page 1823).
V	Destroy (see page 1717)	Frees an instance of TCustomAction.
V	Destroy (see page 1825)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1852)	Launches the associated dialog and assigns an event handler to replace found text.
V	HandlesTarget (see page 1719)	Indicates whether the action can execute its dialog.
V	HandlesTarget (see page 1827)	Indicates whether the action can execute its dialog.
V	Search (see page 1827)	Selects the next occurrence of the search string in the target edit control.
V	UpdateTarget (see page 1828)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
R	ExecuteResult (see page 1720)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1858)	Occurs immediately before the dialog appears.
	OnAccept (see page 1858)	This is OnAccept, a member of class TSearchReplace.
	OnCancel (see page 1858)	This is OnCancel, a member of class TSearchReplace.
	OnHint (see page 1858)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1859)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1853)	Represents the caption of client controls and menu items.
R	Dialog (see page 1853)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 1854)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1854)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1855)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1855)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1856)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1856)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1856)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1857)	Specifies the ShortCut property for client menu items.
	Visible (see page 1857)	Specifies the Visible property for client controls and menu items.

27.1.30.2 TSearchReplace Methods

The methods of the TSearchReplace class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1852)	This is GetDialogClass, a member of class TSearchReplace.

Public Methods

	Name	Description
	ExecuteTarget (see page 1852)	Launches the associated dialog and assigns an event handler to replace found text.

27.1.30.2.1 TSearchReplace.ExecuteTarget Method

Launches the associated dialog and assigns an event handler to replace found text.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches the search-and-replace dialog specified by Dialog (see page 1853). This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 1720) property, and an OnAccept (see page 1858) event occurs if the dialog was brought into focus, or an OnCancel (see page 1858) event occurs if the dialog failed to appear.

Finally, ExecuteTarget sets the OnReplace event handler of the search-and-replace dialog to a private method that performs the replacement specified by the dialog.

See Also

onaccept (see page 1721)

oncancel (see page 1722)

Dialog (see page 1853)

Execute

ExecuteResult (see page 1720)

OnReplace

27.1.30.2.2 TSearchReplace.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchReplace.

27.1.30.3 TSearchReplace Properties

The properties of the TSearchReplace class are listed here.

Published Properties

	Name	Description
	Caption (see page 1853)	Represents the caption of client controls and menu items.
R	Dialog (see page 1853)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 1854)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1854)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1855)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1855)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1856)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1856)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1856)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1857)	Specifies the ShortCut property for client menu items.
	Visible (see page 1857)	Specifies the Visible property for client controls and menu items.

27.1.30.3.1 TSearchReplace.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

27.1.30.3.2 TSearchReplace.Dialog Property

Provides access the find-and-replace dialog that the action executes.

Pascal

```
property Dialog: TReplaceDialog;
```

C++

```
__property TReplaceDialog Dialog;
```

Description

Use Dialog to configure the find-and-replace dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to shut down the modeless find-and-replace dialog.

When using a search/replace action, you do not place a TReplaceDialog component on the form or data module at design time. Instead, you add the TSearchReplace (see page 1850) action to your action list and then use the Dialog property. The TReplaceDialog component is created by the TSearchReplace (see page 1850) constructor.

See Also

ExecuteTarget (see page 1826)

CloseDialog

27.1.30.3.3 TSearchReplace.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

27.1.30.3.4 TSearchReplace.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

27.1.30.3.5 TSearchReplace.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

HelpKey

27.1.30.3.6 TSearchReplace.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpType

HelpContext

HelpKeyword

27.1.30.3.7 TSearchReplace.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

27.1.30.3.8 TSearchReplace.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

27.1.30.3.9 TSearchReplace.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

27.1.30.3.10 TSearchReplace.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

27.1.30.3.11 TSearchReplace.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

27.1.30.4 TSearchReplace Events

The events of the TSearchReplace class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1858)	Occurs immediately before the dialog appears.
	OnAccept (see page 1858)	This is OnAccept, a member of class TSearchReplace.
	OnCancel (see page 1858)	This is OnCancel, a member of class TSearchReplace.
	OnHint (see page 1858)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1859)	Occurs when the application is idle or when the action list updates.

27.1.30.4.1 TSearchReplace.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1718)

Execute

PropertyName

27.1.30.4.2 TSearchReplace.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TSearchReplace.

27.1.30.4.3 TSearchReplace.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TSearchReplace.

27.1.30.4.4 TSearchReplace.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

27.1.30.4.5 TSearchReplace.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
```

```
  for an action that is "checked" when the toolbar is visible.
  This example requires a TActionList (with one TAction) and a
  TToolBar on a form. This example requires a TActionList
  (with one TAction) and a TToolBar on a form.  Double click
  on the TActionList and create an TAction named Action1.  The
  TAction has an OnUpdate event handler called Action1Update.
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

27.1.31 TWindowAction Class

TWindowAction is the base class for action objects that work with MDI forms.

Class Hierarchy

Pascal

```
TWindowAction = class(TAction);
```

C++

```
class TWindowAction : public TAction;
```

File

StdActns

Description

TWindowAction implements the common behavior for actions that work on MDI forms. Each descendant class performs a specific task, such as arranging, cascading, closing, tiling, or minimizing MDI child forms. Do not use TWindowAction directly in an application. Instead, use a TWindowAction descendant such as TWindowClose (see page 1867), TWindowCascade (see page 1865), TWindowMinimizeAll (see page 1869), TWindowArrange (see page 1864), TWindowTileHorizontal (see page 1871), or TWindowTileVertical (see page 1873).

TWindowAction ensures that the action is enabled as long as the target is an MDI parent form. TWindowAction descendants override the ExecuteTarget method to execute their specific function on the targets child forms.

See Also

TWindowArrange (see page 1864)

TWindowCascade (see page 1865)

TWindowClose (see page 1867)

TWindowMinimizeAll (see page 1869)

TWindowTileHorizontal (see page 1871)

TWindowTileVertical (see page 1873)

27.1.31.1 TWindowAction Members

The following tables list the members exposed by TWindowAction.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.31.2 TWindowAction.Create Constructor

Creates and initializes a TAction object.

Pascal

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

C++

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

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList

27.1.31.3 TWindowAction Methods

The methods of the TWindowAction class are listed here.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

27.1.31.3.1 TWindowAction.GetForm Method

Pascal

```
function GetForm(Target: TObject): TForm; virtual;
```

C++

```
virtual __fastcall TForm * GetForm(TObject * Target);
```

Description

This is GetForm, a member of class TWindowAction.

27.1.31.3.2 TWindowAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in TWindowAction (see page 1860), HandlesTarget makes sure that the target is an MDI parent form. If the target is appropriate, HandlesTarget sets the Form (see page 1863) property.

See Also

Form (see page 1863)

FormStyle

27.1.31.3.3 TWindowAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

27.1.31.3.4 TWindowAction.UpdateTarget Method

Enables or disables the action, depending on whether the target form has MDI children.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowAction (see page 1860), UpdateTarget checks that the target object is an MDI parent form with an MDIChildCount property greater than 0. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled

MDIChildCount

27.1.31.4 TWindowAction Properties

The properties of the TWindowAction class are listed here.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.31.4.1 TWindowAction.Form Property

Provides access to the target of the action.

Pascal

```
property Form: TCustomForm;
```

C++

```
__property TCustomForm Form;
```

Description

Form is the MDI parent form on which the action performs its function.

27.1.32 TWindowArrange Class

TWindowArrange arranges the icons of minimized MDI child forms.

Class Hierarchy

Pascal

```
TWindowArrange = class(TWindowAction);
```

C++

```
class TWindowArrange : public TWindowAction;
```

File

StdActns

Description

Add TWindowArrange to an action list to allow users to arrange the icons that represent minimized MDI child forms. Controls such as menu items and tool buttons linked to this action cause the application to call the ArrangeIcons method of the MDI parent form.

TWindowArrange is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1867)

TWindowCascade (see page 1865)

TWindowMinimizeAll (see page 1869)

TWindowTileHorizontal (see page 1871)

TWindowTileVertical (see page 1873)

TForm

27.1.32.1 TWindowArrange Members

The following tables list the members exposed by TWindowArrange.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1865)	Arranges the icons of minimized MDI child forms
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.32.2 TWindowArrange Methods

The methods of the TWindowArrange class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1865)	Arranges the icons of minimized MDI child forms

27.1.32.2.1 TWindowArrange.ExecuteTarget Method

Arranges the icons of minimized MDI child forms

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then calls that form's ArrangeIcons method to arrange the icons of minimized MDI children.

See Also

Form (see page 1863)

ArrangeIcons

27.1.33 TWindowCascade Class

TWindowCascade cascades the MDI child forms.

Class Hierarchy

Pascal

```
TWindowCascade = class(TWindowAction);
```

C++

```
class TWindowCascade : public TWindowAction;
```

File

StdActns

Description

Add TWindowCascade to an action list to allow users to arrange the MDI child forms in an overlapping cascade. Controls such as menu items and tool buttons linked to this action cause the application to call the Cascade method of the MDI parent form.

TWindowCascade is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1867)

TWindowArrange (see page 1864)

TWindowMinimizeAll (see page 1869)

TWindowTileHorizontal (see page 1871)

TWindowTileVertical (see page 1873)

TForm

27.1.33.1 TWindowCascade Members

The following tables list the members exposed by TWindowCascade.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1866)	Arranges the MDI child forms in an overlapping cascade.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.33.2 TWindowCascade Methods

The methods of the TWindowCascade class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1866)	Arranges the MDI child forms in an overlapping cascade.

27.1.33.2.1 TWindowCascade.ExecuteTarget Method

Arranges the MDI child forms in an overlapping cascade.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then calls that form's Cascade method to arrange MDI child windows.

See Also

Form (see page 1863)

Cascade

27.1.34 TWindowClose Class

TWindowClose closes the active MDI child form.

Class Hierarchy

Pascal

```
TWindowClose = class(TWindowAction);
```

C++

```
class TWindowClose : public TWindowAction;
```

File

StdActns

Description

Add TWindowClose to an action list to allow users to close the active MDI child form. Controls such as menu items and tool buttons linked to this action cause the application to call the Close (see page 3643) method of the active MDI child.

TWindowClose is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no active MDI child.

See Also

TWindowArrange (see page 1864)

TWindowCascade (see page 1865)

TWindowMinimizeAll (see page 1869)

TWindowTileHorizontal (see page 1871)

TWindowTileVertical (see page 1873)

TCustomForm

27.1.34.1 TWindowClose Members

The following tables list the members exposed by TWindowClose.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1868)	Closes the active MDI child.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1869)	Enables or disables the action, depending on whether the target form an active child.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.34.2 TWindowClose Methods

The methods of the TWindowClose class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1868)	Closes the active MDI child.
	UpdateTarget (see page 1869)	Enables or disables the action, depending on whether the target form an active child.

27.1.34.2.1 TWindowClose.ExecuteTarget Method

Closes the active MDI child.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then obtains a reference to the active MDI child by reading the form's ActiveMDIChild property. It calls this child form's Close (see page 3643) method to close the active child form.

See Also

Form (see page 1863)

ActiveMDIChild

Close

27.1.34.2.2 TWindowClose.UpdateTarget Method

Enables or disables the action, depending on whether the target form an active child.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowClose (see page 1867), UpdateTarget checks that the target object is an MDI parent form with a non-nil (Delphi) or NULL (C++) ActiveMDIChild property. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled

ActiveMDIChild

27.1.35 TWindowMinimizeAll Class

TWindowMinimizeAll minimizes all MDI child forms.

Class Hierarchy

Pascal

```
TWindowMinimizeAll = class(TWindowAction);
```

C++

```
class TWindowMinimizeAll : public TWindowAction;
```

File

StdActns

Description

Add TWindowMinimizeAll to an action list to allow users to minimize the MDI child forms in an MDI application. Controls such as menu items and tool buttons linked to this action cause the application to iterate through the MDI child forms, setting their WindowState to wsMinimized.

TWindowMinimizeAll is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1867)

TWindowArrange (see page 1864)

TWindowCascade (see page 1865)

TWindowTileHorizontal (see page 1871)

TWindowTileVertical (see page 1873)

TCustomForm

TCustomForm

27.1.35.1 TWindowMinimizeAll Members

The following tables list the members exposed by TWindowMinimizeAll.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1870)	Minimizes all MDI child forms of the target MDI parent.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.35.2 TWindowMinimizeAll Methods

The methods of the TWindowMinimizeAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1870)	Minimizes all MDI child forms of the target MDI parent.

27.1.35.2.1 TWindowMinimizeAll.ExecuteTarget Method

Minimizes all MDI child forms of the target MDI parent.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then iterates through that form's MDI children, setting each child form's WindowState to wsMinimized.

See Also

Form (see page 1863)

MDIChildren

WindowState

27.1.36 TWindowTileHorizontal Class

TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.

Class Hierarchy

Pascal

```
TWindowTileHorizontal = class(TWindowAction);
```

C++

```
class TWindowTileHorizontal : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileHorizontal to an action list to allow users to tile the MDI child forms horizontally. Controls such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children horizontally.

TWindowTileHorizontal is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1867)

TWindowArrange (see page 1864)

TWindowCascade (see page 1865)

TWindowMinimizeAll (see page 1869)

TWindowTileVertical (see page 1873)

TCustomForm

27.1.36.1 TWindowTileHorizontal Members

The following tables list the members exposed by TWindowTileHorizontal.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1872)	Tiles the child forms of the target MDI parent horizontally.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.36.2 TWindowTileHorizontal Methods

The methods of the TWindowTileHorizontal class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1872)	Tiles the child forms of the target MDI parent horizontally.

27.1.36.2.1 TWindowTileHorizontal.ExecuteTarget Method

Tiles the child forms of the target MDI parent horizontally.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms horizontally.

See Also

Form (see page 1863)

MDIChildren

27.1.37 TWindowTileVertical Class

TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Class Hierarchy

Pascal

```
TWindowTileVertical = class(TWindowAction);
```

C++

```
class TWindowTileVertical : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileVertical to an action list to allow users to tile the MDI child forms vertically. Controls such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children vertically.

TWindowTileVertical is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1867)

TWindowArrange (see page 1864)

TWindowCascade (see page 1865)

TWindowMinimizeAll (see page 1869)

TWindowTileVertical

TCustomForm

27.1.37.1 TWindowTileVertical Members

The following tables list the members exposed by TWindowTileVertical.

Protected Methods

	Name	Description
	GetForm (see page 1862)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1862)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1861)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1874)	Tiles the child forms of the target MDI parent vertically.
	HandlesTarget (see page 1862)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1863)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1863)	Provides access to the target of the action.

27.1.37.2 TWindowTileVertical Methods

The methods of the TWindowTileVertical class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1874)	Tiles the child forms of the target MDI parent vertically.

27.1.37.2.1 TWindowTileVertical.ExecuteTarget Method

Tiles the child forms of the target MDI parent vertically.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1863) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms vertically.

See Also

Form (see page 1863)

MDIChildren

27.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBrowseForFolderOption (see page 1875)	This is record StdActns.TBrowseForFolderOption.

27.2.1 StdActns.TBrowseForFolderOption Enumeration

Pascal

```
TBrowseForFolderOption = (
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
);
```

C++

```
enum TBrowseForFolderOption {
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
};
```

File

StdActns

Description

This is record StdActns.TBrowseForFolderOption.

27.3 Types

The following table lists types in this documentation.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 1877)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 1877)	This is type StdActns.TBrowseOptions.
TCommonDialogClass (see page 1878)	This is type StdActns.TCommonDialogClass.

27.3.1 StdActns.TBrowseForFolderCallbackEvent Type

Pascal

```
TBrowseForFolderCallbackEvent = procedure (Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT;
lParam, lpData: LPARAM; var Result: Integer) of object;
```

C++

```
(Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT; lParam, lpData: LPARAM; var Result: Integer)
( TBrowseForFolderCallbackEvent)();
```

File

StdActns

Description

This is type StdActns.TBrowseForFolderCallbackEvent.

27.3.2 StdActns.TBrowseOptions Type

Pascal

```
TBrowseOptions = set of TBrowseForFolderOption;
```

C++

```
TBrowseForFolderOption TBrowseOptions;
```

File

StdActns

Description

This is type StdActns.TBrowseOptions.

27.3.3 StdActns.TCommonDialogClass Type

Pascal

```
TCommonDialogClass = class of TCommonDialog;
```

C++

```
TCommonDialog TCommonDialogClass;
```

File

StdActns

Description

This is type StdActns.TCommonDialogClass.

28 StdConvs Namespace

This is namespace StdConvs.

Topics

Name	Description
Functions (see page 1884)	The following table lists functions in this documentation.
Variables (see page 1886)	The following table lists variables in this documentation.
Constants (see page 1936)	The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 1937)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 1937)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 1938)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 1938)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 1938)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 1939)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 1939)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 1939)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 1940)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 1940)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 1940)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 1941)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 1941)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 1941)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 1942)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 1942)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 1942)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidGallon.

CubicMetersPerUSFluidGill (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 1947)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 1947)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 1947)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 1948)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 1948)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 1948)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 1949)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 1949)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 1949)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 1950)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 1950)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 1950)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 1951)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 1951)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 1951)	This is constant StdConvs.MetersPerFurlong.
MetersPerHand (see page 1952)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 1952)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 1952)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 1953)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 1953)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 1953)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 1954)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 1954)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 1954)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 1955)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 1955)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 1955)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 1956)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 1956)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 1956)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 1957)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 1957)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 1957)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 1958)	This is constant StdConvs.SquareMetersPerSquareYard.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 1884)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 1884)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 1885)	Initialize (see page 3698) conversion variables for C++ program.

Variables

Name	Description
auAcres (see page 1888)	This is variable StdConvs.auAcres.
auAres (see page 1889)	This is variable StdConvs.auAres.
auCentares (see page 1889)	This is variable StdConvs.auCentares.
auHectares (see page 1889)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 1890)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 1890)	This is variable StdConvs.auSquareDecameters.

auSquareDecimeters (see page 1890)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 1891)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 1891)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 1891)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 1892)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 1892)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 1892)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 1893)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 1893)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 1893)	This is variable StdConvs.auSquareYards.
cbArea (see page 1894)	Represent units that measure cbArea.
cbDistance (see page 1894)	Represent units that measure cbDistance.
cbMass (see page 1895)	This is variable StdConvs.cbMass.
cbTemperature (see page 1895)	This is variable StdConvs.cbTemperature.
cbTime (see page 1895)	Represent units that measure cbTime.
cbVolume (see page 1896)	Represent units that measure volume.
duAngstroms (see page 1897)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 1897)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 1898)	This is variable StdConvs.duCentimeters.
duChains (see page 1898)	This is variable StdConvs.duChains.
duCubits (see page 1898)	This is variable StdConvs.duCubits.
duDecameters (see page 1899)	This is variable StdConvs.duDecameters.
duDecimeters (see page 1899)	This is variable StdConvs.duDecimeters.
duFathoms (see page 1899)	This is variable StdConvs.duFathoms.
duFeet (see page 1900)	This is variable StdConvs.duFeet.
duFurlongs (see page 1900)	This is variable StdConvs.duFurlongs.
duGigameters (see page 1900)	This is variable StdConvs.duGigameters.
duHands (see page 1901)	This is variable StdConvs.duHands.
duHectometers (see page 1901)	This is variable StdConvs.duHectometers.
duInches (see page 1901)	This is variable StdConvs.duInches.
duKilometers (see page 1902)	This is variable StdConvs.duKilometers.
duLightYears (see page 1902)	This is variable StdConvs.duLightYears.
duLinks (see page 1902)	This is variable StdConvs.duLinks.
duMegameters (see page 1903)	This is variable StdConvs.duMegameters.
duMeters (see page 1903)	This is variable StdConvs.duMeters.
duMicromicrons (see page 1903)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 1904)	This is variable StdConvs.duMicrons.
duMiles (see page 1904)	This is variable StdConvs.duMiles.
duMillimeters (see page 1904)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 1905)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 1905)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 1905)	This is variable StdConvs.duPaces.
duParsecs (see page 1906)	This is variable StdConvs.duParsecs.
duPicas (see page 1906)	This is variable StdConvs.duPicas.
duPoints (see page 1906)	This is variable StdConvs.duPoints.
duRods (see page 1907)	This is variable StdConvs.duRods.
duYards (see page 1907)	This is variable StdConvs.duYards.
muCentigrams (see page 1907)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 1908)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 1908)	This is variable StdConvs.muDecigrams.
muDrams (see page 1908)	This is variable StdConvs.muDrams.
muGrains (see page 1909)	This is variable StdConvs.muGrains.
muGrams (see page 1909)	This is variable StdConvs.muGrams.
muHectograms (see page 1909)	This is variable StdConvs.muHectograms.

muKilograms (see page 1910)	This is variable StdConvs.muKilograms.
muLongTons (see page 1910)	This is variable StdConvs.muLongTons.
muMetricTons (see page 1910)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 1911)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 1911)	This is variable StdConvs.muMilligrams.
muNanograms (see page 1911)	This is variable StdConvs.muNanograms.
muOunces (see page 1912)	This is variable StdConvs.muOunces.
muPounds (see page 1912)	This is variable StdConvs.muPounds.
muStones (see page 1912)	This is variable StdConvs.muStones.
muTons (see page 1913)	This is variable StdConvs.muTons.
tuCelsius (see page 1913)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 1913)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 1914)	This is variable StdConvs.tuDateTime.
tuDays (see page 1914)	This is variable StdConvs.tuDays.
tuDecades (see page 1914)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 1915)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 1915)	This is variable StdConvs.tuFortnights.
tuHours (see page 1915)	This is variable StdConvs.tuHours.
tuJulianDate (see page 1916)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 1916)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 1916)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 1917)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 1917)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 1917)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 1918)	This is variable StdConvs.tuMonths.
tuRankine (see page 1918)	This is variable StdConvs.tuRankine.
tuReaumur (see page 1918)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 1919)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 1919)	This is variable StdConvs.tuWeeks.
tuYears (see page 1919)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 1920)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 1920)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 1920)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 1921)	This is variable StdConvs.vuCordFeet.
vuCords (see page 1921)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 1921)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 1922)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 1922)	This is variable StdConvs.vuCubicDecimeters.
vuCubicFeet (see page 1922)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 1923)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 1923)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 1923)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 1924)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 1924)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 1924)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 1925)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 1925)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 1925)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 1926)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 1926)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 1926)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 1927)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 1927)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 1927)	This is variable StdConvs.vuDryPecks.

vuDryPints (see page 1928)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 1928)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 1928)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 1929)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 1929)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 1929)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 1930)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 1930)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 1930)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 1931)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 1931)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 1931)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 1932)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 1932)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 1932)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 1933)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 1933)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 1933)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 1934)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 1934)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 1934)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 1935)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 1935)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 1935)	This is variable StdConvs.vuUKQuarts.

28.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 1884)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 1884)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 1885)	Initialize (see page 3698) conversion variables for C++ program.

28.1.1 StdConvs.CelsiusToFahrenheit Function

Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.

Pascal

```
function CelsiusToFahrenheit(const AValue: Double): Double;
```

C++

```
Double CelsiusToFahrenheit(const Double AValue);
```

File

StdConvs

Description

Call CelsiusToFahrenheit to convert the temperature specified by AValue, which is in degrees Celsius, to the corresponding temperature in degrees Fahrenheit.

See Also

FahrenheitToCelsius (see page 1884)

Convert

28.1.2 StdConvs.FahrenheitToCelsius Function

Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.

Pascal

```
function FahrenheitToCelsius(const AValue: Double): Double;
```

C++

```
Double FahrenheitToCelsius(const Double AValue);
```

File

StdConvs

Description

Call FahrenheitToCelsius to convert the temperature specified by AValue, which is in degrees Fahrenheit, to the corresponding temperature in degrees Celsius.

See Also

CelsiusToFahrenheit (see page 1884)

Convert

28.1.3 StdConvs.InitStdConvs Function

Initialize (see page 3698) conversion variables for C++ program.

Pascal

```
procedure InitStdConvs;
```

C++

```
InitStdConvs();
```

File

StdConvs

Description

Insert (see page 3698) InitStdConvs into a C++ program to ensure that StdConvs (see page 1879) global variables are properly initialized. InitStdConvs does nothing and does not need to be called. A reference to InitStdConvs in a C++ program guarantees that the linker will provide a call to the unit's initialization code, even if no other StdConvs (see page 1879) routines are present. This step is not necessary in Delphi code, which always calls the initialization code for referenced units.

See Also

FahrenheitToCelsius (see page 1884)

Convert

28.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
auAcres (see page 1888)	This is variable StdConvs.auAcres.
auAres (see page 1889)	This is variable StdConvs.auAres.
auCentares (see page 1889)	This is variable StdConvs.auCentares.
auHectares (see page 1889)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 1890)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 1890)	This is variable StdConvs.auSquareDecameters.
auSquareDecimeters (see page 1890)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 1891)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 1891)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 1891)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 1892)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 1892)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 1892)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 1893)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 1893)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 1893)	This is variable StdConvs.auSquareYards.
cbArea (see page 1894)	Represent units that measure cbArea.
cbDistance (see page 1894)	Represent units that measure cbDistance.
cbMass (see page 1895)	This is variable StdConvs.cbMass.
cbTemperature (see page 1895)	This is variable StdConvs.cbTemperature.
cbTime (see page 1895)	Represent units that measure cbTime.
cbVolume (see page 1896)	Represent units that measure volume.
duAngstroms (see page 1897)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 1897)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 1898)	This is variable StdConvs.duCentimeters.
duChains (see page 1898)	This is variable StdConvs.duChains.
duCubits (see page 1898)	This is variable StdConvs.duCubits.
duDecameters (see page 1899)	This is variable StdConvs.duDecameters.
duDecimeters (see page 1899)	This is variable StdConvs.duDecimeters.
duFathoms (see page 1899)	This is variable StdConvs.duFathoms.
duFeet (see page 1900)	This is variable StdConvs.duFeet.
duFurlongs (see page 1900)	This is variable StdConvs.duFurlongs.
duGigameters (see page 1900)	This is variable StdConvs.duGigameters.
duHands (see page 1901)	This is variable StdConvs.duHands.
duHectometers (see page 1901)	This is variable StdConvs.duHectometers.
duInches (see page 1901)	This is variable StdConvs.duInches.
duKilometers (see page 1902)	This is variable StdConvs.duKilometers.
duLightYears (see page 1902)	This is variable StdConvs.duLightYears.
duLinks (see page 1902)	This is variable StdConvs.duLinks.
duMegameters (see page 1903)	This is variable StdConvs.duMegameters.
duMeters (see page 1903)	This is variable StdConvs.duMeters.
duMicromicrons (see page 1903)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 1904)	This is variable StdConvs.duMicrons.
duMiles (see page 1904)	This is variable StdConvs.duMiles.

duMillimeters (see page 1904)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 1905)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 1905)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 1905)	This is variable StdConvs.duPaces.
duParsecs (see page 1906)	This is variable StdConvs.duParsecs.
duPicas (see page 1906)	This is variable StdConvs.duPicas.
duPoints (see page 1906)	This is variable StdConvs.duPoints.
duRods (see page 1907)	This is variable StdConvs.duRods.
duYards (see page 1907)	This is variable StdConvs.duYards.
muCentigrams (see page 1907)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 1908)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 1908)	This is variable StdConvs.muDecigrams.
muDrams (see page 1908)	This is variable StdConvs.muDrams.
muGrains (see page 1909)	This is variable StdConvs.muGrains.
muGrams (see page 1909)	This is variable StdConvs.muGrams.
muHectograms (see page 1909)	This is variable StdConvs.muHectograms.
muKilograms (see page 1910)	This is variable StdConvs.muKilograms.
muLongTons (see page 1910)	This is variable StdConvs.muLongTons.
muMetricTons (see page 1910)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 1911)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 1911)	This is variable StdConvs.muMilligrams.
muNanograms (see page 1911)	This is variable StdConvs.muNanograms.
muOunces (see page 1912)	This is variable StdConvs.muOunces.
muPounds (see page 1912)	This is variable StdConvs.muPounds.
muStones (see page 1912)	This is variable StdConvs.muStones.
muTons (see page 1913)	This is variable StdConvs.muTons.
tuCelsius (see page 1913)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 1913)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 1914)	This is variable StdConvs.tuDateTime.
tuDays (see page 1914)	This is variable StdConvs.tuDays.
tuDecades (see page 1914)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 1915)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 1915)	This is variable StdConvs.tuFortnights.
tuHours (see page 1915)	This is variable StdConvs.tuHours.
tuJulianDate (see page 1916)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 1916)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 1916)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 1917)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 1917)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 1917)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 1918)	This is variable StdConvs.tuMonths.
tuRankine (see page 1918)	This is variable StdConvs.tuRankine.
tuReaumur (see page 1918)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 1919)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 1919)	This is variable StdConvs.tuWeeks.
tuYears (see page 1919)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 1920)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 1920)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 1920)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 1921)	This is variable StdConvs.vuCordFeet.
vuCords (see page 1921)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 1921)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 1922)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 1922)	This is variable StdConvs.vuCubicDecimeters.

vuCubicFeet (see page 1922)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 1923)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 1923)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 1923)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 1924)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 1924)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 1924)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 1925)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 1925)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 1925)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 1926)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 1926)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 1926)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 1927)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 1927)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 1927)	This is variable StdConvs.vuDryPecks.
vuDryPints (see page 1928)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 1928)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 1928)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 1929)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 1929)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 1929)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 1930)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 1930)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 1930)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 1931)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 1931)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 1931)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 1932)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 1932)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 1932)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 1933)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 1933)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 1933)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 1934)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 1934)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 1934)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 1935)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 1935)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 1935)	This is variable StdConvs.vuUKQuarts.

28.2.1 StdConvs.auAcres Variable

Pascal

```
auAcres: TConvType;
```

C++

```
TConvType auAcres;
```

File

StdConvs

Description

This is variable StdConvs.auAcres.

28.2.2 StdConvs.auAres Variable

Pascal

```
auAres: TConvType;
```

C++

```
TConvType auAres;
```

File

StdConvs

Description

This is variable StdConvs.auAres.

28.2.3 StdConvs.auCentares Variable

Pascal

```
auCentares: TConvType;
```

C++

```
TConvType auCentares;
```

File

StdConvs

Description

This is variable StdConvs.auCentares.

28.2.4 StdConvs.auHectares Variable

Pascal

```
auHectares: TConvType;
```

C++

```
TConvType auHectares;
```

File

StdConvs

Description

This is variable StdConvs.auHectares.

28.2.5 StdConvs.auSquareCentimeters Variable

Pascal

```
auSquareCentimeters: TConvType;
```

C++

```
TConvType auSquareCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareCentimeters.

28.2.6 StdConvs.auSquareDecameters Variable

Pascal

```
auSquareDecameters: TConvType;
```

C++

```
TConvType auSquareDecameters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecameters.

28.2.7 StdConvs.auSquareDecimeters Variable

Pascal

```
auSquareDecimeters: TConvType;
```

C++

```
TConvType auSquareDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecimeters.

28.2.8 StdConvs.auSquareFeet Variable

Pascal

```
auSquareFeet: TConvType;
```

C++

```
TConvType auSquareFeet;
```

File

StdConvs

Description

This is variable StdConvs.auSquareFeet.

28.2.9 StdConvs.auSquareHectometers Variable

Pascal

```
auSquareHectometers: TConvType;
```

C++

```
TConvType auSquareHectometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareHectometers.

28.2.10 StdConvs.auSquareInches Variable

Pascal

```
auSquareInches: TConvType;
```

C++

```
TConvType auSquareInches;
```

File

StdConvs

Description

This is variable StdConvs.auSquareInches.

28.2.11 StdConvs.auSquareKilometers Variable

Pascal

```
auSquareKilometers: TConvType;
```

C++

```
TConvType auSquareKilometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareKilometers.

28.2.12 StdConvs.auSquareMeters Variable

Pascal

```
auSquareMeters: TConvType;
```

C++

```
TConvType auSquareMeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMeters.

28.2.13 StdConvs.auSquareMiles Variable

Pascal

```
auSquareMiles: TConvType;
```

C++

```
TConvType auSquareMiles;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMiles.

28.2.14 StdConvs.auSquareMillimeters Variable

Pascal

```
auSquareMillimeters: TConvType;
```

C++

```
TConvType auSquareMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMillimeters.

28.2.15 StdConvs.auSquareRods Variable

Pascal

```
auSquareRods: TConvType;
```

C++

```
TConvType auSquareRods;
```

File

StdConvs

Description

This is variable StdConvs.auSquareRods.

28.2.16 StdConvs.auSquareYards Variable

Pascal

```
auSquareYards: TConvType;
```

C++

```
TConvType auSquareYards;
```

File

StdConvs

Description

This is variable StdConvs.auSquareYards.

28.2.17 StdConvs.cbArea Variable

Represent units that measure cbArea.

Pascal

```
cbArea: TConvFamily;
```

C++

```
TConvFamily cbArea;
```

File

StdConvs

Description

Use the cbArea unit variables as parameters to the Convert function to convert an cbArea measurement to a different set of units. The StdConvs (see page 1879) unit registers these cbArea measurement units in the cbArea family and assigns the resulting TConvType values to these global variables.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1885) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1885)

TConvType

28.2.18 StdConvs.cbDistance Variable

Represent units that measure cbDistance.

Pascal

```
cbDistance: TConvFamily;
```

C++

```
TConvFamily cbDistance;
```

File

StdConvs

Description

Use the cbDistance unit variables as parameters to the Convert function to convert a cbDistance measurement to a different set of units. The StdConvs (see page 1879) unit registers these cbDistance measurement units in the cbDistance family and assigns the resulting TConvType values to these global variables.

Note: The cbDistance units vary greatly in scale. If you are converting between cbDistances that are all at one extreme (for example between parsecs and light years or between angstroms and microns), you may want to register your own conversion family (for example astronomical distances or microscopic distances) so that the conversions can be made with greater precision.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1885) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1885)

TConvType

28.2.19 StdConvs.cbMass Variable

Pascal

```
cbMass: TConvFamily;
```

C++

```
TConvFamily cbMass;
```

File

StdConvs

Description

This is variable StdConvs.cbMass.

28.2.20 StdConvs.cbTemperature Variable

Pascal

```
cbTemperature: TConvFamily;
```

C++

```
TConvFamily cbTemperature;
```

File

StdConvs

Description

This is variable StdConvs.cbTemperature.

28.2.21 StdConvs.cbTime Variable

Represent units that measure cbTime.

Pascal

```
cbTime: TConvFamily;
```

C++

```
TConvFamily cbTime;
```

File

StdConvs

Description

Use the cbTime unit variables as parameters to the Convert function to convert a cbTime measurement to a different set of units. The StdConvs (see page 1879) unit registers these cbTime measurement units in the cbTime family and assigns the resulting TConvType values to these global variables.

Note: Conversions involving tuMonths (see page 1918), tuYears (see page 1919), tuDecades (see page 1914), tuCenturies (see page 1913), and tuMillennia (see page 1916) are approximations, due to the fact that not all months nor all years are the same length. If you are only using units that are multiples of years, you may want to register your own conversion family that uses years as a base unit so that conversions are more accurate.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1885) procedure.

See Also

Convert

RegisterConversionType

FahrenheitToCelsius (see page 1884)

CelsiusToFahrenheit (see page 1884)

InitStdConvs (see page 1885)

TConvType

28.2.22 StdConvs.cbVolume Variable

Represent units that measure volume.

Pascal

```
cbVolume: TConvFamily;
```

C++

```
TConvFamily cbVolume;
```

File

StdConvs

Description

Use the cbVolume unit variables as parameters to the Convert function to convert a cbVolume measurement to a different set of units. The StdConvs (see page 1879) unit registers these cbVolume measurement units in the cbVolume family and assigns the resulting TConvType values to these global variables.

The interpretation of cbVolume measurements (ounces, gills, pints, quarts, gallons, pecks, bushels, and buckets) varies, depending on whether you are using American or British Imperial measurements. The vuFluidOunces (see page 1929), vuFluidGills (see page 1929), vuFluidPints (see page 1930), vuFluidQuarts (see page 1930), vuFluidGallons (see page 1929), vuDryPecks (see page 1927), vuDryBushels (see page 1927), and vuDryBuckets (see page 1926) variables refer

to the American units, while the vuUKOunces (see page 1934), vuUKGills (see page 1934), vuUKPints (see page 1935), vuUKQuarts (see page 1935), vuUKGallons (see page 1933), vuUKPecks (see page 1934), vuUKBushels (see page 1933), and vuUKBuckets (see page 1933) variables refer to the British Imperial units. Similarly, certain American dry measurements have the same names as American fluid cbVolume measurements (dry pints, dry quarts, and dry gallons), and are distinguished by using the variables vuDryPints (see page 1928), vuDryQuarts (see page 1928), and vuDryGallons (see page 1927).

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1885) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1885)

TConvType

28.2.23 StdConvs.duAngstroms Variable

Pascal

```
duAngstroms: TConvType;
```

C++

```
TConvType duAngstroms;
```

File

StdConvs

Description

This is variable StdConvs.duAngstroms.

28.2.24 StdConvs.duAstronomicalUnits Variable

Pascal

```
duAstronomicalUnits: TConvType;
```

C++

```
TConvType duAstronomicalUnits;
```

File

StdConvs

Description

This is variable StdConvs.duAstronomicalUnits.

28.2.25 StdConvs.duCentimeters Variable

Pascal

```
duCentimeters: TConvType;
```

C++

```
TConvType duCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.duCentimeters.

28.2.26 StdConvs.duChains Variable

Pascal

```
duChains: TConvType;
```

C++

```
TConvType duChains;
```

File

StdConvs

Description

This is variable StdConvs.duChains.

28.2.27 StdConvs.duCubits Variable

Pascal

```
duCubits: TConvType;
```

C++

```
TConvType duCubits;
```

File

StdConvs

Description

This is variable StdConvs.duCubits.

28.2.28 StdConvs.duDecameters Variable

Pascal

```
duDecameters: TConvType;
```

C++

```
TConvType duDecameters;
```

File

StdConvs

Description

This is variable StdConvs.duDecameters.

28.2.29 StdConvs.duDecimeters Variable

Pascal

```
duDecimeters: TConvType;
```

C++

```
TConvType duDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.duDecimeters.

28.2.30 StdConvs.duFathoms Variable

Pascal

```
duFathoms: TConvType;
```

C++

```
TConvType duFathoms;
```

File

StdConvs

Description

This is variable StdConvs.duFathoms.

28.2.31 StdConvs.duFeet Variable

Pascal

```
duFeet: TConvType;
```

C++

```
TConvType duFeet;
```

File

StdConvs

Description

This is variable StdConvs.duFeet.

28.2.32 StdConvs.duFurlongs Variable

Pascal

```
duFurlongs: TConvType;
```

C++

```
TConvType duFurlongs;
```

File

StdConvs

Description

This is variable StdConvs.duFurlongs.

28.2.33 StdConvs.duGigameters Variable

Pascal

```
duGigameters: TConvType;
```

C++

```
TConvType duGigameters;
```

File

StdConvs

Description

This is variable StdConvs.duGigameters.

28.2.34 StdConvs.duHands Variable

Pascal

```
duHands: TConvType;
```

C++

```
TConvType duHands;
```

File

StdConvs

Description

This is variable StdConvs.duHands.

28.2.35 StdConvs.duHectometers Variable

Pascal

```
duHectometers: TConvType;
```

C++

```
TConvType duHectometers;
```

File

StdConvs

Description

This is variable StdConvs.duHectometers.

28.2.36 StdConvs.duInches Variable

Pascal

```
duInches: TConvType;
```

C++

```
TConvType duInches;
```

File

StdConvs

Description

This is variable StdConvs.duInches.

28.2.37 StdConvs.duKilometers Variable

Pascal

```
duKilometers: TConvType;
```

C++

```
TConvType duKilometers;
```

File

StdConvs

Description

This is variable StdConvs.duKilometers.

28.2.38 StdConvs.duLightYears Variable

Pascal

```
duLightYears: TConvType;
```

C++

```
TConvType duLightYears;
```

File

StdConvs

Description

This is variable StdConvs.duLightYears.

28.2.39 StdConvs.duLinks Variable

Pascal

```
duLinks: TConvType;
```

C++

```
TConvType duLinks;
```

File

StdConvs

Description

This is variable StdConvs.duLinks.

28.2.40 StdConvs.duMegameters Variable

Pascal

```
duMegameters: TConvType;
```

C++

```
TConvType duMegameters;
```

File

StdConvs

Description

This is variable StdConvs.duMegameters.

28.2.41 StdConvs.duMeters Variable

Pascal

```
duMeters: TConvType;
```

C++

```
TConvType duMeters;
```

File

StdConvs

Description

This is variable StdConvs.duMeters.

28.2.42 StdConvs.duMicromicrons Variable

Pascal

```
duMicromicrons: TConvType;
```

C++

```
TConvType duMicromicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicromicrons.

28.2.43 StdConvs.duMicrons Variable

Pascal

```
duMicrons: TConvType;
```

C++

```
TConvType duMicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicrons.

28.2.44 StdConvs.duMiles Variable

Pascal

```
duMiles: TConvType;
```

C++

```
TConvType duMiles;
```

File

StdConvs

Description

This is variable StdConvs.duMiles.

28.2.45 StdConvs.duMillimeters Variable

Pascal

```
duMillimeters: TConvType;
```

C++

```
TConvType duMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.duMillimeters.

28.2.46 StdConvs.duMillimicrons Variable

Pascal

```
duMillimicrons: TConvType;
```

C++

```
TConvType duMillimicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMillimicrons.

28.2.47 StdConvs.duNauticalMiles Variable

Pascal

```
duNauticalMiles: TConvType;
```

C++

```
TConvType duNauticalMiles;
```

File

StdConvs

Description

This is variable StdConvs.duNauticalMiles.

28.2.48 StdConvs.duPaces Variable

Pascal

```
duPaces: TConvType;
```

C++

```
TConvType duPaces;
```

File

StdConvs

Description

This is variable StdConvs.duPaces.

28.2.49 StdConvs.duParsecs Variable

Pascal

```
duParsecs: TConvType;
```

C++

```
TConvType duParsecs;
```

File

StdConvs

Description

This is variable StdConvs.duParsecs.

28.2.50 StdConvs.duPicas Variable

Pascal

```
duPicas: TConvType;
```

C++

```
TConvType duPicas;
```

File

StdConvs

Description

This is variable StdConvs.duPicas.

28.2.51 StdConvs.duPoints Variable

Pascal

```
duPoints: TConvType;
```

C++

```
TConvType duPoints;
```

File

StdConvs

Description

This is variable StdConvs.duPoints.

28.2.52 StdConvs.duRods Variable

Pascal

```
duRods: TConvType;
```

C++

```
TConvType duRods;
```

File

StdConvs

Description

This is variable StdConvs.duRods.

28.2.53 StdConvs.duYards Variable

Pascal

```
duYards: TConvType;
```

C++

```
TConvType duYards;
```

File

StdConvs

Description

This is variable StdConvs.duYards.

28.2.54 StdConvs.muCentigrams Variable

Pascal

```
muCentigrams: TConvType;
```

C++

```
TConvType muCentigrams;
```

File

StdConvs

Description

This is variable StdConvs.muCentigrams.

28.2.55 StdConvs.muDecagrams Variable

Pascal

```
muDecagrams: TConvType;
```

C++

```
TConvType muDecagrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecagrams.

28.2.56 StdConvs.muDecigrams Variable

Pascal

```
muDecigrams: TConvType;
```

C++

```
TConvType muDecigrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecigrams.

28.2.57 StdConvs.muDrams Variable

Pascal

```
muDrams: TConvType;
```

C++

```
TConvType muDrams;
```

File

StdConvs

Description

This is variable StdConvs.muDrams.

28.2.58 StdConvs.muGrains Variable

Pascal

```
muGrains: TConvType;
```

C++

```
TConvType muGrains;
```

File

StdConvs

Description

This is variable StdConvs.muGrains.

28.2.59 StdConvs.muGrams Variable

Pascal

```
muGrams: TConvType;
```

C++

```
TConvType muGrams;
```

File

StdConvs

Description

This is variable StdConvs.muGrams.

28.2.60 StdConvs.muHectograms Variable

Pascal

```
muHectograms: TConvType;
```

C++

```
TConvType muHectograms;
```

File

StdConvs

Description

This is variable StdConvs.muHectograms.

28.2.61 StdConvs.muKilograms Variable

Pascal

```
muKilograms: TConvType;
```

C++

```
TConvType muKilograms;
```

File

StdConvs

Description

This is variable StdConvs.muKilograms.

28.2.62 StdConvs.muLongTons Variable

Pascal

```
muLongTons: TConvType;
```

C++

```
TConvType muLongTons;
```

File

StdConvs

Description

This is variable StdConvs.muLongTons.

28.2.63 StdConvs.muMetricTons Variable

Pascal

```
muMetricTons: TConvType;
```

C++

```
TConvType muMetricTons;
```

File

StdConvs

Description

This is variable StdConvs.muMetricTons.

28.2.64 StdConvs.muMicrograms Variable

Pascal

```
muMicrograms: TConvType;
```

C++

```
TConvType muMicrograms;
```

File

StdConvs

Description

This is variable StdConvs.muMicrograms.

28.2.65 StdConvs.muMilligrams Variable

Pascal

```
muMilligrams: TConvType;
```

C++

```
TConvType muMilligrams;
```

File

StdConvs

Description

This is variable StdConvs.muMilligrams.

28.2.66 StdConvs.muNanograms Variable

Pascal

```
muNanograms: TConvType;
```

C++

```
TConvType muNanograms;
```

File

StdConvs

Description

This is variable StdConvs.muNanograms.

28.2.67 StdConvs.muOunces Variable

Pascal

```
muOunces: TConvType;
```

C++

```
TConvType muOunces;
```

File

StdConvs

Description

This is variable StdConvs.muOunces.

28.2.68 StdConvs.muPounds Variable

Pascal

```
muPounds: TConvType;
```

C++

```
TConvType muPounds;
```

File

StdConvs

Description

This is variable StdConvs.muPounds.

28.2.69 StdConvs.muStones Variable

Pascal

```
muStones: TConvType;
```

C++

```
TConvType muStones;
```

File

StdConvs

Description

This is variable StdConvs.muStones.

28.2.70 StdConvs.muTons Variable

Pascal

```
muTons: TConvType;
```

C++

```
TConvType muTons;
```

File

StdConvs

Description

This is variable StdConvs.muTons.

28.2.71 StdConvs.tuCelsius Variable

Pascal

```
tuCelsius: TConvType;
```

C++

```
TConvType tuCelsius;
```

File

StdConvs

Description

This is variable StdConvs.tuCelsius.

28.2.72 StdConvs.tuCenturies Variable

Pascal

```
tuCenturies: TConvType;
```

C++

```
TConvType tuCenturies;
```

File

StdConvs

Description

This is variable StdConvs.tuCenturies.

28.2.73 StdConvs.tuDateTime Variable

Pascal

```
tuDateTime: TConvType;
```

C++

```
TConvType tuDateTime;
```

File

StdConvs

Description

This is variable StdConvs.tuDateTime.

28.2.74 StdConvs.tuDays Variable

Pascal

```
tuDays: TConvType;
```

C++

```
TConvType tuDays;
```

File

StdConvs

Description

This is variable StdConvs.tuDays.

28.2.75 StdConvs.tuDecades Variable

Pascal

```
tuDecades: TConvType;
```

C++

```
TConvType tuDecades;
```

File

StdConvs

Description

This is variable StdConvs.tuDecades.

28.2.76 StdConvs.tuFahrenheit Variable

Pascal

```
tuFahrenheit: TConvType;
```

C++

```
TConvType tuFahrenheit;
```

File

StdConvs

Description

This is variable StdConvs.tuFahrenheit.

28.2.77 StdConvs.tuFortnights Variable

Pascal

```
tuFortnights: TConvType;
```

C++

```
TConvType tuFortnights;
```

File

StdConvs

Description

This is variable StdConvs.tuFortnights.

28.2.78 StdConvs.tuHours Variable

Pascal

```
tuHours: TConvType;
```

C++

```
TConvType tuHours;
```

File

StdConvs

Description

This is variable StdConvs.tuHours.

28.2.79 StdConvs.tuJulianDate Variable

Pascal

```
tuJulianDate: TConvType;
```

C++

```
TConvType tuJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuJulianDate.

28.2.80 StdConvs.tuKelvin Variable

Pascal

```
tuKelvin: TConvType;
```

C++

```
TConvType tuKelvin;
```

File

StdConvs

Description

This is variable StdConvs.tuKelvin.

28.2.81 StdConvs.tuMillennia Variable

Pascal

```
tuMillennia: TConvType;
```

C++

```
TConvType tuMillennia;
```

File

StdConvs

Description

This is variable StdConvs.tuMillennia.

28.2.82 StdConvs.tuMilliSeconds Variable

Pascal

```
tuMilliSeconds: TConvType;
```

C++

```
TConvType tuMilliSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuMilliSeconds.

28.2.83 StdConvs.tuMinutes Variable

Pascal

```
tuMinutes: TConvType;
```

C++

```
TConvType tuMinutes;
```

File

StdConvs

Description

This is variable StdConvs.tuMinutes.

28.2.84 StdConvs.tuModifiedJulianDate Variable

Pascal

```
tuModifiedJulianDate: TConvType;
```

C++

```
TConvType tuModifiedJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuModifiedJulianDate.

28.2.85 StdConvs.tuMonths Variable

Pascal

```
tuMonths: TConvType;
```

C++

```
TConvType tuMonths;
```

File

StdConvs

Description

This is variable StdConvs.tuMonths.

28.2.86 StdConvs.tuRankine Variable

Pascal

```
tuRankine: TConvType;
```

C++

```
TConvType tuRankine;
```

File

StdConvs

Description

This is variable StdConvs.tuRankine.

28.2.87 StdConvs.tuReaumur Variable

Pascal

```
tuReaumur: TConvType;
```

C++

```
TConvType tuReaumur;
```

File

StdConvs

Description

This is variable StdConvs.tuReaumur.

28.2.88 StdConvs.tuSeconds Variable

Pascal

```
tuSeconds: TConvType;
```

C++

```
TConvType tuSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuSeconds.

28.2.89 StdConvs.tuWeeks Variable

Pascal

```
tuWeeks: TConvType;
```

C++

```
TConvType tuWeeks;
```

File

StdConvs

Description

This is variable StdConvs.tuWeeks.

28.2.90 StdConvs.tuYears Variable

Pascal

```
tuYears: TConvType;
```

C++

```
TConvType tuYears;
```

File

StdConvs

Description

This is variable StdConvs.tuYears.

28.2.91 StdConvs.vuAcreFeet Variable

Pascal

```
vuAcreFeet: TConvType;
```

C++

```
TConvType vuAcreFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreFeet.

28.2.92 StdConvs.vuAcreInches Variable

Pascal

```
vuAcreInches: TConvType;
```

C++

```
TConvType vuAcreInches;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreInches.

28.2.93 StdConvs.vuCentiLiters Variable

Pascal

```
vuCentiLiters: TConvType;
```

C++

```
TConvType vuCentiLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuCentiLiters.

28.2.94 StdConvs.vuCordFeet Variable

Pascal

```
vuCordFeet: TConvType;
```

C++

```
TConvType vuCordFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCordFeet.

28.2.95 StdConvs.vuCords Variable

Pascal

```
vuCords: TConvType;
```

C++

```
TConvType vuCords;
```

File

StdConvs

Description

This is variable StdConvs.vuCords.

28.2.96 StdConvs.vuCubicCentimeters Variable

Pascal

```
vuCubicCentimeters: TConvType;
```

C++

```
TConvType vuCubicCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicCentimeters.

28.2.97 StdConvs.vuCubicDecameters Variable

Pascal

```
vuCubicDecameters: TConvType;
```

C++

```
TConvType vuCubicDecameters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecameters.

28.2.98 StdConvs.vuCubicDecimeters Variable

Pascal

```
vuCubicDecimeters: TConvType;
```

C++

```
TConvType vuCubicDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecimeters.

28.2.99 StdConvs.vuCubicFeet Variable

Pascal

```
vuCubicFeet: TConvType;
```

C++

```
TConvType vuCubicFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicFeet.

28.2.100 StdConvs.vuCubicHectometers Variable

Pascal

```
vuCubicHectometers: TConvType;
```

C++

```
TConvType vuCubicHectometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicHectometers.

28.2.101 StdConvs.vuCubicInches Variable

Pascal

```
vuCubicInches: TConvType;
```

C++

```
TConvType vuCubicInches;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicInches.

28.2.102 StdConvs.vuCubicKilometers Variable

Pascal

```
vuCubicKilometers: TConvType;
```

C++

```
TConvType vuCubicKilometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicKilometers.

28.2.103 StdConvs.vuCubicMeters Variable

Pascal

```
vuCubicMeters: TConvType;
```

C++

```
TConvType vuCubicMeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMeters.

28.2.104 StdConvs.vuCubicMiles Variable

Pascal

```
vuCubicMiles: TConvType;
```

C++

```
TConvType vuCubicMiles;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMiles.

28.2.105 StdConvs.vuCubicMillimeters Variable

Pascal

```
vuCubicMillimeters: TConvType;
```

C++

```
TConvType vuCubicMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMillimeters.

28.2.106 StdConvs.vuCubicYards Variable

Pascal

```
vuCubicYards: TConvType;
```

C++

```
TConvType vuCubicYards;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicYards.

28.2.107 StdConvs.vuDecaLiters Variable

Pascal

```
vuDecaLiters: TConvType;
```

C++

```
TConvType vuDecaLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDecaLiters.

28.2.108 StdConvs.vuDecasteres Variable

Pascal

```
vuDecasteres: TConvType;
```

C++

```
TConvType vuDecasteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecasteres.

28.2.109 StdConvs.vuDeciLiters Variable

Pascal

```
vuDeciLiters: TConvType;
```

C++

```
TConvType vuDeciLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDeciLiters.

28.2.110 StdConvs.vuDecisteres Variable

Pascal

```
vuDecisteres: TConvType;
```

C++

```
TConvType vuDecisteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecisteres.

28.2.111 StdConvs.vuDryBuckets Variable

Pascal

```
vuDryBuckets: TConvType;
```

C++

```
TConvType vuDryBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBuckets.

28.2.112 StdConvs.vuDryBushels Variable

Pascal

```
vuDryBushels: TConvType;
```

C++

```
TConvType vuDryBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBushels.

28.2.113 StdConvs.vuDryGallons Variable

Pascal

```
vuDryGallons: TConvType;
```

C++

```
TConvType vuDryGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuDryGallons.

28.2.114 StdConvs.vuDryPecks Variable

Pascal

```
vuDryPecks: TConvType;
```

C++

```
TConvType vuDryPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPecks.

28.2.115 StdConvs.vuDryPints Variable

Pascal

```
vuDryPints: TConvType;
```

C++

```
TConvType vuDryPints;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPints.

28.2.116 StdConvs.vuDryQuarts Variable

Pascal

```
vuDryQuarts: TConvType;
```

C++

```
TConvType vuDryQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuDryQuarts.

28.2.117 StdConvs.vuFluidCups Variable

Pascal

```
vuFluidCups: TConvType;
```

C++

```
TConvType vuFluidCups;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidCups.

28.2.118 StdConvs.vuFluidGallons Variable

Pascal

```
vuFluidGallons: TConvType;
```

C++

```
TConvType vuFluidGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGallons.

28.2.119 StdConvs.vuFluidGills Variable

Pascal

```
vuFluidGills: TConvType;
```

C++

```
TConvType vuFluidGills;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGills.

28.2.120 StdConvs.vuFluidOunces Variable

Pascal

```
vuFluidOunces: TConvType;
```

C++

```
TConvType vuFluidOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidOunces.

28.2.121 StdConvs.vuFluidPints Variable

Pascal

```
vuFluidPints: TConvType;
```

C++

```
TConvType vuFluidPints;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidPints.

28.2.122 StdConvs.vuFluidQuarts Variable

Pascal

```
vuFluidQuarts: TConvType;
```

C++

```
TConvType vuFluidQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidQuarts.

28.2.123 StdConvs.vuFluidTablespoons Variable

Pascal

```
vuFluidTablespoons: TConvType;
```

C++

```
TConvType vuFluidTablespoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTablespoons.

28.2.124 StdConvs.vuFluidTeaspoons Variable

Pascal

```
vuFluidTeaspoons: TConvType;
```

C++

```
TConvType vuFluidTeaspoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTeaspoons.

28.2.125 StdConvs.vuHectoLiters Variable

Pascal

```
vuHectoLiters: TConvType;
```

C++

```
TConvType vuHectoLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuHectoLiters.

28.2.126 StdConvs.vuKiloLiters Variable

Pascal

```
vuKiloLiters: TConvType;
```

C++

```
TConvType vuKiloLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuKiloLiters.

28.2.127 StdConvs.vuLiters Variable

Pascal

```
vuLiters: TConvType;
```

C++

```
TConvType vuLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuLiters.

28.2.128 StdConvs.vuMilliLiters Variable

Pascal

```
vuMilliLiters: TConvType;
```

C++

```
TConvType vuMilliLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuMilliLiters.

28.2.129 StdConvs.vuSteres Variable

Pascal

```
vuSteres: TConvType;
```

C++

```
TConvType vuSteres;
```

File

StdConvs

Description

This is variable StdConvs.vuSteres.

28.2.130 StdConvs.vuUKBuckets Variable

Pascal

```
vuUKBuckets: TConvType;
```

C++

```
TConvType vuUKBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBuckets.

28.2.131 StdConvs.vuUKBushels Variable

Pascal

```
vuUKBushels: TConvType;
```

C++

```
TConvType vuUKBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBushels.

28.2.132 StdConvs.vuUKGallons Variable

Pascal

```
vuUKGallons: TConvType;
```

C++

```
TConvType vuUKGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGallons.

28.2.133 StdConvs.vuUKGills Variable

Pascal

```
vuUKGills: TConvType;
```

C++

```
TConvType vuUKGills;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGills.

28.2.134 StdConvs.vuUKOunces Variable

Pascal

```
vuUKOunces: TConvType;
```

C++

```
TConvType vuUKOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuUKOunces.

28.2.135 StdConvs.vuUKPecks Variable

Pascal

```
vuUKPecks: TConvType;
```

C++

```
TConvType vuUKPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPecks.

28.2.136 StdConvs.vuUKPints Variable

Pascal

```
vuUKPints: TConvType;
```

C++

```
TConvType vuUKPints;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPints.

28.2.137 StdConvs.vuUKPottles Variable

Pascal

```
vuUKPottles: TConvType;
```

C++

```
TConvType vuUKPottles;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPottles.

28.2.138 StdConvs.vuUKQuarts Variable

Pascal

```
vuUKQuarts: TConvType;
```

C++

```
TConvType vuUKQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuUKQuarts.

28.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 1937)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 1937)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 1938)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 1938)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 1938)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 1939)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 1939)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 1939)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 1940)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 1940)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 1940)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 1941)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 1941)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 1941)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 1942)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 1942)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 1942)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 1943)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 1944)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidGallon.
CubicMetersPerUSFluidGill (see page 1945)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 1946)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 1947)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 1947)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 1947)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 1948)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 1948)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 1948)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 1949)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 1949)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 1949)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 1950)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 1950)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 1950)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 1951)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 1951)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 1951)	This is constant StdConvs.MetersPerFurlong.

MetersPerHand (see page 1952)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 1952)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 1952)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 1953)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 1953)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 1953)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 1954)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 1954)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 1954)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 1955)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 1955)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 1955)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 1956)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 1956)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 1956)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 1957)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 1957)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 1957)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 1958)	This is constant StdConvs.SquareMetersPerSquareYard.

28.3.1 StdConvs.CubicMetersPerAcreFoot Constant

Pascal

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

C++

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreFoot.

28.3.2 StdConvs.CubicMetersPerAcreInch Constant

Pascal

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

C++

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreInch.

28.3.3 StdConvs.CubicMetersPerCord Constant

Pascal

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

C++

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCord.

28.3.4 StdConvs.CubicMetersPerCordFoot Constant

Pascal

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

C++

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCordFoot.

28.3.5 StdConvs.CubicMetersPerCubicFoot Constant

Pascal

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

C++

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicFoot.

28.3.6 StdConvs.CubicMetersPerCubicInch Constant

Pascal

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

C++

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicInch.

28.3.7 StdConvs.CubicMetersPerCubicMile Constant

Pascal

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

C++

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicMile.

28.3.8 StdConvs.CubicMetersPerCubicYard Constant

Pascal

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

C++

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicYard.

28.3.9 StdConvs.CubicMetersPerUKBucket Constant

Pascal

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

C++

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBucket.

28.3.10 StdConvs.CubicMetersPerUKBushel Constant

Pascal

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

C++

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBushel.

28.3.11 StdConvs.CubicMetersPerUKGallon Constant

Pascal

```
CubicMetersPerUKGallon = 0.00454609;
```

C++

```
CubicMetersPerUKGallon = 0.00454609;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGallon.

28.3.12 StdConvs.CubicMetersPerUKGill Constant

Pascal

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

C++

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGill.

28.3.13 StdConvs.CubicMetersPerUKOunce Constant

Pascal

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

C++

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKOunce.

28.3.14 StdConvs.CubicMetersPerUKPeck Constant

Pascal

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

C++

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPeck.

28.3.15 StdConvs.CubicMetersPerUKPint Constant

Pascal

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

C++

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPint.

28.3.16 StdConvs.CubicMetersPerUKPottle Constant

Pascal

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

C++

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPottle.

28.3.17 StdConvs.CubicMetersPerUKQuart Constant

Pascal

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

C++

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKQuart.

28.3.18 StdConvs.CubicMetersPerUSDryBucket Constant

Pascal

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

C++

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBucket.

28.3.19 StdConvs.CubicMetersPerUSDryBushel Constant

Pascal

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

C++

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBushel.

28.3.20 StdConvs.CubicMetersPerUSDryGallon Constant

Pascal

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

C++

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryGallon.

28.3.21 StdConvs.CubicMetersPerUSDryPeck Constant

Pascal

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

C++

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPeck.

28.3.22 StdConvs.CubicMetersPerUSDryPint Constant

Pascal

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

C++

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPint.

28.3.23 StdConvs.CubicMetersPerUSDryQuart Constant

Pascal

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

C++

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryQuart.

28.3.24 StdConvs.CubicMetersPerUSFluidCup Constant

Pascal

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

C++

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidCup.

28.3.25 StdConvs.CubicMetersPerUSFluidGallon Constant

Pascal

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

C++

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGallon.

28.3.26 StdConvs.CubicMetersPerUSFluidGill Constant

Pascal

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

C++

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGill.

28.3.27 StdConvs.CubicMetersPerUSFluidOunce Constant

Pascal

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

C++

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidOunce.

28.3.28 StdConvs.CubicMetersPerUSFluidPint Constant

Pascal

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

C++

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidPint.

28.3.29 StdConvs.CubicMetersPerUSFluidQuart Constant

Pascal

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

C++

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidQuart.

28.3.30 StdConvs.CubicMetersPerUSFluidTablespoon Constant

Pascal

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

C++

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTablespoon.

28.3.31 StdConvs.CubicMetersPerUSFluidTeaspoon Constant

Pascal

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

C++

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.

28.3.32 StdConvs.GramsPerDrams Constant

Pascal

```
GramsPerDrams = GramsPerPound / 256;
```

C++

```
GramsPerDrams = GramsPerPound / 256;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerDrams.

28.3.33 StdConvs.GramsPerGrains Constant

Pascal

```
GramsPerGrains = GramsPerPound / 7000;
```

C++

```
GramsPerGrains = GramsPerPound / 7000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerGrains.

28.3.34 StdConvs.GramsPerLongTons Constant

Pascal

```
GramsPerLongTons = GramsPerPound * 2240;
```

C++

```
GramsPerLongTons = GramsPerPound * 2240;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerLongTons.

28.3.35 StdConvs.GramsPerOunces Constant

Pascal

```
GramsPerOunces = GramsPerPound / 16;
```

C++

```
GramsPerOunces = GramsPerPound / 16;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerOunces.

28.3.36 StdConvs.GramsPerPound Constant

Pascal

```
GramsPerPound = 453.59237;
```

C++

```
GramsPerPound = 453.59237;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerPound.

28.3.37 StdConvs.GramsPerStones Constant

Pascal

```
GramsPerStones = GramsPerPound * 14;
```

C++

```
GramsPerStones = GramsPerPound * 14;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerStones.

28.3.38 StdConvs.GramsPerTons Constant

Pascal

```
GramsPerTons = GramsPerPound * 2000;
```

C++

```
GramsPerTons = GramsPerPound * 2000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerTons.

28.3.39 StdConvs.MetersPerAstronomicalUnit Constant

Pascal

```
MetersPerAstronomicalUnit = 1.49598E11;
```

C++

```
MetersPerAstronomicalUnit = 1.49598E11;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerAstronomicalUnit.

28.3.40 StdConvs.MetersPerChain Constant

Pascal

```
MetersPerChain = MetersPerRod * 4;
```

C++

```
MetersPerChain = MetersPerRod * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerChain.

28.3.41 StdConvs.MetersPerCubit Constant

Pascal

```
MetersPerCubit = 0.4572;
```

C++

```
MetersPerCubit = 0.4572;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerCubit.

28.3.42 StdConvs.MetersPerFathom Constant

Pascal

```
MetersPerFathom = MetersPerFoot * 6;
```

C++

```
MetersPerFathom = MetersPerFoot * 6;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFathom.

28.3.43 StdConvs.MetersPerFoot Constant

Pascal

```
MetersPerFoot = MetersPerInch * 12;
```

C++

```
MetersPerFoot = MetersPerInch * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFoot.

28.3.44 StdConvs.MetersPerFurlong Constant

Pascal

```
MetersPerFurlong = MetersPerYard * 220;
```

C++

```
MetersPerFurlong = MetersPerYard * 220;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFurlong.

28.3.45 StdConvs.MetersPerHand Constant

Pascal

```
MetersPerHand = MetersPerInch * 4;
```

C++

```
MetersPerHand = MetersPerInch * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerHand.

28.3.46 StdConvs.MetersPerInch Constant

Pascal

```
MetersPerInch = 0.0254;
```

C++

```
MetersPerInch = 0.0254;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerInch.

28.3.47 StdConvs.MetersPerLightSecond Constant

Pascal

```
MetersPerLightSecond = 2.99792458E8;
```

C++

```
MetersPerLightSecond = 2.99792458E8;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightSecond.

28.3.48 StdConvs.MetersPerLightYear Constant

Pascal

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

C++

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightYear.

28.3.49 StdConvs.MetersPerLink Constant

Pascal

```
MetersPerLink = MetersPerChain / 100;
```

C++

```
MetersPerLink = MetersPerChain / 100;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLink.

28.3.50 StdConvs.MetersPerMile Constant

Pascal

```
MetersPerMile = MetersPerFoot * 5280;
```

C++

```
MetersPerMile = MetersPerFoot * 5280;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerMile.

28.3.51 StdConvs.MetersPerNauticalMiles Constant

Pascal

```
MetersPerNauticalMiles = 1852;
```

C++

```
MetersPerNauticalMiles = 1852;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerNauticalMiles.

28.3.52 StdConvs.MetersPerPace Constant

Pascal

```
MetersPerPace = MetersPerInch * 30;
```

C++

```
MetersPerPace = MetersPerInch * 30;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPace.

28.3.53 StdConvs.MetersPerParsec Constant

Pascal

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

C++

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerParsec.

28.3.54 StdConvs.MetersPerPica Constant

Pascal

```
MetersPerPica = MetersPerPoint * 12;
```

C++

```
MetersPerPica = MetersPerPoint * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPica.

28.3.55 StdConvs.MetersPerPoint Constant

Pascal

```
MetersPerPoint = MetersPerInch * 0.013837;
```

C++

```
MetersPerPoint = MetersPerInch * 0.013837;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPoint.

28.3.56 StdConvs.MetersPerRod Constant

Pascal

```
MetersPerRod = MetersPerFoot * 16.5;
```

C++

```
MetersPerRod = MetersPerFoot * 16.5;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerRod.

28.3.57 StdConvs.MetersPerYard Constant

Pascal

```
MetersPerYard = MetersPerFoot * 3;
```

C++

```
MetersPerYard = MetersPerFoot * 3;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerYard.

28.3.58 StdConvs.SquareMetersPerAcre Constant

Pascal

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

C++

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerAcre.

28.3.59 StdConvs.SquareMetersPerSquareFoot Constant

Pascal

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

C++

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareFoot.

28.3.60 StdConvs.SquareMetersPerSquareInch Constant

Pascal

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

C++

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareInch.

28.3.61 StdConvs.SquareMetersPerSquareMile Constant

Pascal

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

C++

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareMile.

28.3.62 StdConvs.SquareMetersPerSquareRod Constant

Pascal

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

C++

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareRod.

28.3.63 StdConvs.SquareMetersPerSquareYard Constant

Pascal

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

C++

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareYard.

29 StdCtrls Namespace

This is namespace StdCtrls.

Topics

Name	Description
Classes (see page 1961)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 2906)	The following table lists structs, records, enums in this documentation.
Types (see page 2915)	The following table lists types in this documentation.

Classes

	Name	Description
	TButton (see page 1961)	TButton is a push button control.
	TButtonActionLink (see page 2030)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 2033)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 2040)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 2106)	TComboBox combines an edit box with a scrollable list.
	TCustomButton (see page 2179)	Base class for a push button control.
	TCustomCheckBox (see page 2196)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 2203)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 2241)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 2257)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 2263)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 2299)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 2302)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 2316)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 2361)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2649) and TDBMemo.
	TCustomStaticText (see page 2375)	TCustomStaticText is the base class for TStaticText (see page 2853).
	TEdit (see page 2380)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 2451)	TGroupBox represents a Windows group box.
	TImageMargins (see page 2517)	Margins of image on button.
	TLabel (see page 2522)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 2572)	TListBox displays a collection of items in a scrollable list.

	TMemo (see page 2649)	TMemo is a wrapper for a Windows multiline edit control.
	TPushButtonActionLink (see page 2722)	This is class StdCtrls.TPushButtonActionLink.
	TRadioButton (see page 2724)	TRadioButton is a wrapper for a system radio button.
	TScrollBar (see page 2790)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 2853)	TStaticText is a windowed control that displays text on a form.

Structs, Records, Enums

	Name	Description
	TCheckBoxState (see page 2906)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 2907)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls unit.
	TEditCharCase (see page 2907)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 2908)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TImageAlignment (see page 2909)	Alignment of image on TButton (see page 1961).
	TListBoxStyle (see page 2910)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 2911)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 2912)	Identifies a set of scrollbars by their orientation.
	TSelection (see page 2913)	TSelection definese the start and position of a selection.
	TStaticBorderStyle (see page 2913)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 2914)	TTextLayout indicates where text appears within the client area of a control.

Types

Name	Description
TButtonActionLinkClass (see page 2915)	TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2030).
TCustomComboBoxStringsClass (see page 2915)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2257).
TDrawItemEvent (see page 2916)	TScrollEvent (see page 2920) is the type of an OnScroll event handler.
TFNDrawText (see page 2916)	This is type StdCtrls.TFNDrawText.
TLBFindDataEvent (see page 2917)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 2917)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 2918)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 2918)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 2918)	Indicates state information that can influence how an item is drawn.
TPushButtonActionLinkClass (see page 2919)	This is type StdCtrls.TPushButtonActionLinkClass.
TScrollEvent (see page 2920)	This is type StdCtrls.TScrollEvent.

29.1 Classes

The following table lists classes in this documentation.

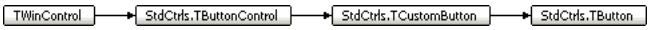
Classes

	Name	Description
	TButton (see page 1961)	TButton is a push button control.
	TButtonActionLink (see page 2030)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 2033)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 2040)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 2106)	TComboBox combines an edit box with a scrollable list.
	TCustomButton (see page 2179)	Base class for a push button control.
	TCustomCheckBox (see page 2196)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 2203)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 2241)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 2257)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 2263)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 2299)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 2302)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 2316)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 2361)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2649) and TDBMemo.
	TCustomStaticText (see page 2375)	TCustomStaticText is the base class for TStaticText (see page 2853).
	TEdit (see page 2380)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 2451)	TGroupBox represents a Windows group box.
	TImageMargins (see page 2517)	Margins of image on button.
	TLabel (see page 2522)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 2572)	TListBox displays a collection of items in a scrollable list.
	TMemo (see page 2649)	TMemo is a wrapper for a Windows multiline edit control.
	TPushButtonActionLink (see page 2722)	This is class StdCtrls.TPushButtonActionLink.
	TRadioButton (see page 2724)	TRadioButton is a wrapper for a system radio button.
	TScrollBar (see page 2790)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 2853)	TStaticText is a windowed control that displays text on a form.

29.1.1 TButton Class

TButton is a push button control.

Class Hierarchy



Pascal

```
TButton = class(TCustomButton);
```

C++

```
class TButton : public TCustomButton;
```

File

StdCtrls

Description

Use TButton to put a standard push button on a form. TButton introduces several properties to control its behavior in a dialog box setting. Users choose button controls to initiate actions.

To use a button that displays a bitmap instead of a label, use TBitBtn. To use a button that can remain in a depressed position, use TSpeedButton.

Note: Since the TButton caption is always centered, changing the BiDi alignment has no effect.

See Also

TBitBtn

TCustomButton (see page 2179)

TSpeedButton

Button Controls

Buttons and Similar Controls

29.1.1.1 TButton Members

The following tables list the members exposed by TButton.

Enumerations

	Name	Description
	TButtonStyle (see page 2181)	Style (see page 2194) of TButton (see page 1961).

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	ActionChange (see page 2182)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2035)	Initializes a window-creation parameter data structure.
	CreateParams (see page 2183)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2184)	Creates the underlying window.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetActionLinkClass (see page 2184)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2036)	Returns whether the button control is checked.
	Loaded (see page 2184)	Initializes the control after it is loaded from a stream.
	Notification (see page 2185)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2185)	This is SetButtonStyle, a member of class TCustomButton.
	SetChecked (see page 2036)	Sets the Checked (see page 2037) property.
	UpdateImageList (see page 2186)	This is UpdateImageList, a member of class TCustomButton.
	UpdateImages (see page 2186)	This is UpdateImages, a member of class TCustomButton.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2037)	Specifies whether the button control is checked.

	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Click (see page 2183)	Simulates a mouse click, as if the user had clicked the button.
	Create (see page 2034)	Creates a button control.
	Create (see page 2180)	Creates and initializes a button instance.
	Destroy (see page 2181)	Destroys an instance of TWinControl.
	UseRightToLeftAlignment (see page 2186)	Specifies whether the control's alignment is in a right-to-left mode.

Published Events

	Name	Description
	OnClick (see page 1998)	Occurs when the user clicks the control.
	OnContextPopup (see page 2000)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2001)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2004)	Occurs when the user drags an object over a control.
	OnDropDownClick (see page 2007)	Down arrow part of Vista style split button is clicked.
	OnEndDock (see page 2008)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2008)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2009)	Occurs when a control receives the input focus.
	OnExit (see page 2011)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2014)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2018)	Occurs when a key is pressed.
	OnKeyUp (see page 2019)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2021)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2022)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2024)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2024)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2024)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2027)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2029)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2030)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 1965)	Designates the action associated with the control.
	Align (see page 1966)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1969)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 1970)	Specifies the bi-directional mode for the control.
	Cancel (see page 1970)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 1971)	Specifies a text string that identifies the control to the user.
	CommandLinkHint (see page 1973)	Text displayed as hint below button caption for Command Link.
	Constraints (see page 1973)	Specifies the size constraints for the control.
	Default (see page 1974)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 1975)	Image index for disabled button state.
	DoubleBuffered (see page 1976)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	DragCursor (see page 1977)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1978)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1978)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownMenu (see page 1980)	Split button drop down menu.
	ElevationRequired (see page 1980)	Put shield icon on button indicating elevated access rights required.
	Enabled (see page 1981)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1981)	Controls the attributes of text written on or in the control.
	HotImageIndex (see page 1982)	Image index for hot button state.
	ImageAlignment (see page 1983)	Alignment of image on button.
	ImageIndex (see page 1983)	Image index for normal button state.
	ImageMargins (see page 1984)	Margins of image on button.
	Images (see page 1984)	List of images for a button.
	ModalResult (see page 1985)	Determines whether and how the button closes its (modal) parent form.
	ParentBiDiMode (see page 1985)	Specifies whether the control uses its parent's BiDiMode.
	ParentDoubleBuffered (see page 1986)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1986)	Determines where a control looks for its font information.
	ParentShowHint (see page 1987)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1987)	Identifies the pop-up menu associated with the control.
	PressedImageIndex (see page 1992)	Image index for pressed button state.
	SelectedImageIndex (see page 1993)	Image index for selected button state.
	ShowHint (see page 1993)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 1995)	Style of a button.
	TabOrder (see page 1995)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1996)	Determines if the user can tab to a control.
	Visible (see page 1996)	Determines whether the component appears onscreen.
	WordWrap (see page 1997)	Specifies whether the button text wraps to fit the width of the control.

29.1.1.2 TButton Properties

The properties of the TButton class are listed here.

Published Properties

	Name	Description
	Action (see page 1965)	Designates the action associated with the control.
	Align (see page 1966)	Determines how the control aligns within its container (parent control).
	Anchors (see page 1969)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 1970)	Specifies the bi-directional mode for the control.
	Cancel (see page 1970)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 1971)	Specifies a text string that identifies the control to the user.
	CommandLinkHint (see page 1973)	Text displayed as hint below button caption for Command Link.
	Constraints (see page 1973)	Specifies the size constraints for the control.
	Default (see page 1974)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 1975)	Image index for disabled button state.
	DoubleBuffered (see page 1976)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 1977)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 1978)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 1978)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownMenu (see page 1980)	Split button drop down menu.

	ElevationRequired (see page 1980)	Put shield icon on button indicating elevated access rights required.
	Enabled (see page 1981)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1981)	Controls the attributes of text written on or in the control.
	HotImageIndex (see page 1982)	Image index for hot button state.
	ImageAlignment (see page 1983)	Alignment of image on button.
	ImageIndex (see page 1983)	Image index for normal button state.
	ImageMargins (see page 1984)	Margins of image on button.
	Images (see page 1984)	List of images for a button.
	ModalResult (see page 1985)	Determines whether and how the button closes its (modal) parent form.
	ParentBiDiMode (see page 1985)	Specifies whether the control uses its parent's BiDiMode.
	ParentDoubleBuffered (see page 1986)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 1986)	Determines where a control looks for its font information.
	ParentShowHint (see page 1987)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 1987)	Identifies the pop-up menu associated with the control.
	PressedImageIndex (see page 1992)	Image index for pressed button state.
	SelectedImageIndex (see page 1993)	Image index for selected button state.
	ShowHint (see page 1993)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 1995)	Style of a button.
	TabOrder (see page 1995)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 1996)	Determines if the user can tab to a control.
	Visible (see page 1996)	Determines whether the component appears onscreen.
	WordWrap (see page 1997)	Specifies whether the button text wraps to fit the width of the control.

29.1.1.2.1 TButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

29.1.1.2.2 TButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
```

```
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );
```

```
procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

29.1.1.2.3 TButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.1.2.4 TButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.1.2.5 TButton.Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

See Also

Default (see page 2188)

OnClick

29.1.1.2.6 TButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
```

```
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
```

```
    Const ButtonCaptions: array of String);
  var
    I: Integer;
  begin
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

29.1.1.2.7 TButton.CommandLinkHint Property

Text displayed as hint below button caption for Command Link.

Pascal

```
property CommandLinkHint: string;
```

C++

```
__property UnicodeString CommandLinkHint;
```

Description

CommandLinkHint is the text displayed as a hint below the button caption when the Style property is **bsCommandLink** to specify a Command Link button.

This is available on Windows Vista only.

See Also

Style (see page 2194)

TButtonStyle

29.1.1.2.8 TButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.1.2.9 TButton.Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Cancel (see page 2187)

OnClick

29.1.1.2.10 TButton.DisabledImageIndex Property

Image index for disabled button state.

Pascal

```
property DisabledImageIndex: TImageIndex;
```

C++

```
__property TImageIndex DisabledImageIndex;
```

Description

DisabledImageIndex is an image index into the Images property for the disabled button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.1.2.11 TButton.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.1.2.12 TButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.1.2.13 TButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.1.2.14 TButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.1.2.15 TButton.DropDownMenu Property

Split button drop down menu.

Pascal

```
property DropDownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropDownMenu;
```

Description

When the Style property is **bsSplitButton**, the button is a Vista style split button with a drop down menu. DropDownMenu specifies the drop down menu for the down arrow button part. The OnDropDownClick event also occurs when the down arrow button part is clicked.

This is available on Windows Vista only.

See Also

TPopupMenu

Style (see page 2194)

OnDropDownClick (see page 2195)

TButtonStyle

29.1.1.2.16 TButton.ElevationRequired Property

Put shield icon on button indicating elevated access rights required.

Pascal

```
property ElevationRequired: Boolean;
```

C++

```
__property Boolean ElevationRequired;
```

Description

When ElevationRequired is true, the button has a shield icon, indicating that elevated access rights are required. Vista UI guidelines specify that when there is a button that performs an operation requiring elevated access rights, it should have this shield icon.

This icon overrides any of the properties specifying a button image for various button states, as in ImageIndex.

This is available on Windows Vista only.

See Also

ImageIndex (see page 2191)

Images (see page 2192)

29.1.1.2.17 TButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.1.2.18 TButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
```

```
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.1.2.19 TButton.HotImageIndex Property

Image index for hot button state.

Pascal

```
property HotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex HotImageIndex;
```

Description

HotImageIndex is an image index into the Images property for the hot button state, i.e., the cursor hovering over the button. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.1.2.20 TButton.ImageAlignment Property

Alignment of image on button.

Pascal

```
property ImageAlignment: TImageAlignment;
```

C++

```
__property TImageAlignment ImageAlignment;
```

Description

TImageAlignment indicates the alignment of an image on a button. It is a TImageAlignment type.

See Also

ImageMargins (see page 2192)

Images (see page 2192)

TImageAlignment (see page 2909)

29.1.1.2.21 TButton.ImageIndex Property

Image index for normal button state.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an image index into the Images property for the normal button state. If the index value is -1, no image is used.

In Vista, a standard green arrow is used when the index is -1 and the Style property is **bsCommandLink** for a Command Link button.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

Style (see page 2194)

TButtonStyle

29.1.1.2.22 TButton.ImageMargins Property

Margins of image on button.

Pascal

```
property ImageMargins: TImageMargins;
```

C++

```
__property TImageMargins ImageMargins;
```

Description

A button image may be positioned using margins. ImageMargins holds aTImageMargins object containing the margins of an image on a button. The TImageMargins.OnChange event is triggered when a margin changes.

See Also

ImageAlignment (see page 2191)

TImageMargins (see page 2517)

Images (see page 2192)

OnChange (see page 2521)

29.1.1.2.23 TButton.Images Property

List of images for a button.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a TCustomImageList object listing the various images that can be placed on a button.

The button's image depends on the button state. These images are determined by the value of the image state properties that index ImageIndex: DisabledImageIndex, HotImageIndex, ImageIndex, PressedImageIndex, and SelectedImageIndex. When the ImageIndex index is -1, it generally means that no image is used for the normal state. For the other state image indexes, -1 means to use the same image as the normal state.

The ImageMargins and ImageAlignment properties orient the image inside the button. The image is not sized to fit the button, and the button is not stretched.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageAlignment (see page 2191)

ImageIndex (see page 2191)

ImageMargins (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.1.2.24 TButton.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the button component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick event handlers are required for the buttons.

The following constants are of TModalResult type and defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.
mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.
mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.
mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.
mrClose	mrYesToAll + 1	The user exited with the CLOSE button.

See Also

TModalResult

29.1.1.2.25 TButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.1.2.26 TButton.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.1.2.27 TButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.1.2.28 TButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.1.2.29 TButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
```

```
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.1.2.30 TButton.PressedImageIndex Property

Image index for pressed button state.

Pascal

```
property PressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex PressedImageIndex;
```

Description

PressedImageIndex is an image index into the Images property for the pressed button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

SelectedImageIndex (see page 2194)

29.1.1.2.31 TButton.SelectedImageIndex Property

Image index for selected button state.

Pascal

```
property SelectedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex SelectedImageIndex;
```

Description

SelectedImageIndex is an image index into the Images property for the selected button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

29.1.1.2.32 TButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.1.2.33 TButton.Style Property

Style of a button.

Pascal

```
property Style: TButtonStyle;
```

C++

```
__property TButtonStyle Style;
```

Description

Style indicates a button's style, which is a TButtonStyle type.

See Also

TButtonStyle

29.1.1.2.34 TButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.1.2.35 TButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.1.2.36 TButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
```

```
    {
      TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this); // The owner will clean this up.
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.1.2.37 TButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

29.1.1.3 TButton Events

The events of the TButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 1998)	Occurs when the user clicks the control.
	OnContextPopup (see page 2000)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2001)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2004)	Occurs when the user drags an object over a control.
	OnDropDownClick (see page 2007)	Down arrow part of Vista style split button is clicked.
	OnEndDock (see page 2008)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2008)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2009)	Occurs when a control receives the input focus.
	OnExit (see page 2011)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2014)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2018)	Occurs when a key is pressed.
	OnKeyUp (see page 2019)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2021)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2022)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2024)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2024)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2024)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2027)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2029)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2030)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.1.3.1 TButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.1.3.2 TButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.1.3.3 TButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.1.3.4 TButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
```

```
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.1.3.5 TButton.OnDropDownClick Event

Down arrow part of Vista style split button is clicked.

Pascal

```
property OnDropDownClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDownClick;
```

Description

OnDropDownClick occurs when the down arrow part of a Vista style split button is clicked. This only applies when the button's Style property is **bsSplitButton**.

This is available on Windows Vista only.

See Also

DropDownMenu (see page 2189)

Style (see page 2194)

TButtonStyle

29.1.1.3.6 TButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.1.3.7 TButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.1.3.8 TButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.1.3.9 TButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
```

```
  to contain the empty string, to use the Courier font and to
  use a particular mask, corresponding to IP address coding convetions.
  The mask edit boxes display the IP address, the subnet mask
  and the subnet address correspondingly. When the user exits the
  subnet mask edit box, the subnet address is automatically calculated
  using bit-wise AND operations and displayed in the third mask edit box.
  An additional test is made to assure that the values in the IP address
  and in the subnet mask are in the interval from 0 to 255,
  as this validation test cannot be accomplished
  using the edit mask property alone.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { initialize the maskedit boxes to contain the empty string }
    MaskEdit1.Text := '';
    MaskEdit2.Text := '';
    MaskEdit3.Text := '';

    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
```

```
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

29.1.1.3.10 TButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse

buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
```

```
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;
```

```
procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
```

```
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.1.3.11 TButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/
```

```
void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.1.3.12 TButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
```

```
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.1.3.13 TButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.1.3.14 TButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.1.3.15 TButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.1.3.16 TButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.1.3.17 TButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.1.3.18 TButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.1.3.19 TButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.1.3.20 TButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

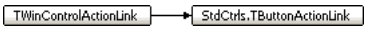
OnDragOver

OnDragDrop

29.1.2 TButtonActionLink Class

TButtonActionLink is an action link designed specifically for button control clients.

Class Hierarchy



Pascal

```
TButtonActionLink = class(TWinControlActionLink);
```

C++

```
class TButtonActionLink : public TWinControlActionLink;
```

File

StdCtrls

Description

TButtonActionLink is designed to specifically link actions to button control clients. TButtonActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of button control clients.

TButtonActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TButtonActionLink adds support for linking the checked property to the appropriate property of a button control.

29.1.2.1 TButtonActionLink Members

The following tables list the members exposed by TButtonActionLink.

Protected Fields

	Name	Description
	FClient (see page 2031)	This is FClient, a member of class TButtonActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2032)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 2032)	Indicates whether the checked state for the action and the client are linked.
	SetChecked (see page 2032)	Sets the checked state of a button control if the client is linked to the action for this property.

29.1.2.2 TButtonActionLink Fields

The fields of the TButtonActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 2031)	This is FClient, a member of class TButtonActionLink.

29.1.2.2.1 TButtonActionLink.FClient Field

Pascal

```
FClient: TButtonControl;
```

C++

```
TButtonControl FClient;
```

Description

This is FClient, a member of class TButtonActionLink.

29.1.2.3 TButtonActionLink Methods

The methods of the TButtonActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2032)	Assigns a client button that is linked to the associated action.

	IsCheckedLinked (see page 2032)	Indicates whether the checked state for the action and the client are linked.
	SetChecked (see page 2032)	Sets the checked state of a button control if the client is linked to the action for this property.

29.1.2.3.1 TButtonActionLink.AssignClient Method

Assigns a client button that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TButtonActionLink (see page 2030) overrides AssignClient to ensure that the client for the action is a button control.

See Also

Action

29.1.2.3.2 TButtonActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked

Checked (see page 2037)

SetChecked (see page 2032)

29.1.2.3.3 TButtonActionLink.SetChecked Method

Sets the checked state of a button control if the client is linked to the action for this property.

Pascal

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

C++

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

Description

SetChecked sets the checked state of a button control, and is called only if the client is linked for this property. Value specifies whether the checked state should be set.

See Also

Checked

Checked (see page 2037)

IsCheckedLinked (see page 2032)

29.1.3 TButtonControl Class

TButtonControl is the base class for objects that represent Windows button controls.

Class Hierarchy

Pascal

```
TButtonControl = class(TWinControl);
```

C++

```
class TButtonControl : public TWinControl;
```

File

StdCtrls

Description

TButtonControl encapsulates behavior common to button controls, check boxes, and radio buttons. Use TButtonControl as a base class when defining a button that is a wrapper for a Windows button. Do not create instances of TButtonControl directly in an application. Instead, use a descendant of TButtonControl to put a push button, check box, or radio button on a form.

Note: Some buttons, such as TSpeedButton, are not wrappers for Windows buttons. These controls do not descend from TButtonControl.

See Also

TButton (see page 1961)

TCustomCheckBox (see page 2196)

TRadioButton (see page 2724)

TSpeedButton

29.1.3.1 TButtonControl Members

The following tables list the members exposed by TButtonControl.

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2035)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2036)	Returns whether the button control is checked.
	SetChecked (see page 2036)	Sets the Checked (see page 2037) property.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2037)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.
	WordWrap (see page 2039)	Specifies whether the button text wraps to fit the width of the control.

Public Methods

	Name	Description
	Create (see page 2034)	Creates a button control.

29.1.3.2 TButtonControl.Create Constructor

Creates a button control.

Pascal

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

C++

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

Description

AOwner is the component that is responsible for freeing the button. It becomes the value of the Owner property.

29.1.3.3 TButtonControl Methods

The methods of the TButtonControl class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2035)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2036)	Returns whether the button control is checked.
	SetChecked (see page 2036)	Sets the Checked (see page 2037) property.
	WndProc (see page 2036)	Handles Windows messages.

29.1.3.3.1 TButtonControl.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked (see page 2037) state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers

should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction

29.1.3.3.2 TButtonControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

29.1.3.3.3 TButtonControl.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink (see page 2030) class type.

See Also

TButtonActionLink (see page 2030)

29.1.3.3.4 TButtonControl.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked (see page 2037) property.

See Also

Checked (see page 2037)

29.1.3.3.5 TButtonControl.SetChecked Method

Sets the Checked (see page 2037) property.

Pascal

```
procedure SetChecked(Value: Boolean); virtual;
```

C++

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

Description

SetChecked is the write implementer for the Checked (see page 2037) property.

See Also

Checked (see page 2037)

29.1.3.3.6 TButtonControl.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc enables button objects to receive input focus when WM_LBUTTONDOWN and WM_LBUTTONDBLCLK messages occur.

See Also

WndProc

29.1.3.4 TButtonControl Properties

The properties of the TButtonControl class are listed here.

Protected Properties

	Name	Description
	Checked (see page 2037)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.
	WordWrap (see page 2039)	Specifies whether the button text wraps to fit the width of the control.

29.1.3.4.1 TButtonControl.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1); // PageControl1 is the parent, so it will
clean up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
```

```
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

WndProc

29.1.3.4.2 TButtonControl.ClicksDisabled Property

Specifies whether the button control accepts mouse clicks.

Pascal

```
property ClicksDisabled: Boolean;
```

C++

```
__property Boolean ClicksDisabled;
```

Description

Use ClicksDisabled to determine whether a button control accepts mouse clicks. If clicksdisabled is true, the button control doesn't accept mouse clicks. Otherwise, the button control accepts mouse clicks.

See Also

WndProc

29.1.3.4.3 TButtonControl.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

29.1.4 TCheckBox Class

TCheckBox represents a check box that can be on (checked) or off (unchecked)

Class Hierarchy

Pascal

```
TCheckBox = class(TCustomCheckBox);
```

C++

```
class TCheckBox : public TCustomCheckBox;
```

File

StdCtrls

Description

A TCheckBox component presents an option for the user. The user can check the box to select the option, or uncheck it to deselect the option.

See Also

TCustomCheckBox (see page 2196)

Buttons and Similar Controls

Check Boxes

29.1.4.1 TCheckBox Members

The following tables list the members exposed by TCheckBox.

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	Click (see page 2197)	Click generates an OnClick event.
	CreateParams (see page 2035)	Initializes a window-creation parameter data structure.
	CreateParams (see page 2198)	Initializes a window-creation parameter record.
	CreateWnd (see page 2198)	Creates a Windows control corresponding to the check box.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2036)	Returns whether the button control is checked.
	GetChecked (see page 2199)	Returns the value of the Checked (see page 2037) property.
	SetChecked (see page 2036)	Sets the Checked (see page 2037) property.
	SetChecked (see page 2199)	Sets the value of the Checked (see page 2037) property.
	Toggle (see page 2200)	Changes the state of the check box.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Create (see page 2034)	Creates a button control.
	Create (see page 2197)	Creates and initializes a TCustomCheckBox (see page 2196) instance.
	GetControlsAlignment (see page 2199)	Returns the alignment of the check box caption.

Published Events

	Name	Description
	OnClick (see page 2074)	Occurs when the user clicks the control.
	OnContextPopup (see page 2076)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2077)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2080)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2083)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2084)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2085)	Occurs when a control receives the input focus.
	OnExit (see page 2087)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2090)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2093)	Occurs when a key is pressed.
	OnKeyUp (see page 2095)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2096)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2097)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2100)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2100)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2100)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2102)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2105)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2105)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2043)	Designates the action associated with the control.
	Align (see page 2043)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2047)	Determines whether the checkbox label aligns to the left or right of the tick box.
	AllowGrayed (see page 2047)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2048)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2049)	Specifies the bi-directional mode for the control.
	Caption (see page 2049)	Specifies a text string that identifies the control to the user.
	Checked (see page 2052)	Specifies whether the button control is checked.
	Color (see page 2054)	Specifies the background color of the control.
	Constraints (see page 2055)	Specifies the size constraints for the control.
	Ctl3D (see page 2055)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2056)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2057)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2058)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2058)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2060)	Controls whether the control responds to mouse, keyboard, and timer events.

	Font (see page 2060)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2061)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2062)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2062)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2062)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2063)	Determines where a control looks for its font information.
	ParentShowHint (see page 2063)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2064)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2069)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 2070)	Indicates whether the check box is selected, deselected, or grayed.
	TabOrder (see page 2071)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2072)	Determines if the user can tab to a control.
	Visible (see page 2072)	Determines whether the component appears onscreen.
	WordWrap (see page 2073)	Specifies whether the button text wraps to fit the width of the control.

29.1.4.2 TCheckBox Properties

The properties of the TCheckBox class are listed here.

Published Properties

	Name	Description
	Action (see page 2043)	Designates the action associated with the control.
	Align (see page 2043)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2047)	Determines whether the checkbox label aligns to the left or right of the tick box.
	AllowGrayed (see page 2047)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2048)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2049)	Specifies the bi-directional mode for the control.
	Caption (see page 2049)	Specifies a text string that identifies the control to the user.
	Checked (see page 2052)	Specifies whether the button control is checked.
	Color (see page 2054)	Specifies the background color of the control.
	Constraints (see page 2055)	Specifies the size constraints for the control.
	Ctl3D (see page 2055)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2056)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2057)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2058)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2058)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2060)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2060)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2061)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2062)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2062)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2062)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2063)	Determines where a control looks for its font information.
	ParentShowHint (see page 2063)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2064)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2069)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 2070)	Indicates whether the check box is selected, deselected, or grayed.

	TabOrder (see page 2071)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2072)	Determines if the user can tab to a control.
	Visible (see page 2072)	Determines whether the component appears onscreen.
	WordWrap (see page 2073)	Specifies whether the button text wraps to fit the width of the control.

29.1.4.2.1 TCheckBox.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

29.1.4.2.2 TCheckBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
```

```
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

29.1.4.2.3 TCheckBox.Alignment Property

Determines whether the checkbox label aligns to the left or right of the tick box.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

The Alignment property is a little confusing. It implies text justification, but it is not quite that simple. The TLeftRight alignment type may have one of the following values :

Value	Meaning
taLeftJustify	The checkbox label is displayed on the left of the checkbox. The label is aligned to the left of the control, and the checkbox is aligned to the right of the control.
taRightJustify	This is the default value. The checkbox is aligned to the left of the control. The label is aligned to the right side of the checkbox : it is not aligned to the right of the control.

See Also

TLeftRight

29.1.4.2.4 TCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked

State (see page 2201)

29.1.4.2.5 TCheckBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.4.2.6 TCheckBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.4.2.7 TCheckBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
```

```
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
```

```
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
```

```
end;
```

29.1.4.2.8 TCheckBox.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1); // PageControl1 is the parent, so it will
clean up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
```

```
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
```

```
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

WndProc

29.1.4.2.9 TCheckBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.4.2.10 TCheckBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.4.2.11 TCheckBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.4.2.12 TCheckBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
```

```
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.4.2.13 TCheckBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.4.2.14 TCheckBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.4.2.15 TCheckBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
```

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.4.2.16 TCheckBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.4.2.17 TCheckBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.4.2.18 TCheckBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.4.2.19 TCheckBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.4.2.20 TCheckBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.4.2.21 TCheckBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered

property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.4.2.22 TCheckBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.4.2.23 TCheckBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.4.2.24 TCheckBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
```

```
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
```

```
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
```

```
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.4.2.25 TCheckBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.4.2.26 TCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
```

```
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 2201)

Checked

29.1.4.2.27 TCheckBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.4.2.28 TCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.4.2.29 TCheckBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.4.2.30 TCheckBox.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

29.1.4.3 TCheckBox Events

The events of the TCheckBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 2074)	Occurs when the user clicks the control.
	OnContextPopup (see page 2076)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2077)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2080)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2083)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2084)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2085)	Occurs when a control receives the input focus.
	OnExit (see page 2087)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2090)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2093)	Occurs when a key is pressed.
	OnKeyUp (see page 2095)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2096)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2097)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2100)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2100)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2100)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2102)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2105)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2105)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.4.3.1 TCheckBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.4.3.2 TCheckBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.4.3.3 TCheckBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.4.3.4 TCheckBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
```

```
  label control to be dropped on a panel control and stops the
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.4.3.5 TCheckBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.4.3.6 TCheckBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.4.3.7 TCheckBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.4.3.8 TCheckBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

29.1.4.3.9 TCheckBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
```

```
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.4.3.10 TCheckBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.4.3.11 TCheckBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.4.3.12 TCheckBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.4.3.13 TCheckBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.4.3.14 TCheckBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.4.3.15 TCheckBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.4.3.16 TCheckBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.4.3.17 TCheckBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.4.3.18 TCheckBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.4.3.19 TCheckBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

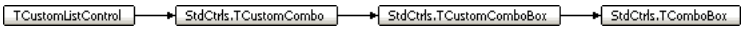
OnDragOver

OnDragDrop

29.1.5 TComboBox Class

TComboBox combines an edit box with a scrollable list.

Class Hierarchy



Pascal

```
TComboBox = class(TCustomComboBox);
```

C++

```
class TComboBox : public TCustomComboBox;
```

File

StdCtrls

Description

A TComboBox component is an edit box with a scrollable drop-down list attached to it. Users can select an item from the list or type directly into the edit box.

Note: The width of the button in a TComboBox is equal to the width that Windows uses for scrollbars. This width depends on the color scheme that the user has chosen (by right-clicking the desktop, selecting Properties, and choosing Appearance). On Windows XP, the user can specifically change the width of the scrollbar by clicking the Advanced button and selecting scrollbar as the Item. If you carefully size your combo box so all the text is visible, then a user running under a different color scheme may find that the text is obscured by the button. One solution is to set the scrollbar wide when developing the application. Most color schemes use a scrollbar that is at most 21 pixels wide.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
```

```
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

TCustomComboBox (see page 2241)

TEdit (see page 2380)

TListBox (see page 2572)

Combo Boxes

List Controls

29.1.5.1 TComboBox Members

The following tables list the members exposed by TComboBox.

Protected Fields

	Name	Description
	FDefEditProc (see page 2206)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 2206)	This is FDefListProc, a member of class TCustomCombo.
	FDropHandle (see page 2207)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 2207)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 2207)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 2207)	This is FEditInstance, a member of class TCustomCombo.
	FFocusChanged (see page 2207)	This is FFocusChanged, a member of class TCustomCombo.
	FIsFocused (see page 2208)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 2208)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 2208)	This is FListInstance, a member of class TCustomCombo.
	FSaveIndex (see page 2208)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 2211)	Adjusts the size and position of the drop down list.
	Change (see page 2211)	Generates an OnChange (see page 2237) event.
	CloseUp (see page 2212)	Generates an OnCloseUp (see page 2239) event.
V	ComboWndProc (see page 2213)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateParams (see page 2245)	Initializes the window-creation parameter record.
V	CreateWnd (see page 2214)	Creates the Windows control that represents the combo box.
V	CreateWnd (see page 2245)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 2214)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 2246)	Destroys the Windows control that represents the combo box.
V	DoSetTextHint (see page 2246)	This is DoSetTextHint, a member of class TCustomComboBox.
V	DrawItem (see page 2246)	Generates an OnDrawItem (see page 2256) event.
	DropDown (see page 2215)	Generates an OnDropDown (see page 2239) event.
V	EditWndProc (see page 2215)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2216)	Returns the value of the DroppedDown (see page 2225) property.
A	GetItemCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
V	GetItemCount (see page 2247)	Returns the number of items in the Items (see page 2227) property list.
A	GetItemHt (see page 2217)	Returns the value of the ItemHeight (see page 2226) property.
V	GetItemHt (see page 2247)	Returns the value of the ItemHeight (see page 2226) property.
V	GetItemIndex (see page 2217)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 2217)	Indicates the class that implements the Items (see page 2227) property.
V	GetItemsClass (see page 2247)	Returns the class used to implement the Items (see page 2227) property.
	GetSelLength (see page 2218)	Returns the value of the SelLength (see page 2230) property.
	GetSelStart (see page 2218)	Returns the value of the SelStart (see page 2233) property.
	GetSelText (see page 2248)	Returns the value of the SelText (see page 2253) property.
V	KeyPress (see page 2248)	Responds to WM_CHAR messages sent to the combo box.
	ListWndProc (see page 2218)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2219)	Initializes the combo box after its form is loaded into memory.
V	MeasureItem (see page 2249)	Generates an OnMeasureItem (see page 2257) event.
	Select (see page 2219)	Generates an OnSelect (see page 2241) event.
	SelectItem (see page 2249)	Selects a specified item from the drop-down list.
V	SetDropDownCount (see page 2220)	Sets the value of the DropDownCount (see page 2225) property.

	SetDroppedDown (see page 2220)	Sets the value of the DroppedDown (see page 2225) property.
V	SetItemHeight (see page 2221)	Sets the value of the ItemHeight (see page 2226) property.
V	SetItemIndex (see page 2221)	Sets the value of the ItemIndex property.
V	SetItems (see page 2221)	Sets the value of the Items (see page 2227) property.
	SetMaxLength (see page 2222)	Sets the value of the MaxLength (see page 2229) property.
	SetSelLength (see page 2222)	Sets the value of the SelLength (see page 2230) property.
	SetSelStart (see page 2222)	Sets the value of the SelStart (see page 2233) property.
V	SetStyle (see page 2249)	Sets the Style (see page 2255) property.
V	WndProc (see page 2223)	Processes messages received by the combo box.
V	WndProc (see page 2250)	Processes messages received by the combo box.

Protected Properties

	Name	Description
R	EditHandle (see page 2225)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2226)	Specifies the number of items in the drop-down list.
R	ListHandle (see page 2229)	Provides access to the window handle of the list of items in the combo box.

Public Methods

	Name	Description
V	AddItem (see page 2209)	Adds an item to the drop-down list of items.
V	Clear (see page 2212)	Deletes all text from the edit box and all items from the Items (see page 2227) list.
V	ClearSelection (see page 2212)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 2213)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2205)	Creates and initializes a TCustomCombo (see page 2203) instance.
V	Create (see page 2244)	Creates and initializes a TCustomComboBox (see page 2241) instance.
V	DeleteSelected (see page 2214)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2205)	Destroys the combo box and frees its memory.
V	Destroy (see page 2244)	Destroys the combo box and frees its memory.
V	Focused (see page 2215)	Indicates whether the combo box has input focus.
V	GetCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
V	SelectAll (see page 2220)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
R	Canvas (see page 2224)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2225)	Indicates whether the drop-down list is currently displayed.
	SelLength (see page 2230)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2233)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 2253)	Represents the selected text in the edit region.

Published Events

	Name	Description
	OnChange (see page 2149)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 2152)	Occurs when the user clicks the control.
	OnCloseUp (see page 2153)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 2154)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2154)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2155)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2158)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2161)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2161)	Occurs when the user opens the drop-down list.

	OnEndDock (see page 2163)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2163)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2164)	Occurs when a control receives the input focus.
	OnExit (see page 2167)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2170)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2173)	Occurs when a key is pressed.
	OnKeyUp (see page 2174)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2176)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 2177)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2177)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 2177)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 2177)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2178)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2112)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2115)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 2116)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2116)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2117)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2117)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 2117)	Specifies which edges of the control are beveled.
	BevelInner (see page 2118)	Specifies the cut of the inner bevel.
	BevelKind (see page 2118)	Specifies the control's bevel style.
	BevelOuter (see page 2119)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2119)	Specifies the bi-directional mode for the control.
	CharCase (see page 2120)	Determines the case of the text in the combo box.
	Color (see page 2120)	Specifies the background color of the control.
	Constraints (see page 2121)	Specifies the size constraints for the control.
	Ctl3D (see page 2121)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2122)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2124)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2124)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2124)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 2126)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 2126)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2127)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2128)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2128)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 2129)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 2130)	Specifies the index of the selected item.
	Items (see page 2131)	Provides access to the list of items (strings) in the list portion of the combo box.
	MaxLength (see page 2133)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 2134)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2134)	Determines where a control looks for its color information.

	ParentCtl3D (see page 2135)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2135)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2135)	Determines where a control looks for its font information.
	ParentShowHint (see page 2136)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2136)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2141)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2143)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2144)	Determines the display style of the combo box.
	TabOrder (see page 2145)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2145)	Determines if the user can tab to a control.
	Text (see page 2146)	Contains a text string associated with the control.
	TextHint (see page 2147)	This is TextHint, a member of class TComboBox.
	Visible (see page 2147)	Determines whether the component appears onscreen.

29.1.5.2 TComboBox Properties

The properties of the TComboBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2112)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2115)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 2116)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2116)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2117)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2117)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 2117)	Specifies which edges of the control are beveled.
	BevelInner (see page 2118)	Specifies the cut of the inner bevel.
	BevelKind (see page 2118)	Specifies the control's bevel style.
	BevelOuter (see page 2119)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2119)	Specifies the bi-directional mode for the control.
	CharCase (see page 2120)	Determines the case of the text in the combo box.
	Color (see page 2120)	Specifies the background color of the control.
	Constraints (see page 2121)	Specifies the size constraints for the control.
	Ctl3D (see page 2121)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2122)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2124)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2124)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2124)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 2126)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 2126)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2127)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2128)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2128)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 2129)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 2130)	Specifies the index of the selected item.
	Items (see page 2131)	Provides access to the list of items (strings) in the list portion of the combo box.

	MaxLength (see page 2133)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 2134)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2134)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2135)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2135)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2135)	Determines where a control looks for its font information.
	ParentShowHint (see page 2136)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2136)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2141)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2143)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2144)	Determines the display style of the combo box.
	TabOrder (see page 2145)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2145)	Determines if the user can tab to a control.
	Text (see page 2146)	Contains a text string associated with the control.
	TextHint (see page 2147)	This is TextHint, a member of class TComboBox.
	Visible (see page 2147)	Determines whether the component appears onscreen.

29.1.5.2.1 TComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
```

```
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.5.2.2 TComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.5.2.3 TComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 2253)

29.1.5.2.4 TComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, sometimes saving you keystrokes.

As you type, the ItemIndex (see page 2130) property is updated with the list entry that matches the characters you have entered. As you type, different list enrties may match, and ItemIndex (see page 2130) is changed accordingly.

There is a delay of AutoCompleteDelay (see page 2117) milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 2117)

29.1.5.2.5 TComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2116) is true.

See Also

AutoComplete (see page 2116)

29.1.5.2.6 TComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 2251)

OnDropDown (see page 2239)

DroppedDown (see page 2225)

29.1.5.2.7 TComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

29.1.5.2.8 TComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

29.1.5.2.9 TComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

29.1.5.2.10 TComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

29.1.5.2.11 TComboBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.5.2.12 TComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text

29.1.5.2.13 TComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
```

```
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.5.2.14 TComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.5.2.15 TComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.5.2.16 TComboBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
```

```
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.5.2.17 TComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.5.2.18 TComboBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.5.2.19 TComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.5.2.20 TComboBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items (see page 2227)

ItemCount (see page 2226)

29.1.5.2.21 TComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.5.2.22 TComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.5.2.23 TComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

29.1.5.2.24 TComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

29.1.5.2.25 TComboBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 2227)

OnMeasureItem (see page 2257)

Style (see page 2255)

29.1.5.2.26 TComboBox.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex;
```

C++

```
__property ItemIndex;
```

Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1. If the list control supports multiple selected items, ItemIndex is the index of the selected item that has focus.

Set ItemIndex programmatically to select an item by passing in the index value.

Note: To implement the ItemIndex property in a TCustomListControl descendant, override the protected GetItemIndex and SetItemIndex methods.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
```

```
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
             mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

GetItemIndex

SetItemIndex

29.1.5.2.27 TComboBox.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", reinterpret_cast<TObject *>(vsIcon));
  ComboBox1->Items->AddObject("vsList", reinterpret_cast<TObject *>(vsList));
  ComboBox1->Items->AddObject("vsReport", reinterpret_cast<TObject *>(vsReport));
  ComboBox1->Items->AddObject("vsSmallIcon", reinterpret_cast<TObject *>(vsSmallIcon));
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
```

```
    with ListView1 do
    begin
      SmallImages := ImageList1;
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := 'Image' + IntToStr(I);
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

ItemIndex

TStrings

Assign

29.1.5.2.28 TComboBox.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that

may be entered into an edit control.

See Also

Text

29.1.5.2.29 TComboBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.5.2.30 TComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.5.2.31 TComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.5.2.32 TComboBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.5.2.33 TComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.5.2.34 TComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.5.2.35 TComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.5.2.36 TComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.5.2.37 TComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items list alphabetically. New items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
```

```
  open forms when the user drops down the list. The list is
  regenerated every time the list is dropped down, so if the
  application opens or closes forms, the combo box stays current.
  }
  procedure TForm1.ComboBox1DropDown(Sender: TObject);
  var
    I: Integer;
  begin
    with ComboBox1 do
    begin
      Items.BeginUpdate; { prevent repaints until done }
      Items.Clear; { empty the list of any old values }
      for I := 0 to Screen.CustomFormCount - 1 do
      begin
        if (Screen.CustomForms[I].Visible) then
          Items.Add(Screen.CustomForms[I].Caption); { add form name }
      end;
      Items.EndUpdate; {reenable painting }
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ComboBox1.Sorted := True; { make sure the form names are sorted }
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form3.Show;
  end;
```

See Also

Items

Using TDBListBox and TDBComboBox

29.1.5.2.38 TComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem event occurs as well.

See Also

Height

ItemHeight

OnDrawItem (see page 2256)

OnMeasureItem (see page 2257)

29.1.5.2.39 TComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.5.2.40 TComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.5.2.41 TComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
```

```
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2368)

29.1.5.2.42 TComboBox.TextHint Property

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

This is TextHint, a member of class TComboBox.

29.1.5.2.43 TComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.5.3 TComboBox Events

The events of the TComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 2149)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 2152)	Occurs when the user clicks the control.
	OnCloseUp (see page 2153)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 2154)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2154)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2155)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2158)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2161)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2161)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 2163)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2163)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2164)	Occurs when a control receives the input focus.
	OnExit (see page 2167)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2170)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2173)	Occurs when a key is pressed.
	OnKeyUp (see page 2174)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2176)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 2177)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2177)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 2177)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 2177)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2178)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.5.3.1 TComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
```

```
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
    end;
  ComboBox1.ItemIndex := 0;
  PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text

OnCloseUp (see page 2239)

29.1.5.3.2 TComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.5.3.3 TComboBox.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 2239)

DroppedDown (see page 2225)

29.1.5.3.4 TComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.5.3.5 TComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.5.3.6 TComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.5.3.7 TComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}
```

```
void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
```

```
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.5.3.8 TComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas

Items

OnMeasureItem (see page 2257)

Style (see page 2255)

29.1.5.3.9 TComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
```

```
      begin
        if (Screen.CustomForms[I].Visible) then
          Items.Add(Screen.CustomForms[I].Caption); { add form name }
      end;
      Items.EndUpdate; {reenable painting }
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ComboBox1.Sorted := True; { make sure the form names are sorted }
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form3.Show;
  end;
```

See Also

DroppedDown (see page 2225)

OnCloseUp (see page 2239)

29.1.5.3.10 TComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.5.3.11 TComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.5.3.12 TComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
```

```
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.5.3.13 TComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

29.1.5.3.14 TComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.5.3.15 TComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.5.3.16 TComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown

OnKeyPress

29.1.5.3.17 TComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem. Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight

Items

OnDrawItem (see page 2256)

Style (see page 2255)

29.1.5.3.18 TComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.5.3.19 TComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.5.3.20 TComboBox.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 2251)

ItemIndex

Text

29.1.5.3.21 TComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.5.3.22 TComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.6 TCustomButton Class

Base class for a push button control.

Class Hierarchy

Pascal

```
TCustomButton = class(TButtonControl);
```

C++

```
class TCustomButton : public TButtonControl;
```

File

StdCtrls

Description

Use TCustomButton as a base class for a standard push button on a form.

See Also

TButton (see page 1961)

29.1.6.1 TCustomButton Members

The following tables list the members exposed by TCustomButton.

Enumerations

	Name	Description
	TButtonStyle (see page 2181)	Style (see page 2194) of TButton (see page 1961).

Protected Methods

	Name	Description
	ActionChange (see page 2182)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2183)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2184)	Creates the underlying window.
	GetActionLinkClass (see page 2184)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2036)	Returns whether the button control is checked.
	Loaded (see page 2184)	Initializes the control after it is loaded from a stream.
	Notification (see page 2185)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2185)	This is SetButtonStyle, a member of class TCustomButton.

	SetChecked (see page 2036)	Sets the Checked (see page 2037) property.
	UpdateImageList (see page 2186)	This is UpdateImageList, a member of class TCustomButton.
	UpdateImages (see page 2186)	This is UpdateImages, a member of class TCustomButton.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2037)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.
	WordWrap (see page 2039)	Specifies whether the button text wraps to fit the width of the control.

Public Events

	Name	Description
	OnDropDownClick (see page 2195)	Down arrow part of Vista style split button is clicked.

Public Methods

	Name	Description
	Click (see page 2183)	Simulates a mouse click, as if the user had clicked the button.
	Create (see page 2180)	Creates and initializes a button instance.
	Destroy (see page 2181)	Destroys an instance of TWinControl.
	UseRightToLeftAlignment (see page 2186)	Specifies whether the control's alignment is in a right-to-left mode.

Public Properties

	Name	Description
	Cancel (see page 2187)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	CommandLinkHint (see page 2187)	Text displayed as hint below button caption for Command Link.
	Default (see page 2188)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 2189)	Image index for disabled button state.
	DropDownMenu (see page 2189)	Split button drop down menu.
	ElevationRequired (see page 2190)	Put shield icon on button indicating elevated access rights required.
	HotImageIndex (see page 2190)	Image index for hot button state.
	ImageAlignment (see page 2191)	Alignment of image on button.
	ImageIndex (see page 2191)	Image index for normal button state.
	ImageMargins (see page 2192)	Margins of image on button.
	Images (see page 2192)	List of images for a button.
	ModalResult (see page 2193)	Determines whether and how the button closes its (modal) parent form.
	PressedImageIndex (see page 2193)	Image index for pressed button state.
	SelectedImageIndex (see page 2194)	Image index for selected button state.
	Style (see page 2194)	Style of a button.
	TabStop (see page 2195)	Determines if the user can tab to a control.

29.1.6.2 TCustomButton.Create Constructor

Creates and initializes a button instance.

Pascal

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

C++

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

Description

Use Create to programmatically instantiate a button. Buttons added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the button instance. It becomes the value of the Owner property.

See Also

Owner

29.1.6.3 TCustomButton.Destroy Destructor

Destroys an instance of TWinControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomButton();
```

Description

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

See Also

System::TObject::Free (see page 3382)

Destroy

29.1.6.4 TCustomButton Enumerations

The enumerations of the TCustomButton class are listed here.

Enumerations

	Name	Description
	TButtonStyle (see page 2181)	Style (see page 2194) of TButton (see page 1961).

29.1.6.4.1 StdCtrls.TCustomButton.TButtonStyle Enumeration

Style (see page 2194) of TButton (see page 1961).

Pascal

```
TButtonStyle = (
  bsPushButton,
  bsCommandLink,
  bsSplitButton
);
```

C++

```
enum TButtonStyle {
  bsPushButton,
  bsCommandLink,
  bsSplitButton
};
```

File

StdCtrls

Description

TButtonStyle indicates a TButton (see page 1961)'s style, which is the value of its Style (see page 2194) property. Following are the possible values of TButtonStyle:

Value	Meaning
bsPushButton	The button is a standard push button.
bsCommandLink	Command Link style button. Display the text in CommandLinkHint (see page 2187) as a hint below the button caption. Windows Vista only.
bsSplitButton	Make the button a Vista style split button, which has a drop down menu. Windows Vista only.

See Also

TButton (see page 1961)

CommandLinkHint (see page 1973)

Style (see page 1995)

29.1.6.5 TCustomButton Methods

The methods of the TCustomButton class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2182)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2183)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2184)	Creates the underlying window.
	GetActionLinkClass (see page 2184)	Returns the appropriate action link class that is associated with button controls.
	Loaded (see page 2184)	Initializes the control after it is loaded from a stream.
	Notification (see page 2185)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2185)	This is SetButtonStyle, a member of class TCustomButton.
	UpdateImageList (see page 2186)	This is UpdateImageList, a member of class TCustomButton.
	UpdateImages (see page 2186)	This is UpdateImages, a member of class TCustomButton.

Public Methods

	Name	Description
	Click (see page 2183)	Simulates a mouse click, as if the user had clicked the button.
	UseRightToLeftAlignment (see page 2186)	Specifies whether the control's alignment is in a right-to-left mode.

29.1.6.5.1 TCustomButton.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction

29.1.6.5.2 TCustomButton.Click Method

Simulates a mouse click, as if the user had clicked the button.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Calling Click assigns the value of the button's ModalResult (see page 2193) property to the ModalResult (see page 2193) property of its parent form, and then generates an OnClick event.

See Also

ModalResult (see page 2193)

OnClick

29.1.6.5.3 TCustomButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

29.1.6.5.4 TCustomButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

29.1.6.5.5 TCustomButton.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink class type.

See Also

TButtonActionLink (see page 2030)

29.1.6.5.6 TCustomButton.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

29.1.6.5.7 TCustomButton.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

29.1.6.5.8 TCustomButton.SetButtonStyle Method

Pascal

```
procedure SetButtonStyle(ADefault: Boolean); virtual;
```

C++

```
virtual __fastcall SetButtonStyle(Boolean ADefault);
```

Description

This is SetButtonStyle, a member of class TCustomButton.

29.1.6.5.9 TCustomButton.UpdateImageList Method

Pascal

```
procedure UpdateImageList; virtual;
```

C++

```
virtual __fastcall UpdateImageList();
```

Description

This is UpdateImageList, a member of class TCustomButton.

29.1.6.5.10 TCustomButton.UpdateImages Method

Pascal

```
procedure UpdateImages; virtual;
```

C++

```
virtual __fastcall UpdateImages();
```

Description

This is UpdateImages, a member of class TCustomButton.

29.1.6.5.11 TCustomButton.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

For this class, UseRightToLeftAlignment always returns false. This is because the alignment of a button should never be reversed automatically when running in Middle Eastern locales.

See Also

ModalResult (see page 2193)

OnClick

29.1.6.6 TCustomButton Properties

The properties of the TCustomButton class are listed here.

Public Properties

	Name	Description
	Cancel (see page 2187)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	CommandLinkHint (see page 2187)	Text displayed as hint below button caption for Command Link.
	Default (see page 2188)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.

	DisabledImageIndex (see page 2189)	Image index for disabled button state.
	DropDownMenu (see page 2189)	Split button drop down menu.
	ElevationRequired (see page 2190)	Put shield icon on button indicating elevated access rights required.
	HotImageIndex (see page 2190)	Image index for hot button state.
	ImageAlignment (see page 2191)	Alignment of image on button.
	ImageIndex (see page 2191)	Image index for normal button state.
	ImageMargins (see page 2192)	Margins of image on button.
	Images (see page 2192)	List of images for a button.
	ModalResult (see page 2193)	Determines whether and how the button closes its (modal) parent form.
	PressedImageIndex (see page 2193)	Image index for pressed button state.
	SelectedImageIndex (see page 2194)	Image index for selected button state.
	Style (see page 2194)	Style of a button.
	TabStop (see page 2195)	Determines if the user can tab to a control.

29.1.6.6.1 TCustomButton.Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

See Also

Default (see page 2188)

OnClick

29.1.6.6.2 TCustomButton.CommandLinkHint Property

Text displayed as hint below button caption for Command Link.

Pascal

```
property CommandLinkHint: string;
```

C++

```
__property UnicodeString CommandLinkHint;
```

Description

CommandLinkHint is the text displayed as a hint below the button caption when the Style (see page 2194) property is **bsCommandLink** to specify a Command Link button.

This is available on Windows Vista only.

See Also

Style (see page 2194)

TButtonStyle

29.1.6.6.3 TCustomButton.Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
```

```
end;
```

See Also

Cancel (see page 2187)

OnClick

29.1.6.6.4 TCustomButton.DisabledImageIndex Property

Image index for disabled button state.

Pascal

```
property DisabledImageIndex: TImageIndex;
```

C++

```
__property TImageIndex DisabledImageIndex;
```

Description

DisabledImageIndex is an image index into the Images (see page 2192) property for the disabled button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.6.6.5 TCustomButton.DropDownMenu Property

Split button drop down menu.

Pascal

```
property DropDownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropDownMenu;
```

Description

When the Style (see page 2194) property is **bsSplitButton**, the button is a Vista style split button with a drop down menu. DropDownMenu specifies the drop down menu for the down arrow button part. The OnDropDownClick (see page 2195) event also occurs when the down arrow button part is clicked.

This is available on Windows Vista only.

See Also

TPopupMenu

Style (see page 2194)

OnDropDownClick (see page 2195)

TButtonStyle

29.1.6.6.6 TCustomButton.ElevationRequired Property

Put shield icon on button indicating elevated access rights required.

Pascal

```
property ElevationRequired: Boolean;
```

C++

```
__property Boolean ElevationRequired;
```

Description

When ElevationRequired is true, the button has a shield icon, indicating that elevated access rights are required. Vista UI guidelines specify that when there is a button that performs an operation requiring elevated access rights, it should have this shield icon.

This icon overrides any of the properties specifying a button image for various button states, as in ImageIndex (see page 2191).

This is available on Windows Vista only.

See Also

ImageIndex (see page 2191)

Images (see page 2192)

29.1.6.6.7 TCustomButton.HotImageIndex Property

Image index for hot button state.

Pascal

```
property HotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex HotImageIndex;
```

Description

HotImageIndex is an image index into the Images (see page 2192) property for the hot button state, i.e., the cursor hovering over the button. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.6.6.8 TCustomButton.ImageAlignment Property

Alignment of image on button.

Pascal

```
property ImageAlignment: TImageAlignment;
```

C++

```
__property TImageAlignment ImageAlignment;
```

Description

TImageAlignment (see page 2909) indicates the alignment of an image on a button. It is a TImageAlignment (see page 2909) type.

See Also

ImageMargins (see page 2192)

Images (see page 2192)

TImageAlignment (see page 2909)

29.1.6.6.9 TCustomButton.ImageIndex Property

Image index for normal button state.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an image index into the Images (see page 2192) property for the normal button state. If the index value is -1, no image is used.

In Vista, a standard green arrow is used when the index is -1 and the Style (see page 2194) property is **bsCommandLink** for a Command Link button.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

Images (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

Style (see page 2194)

TButtonStyle

29.1.6.6.10 TCustomButton.ImageMargins Property

Margins of image on button.

Pascal

```
property ImageMargins: TImageMargins;
```

C++

```
__property TImageMargins ImageMargins;
```

Description

A button image may be positioned using margins. ImageMargins holds aTImageMargins object containing the margins of an image on a button. The TImageMargins.OnChange event is triggered when a margin changes.

See Also

ImageAlignment (see page 2191)

TImageMargins (see page 2517)

Images (see page 2192)

OnChange (see page 2521)

29.1.6.6.11 TCustomButton.Images Property

List of images for a button.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a TCustomImageList object listing the various images that can be placed on a button.

The button's image depends on the button state. These images are determined by the value of the image state properties that index ImageIndex (see page 2191): DisabledImageIndex (see page 2189), HotImageIndex (see page 2190), ImageIndex (see page 2191), PressedImageIndex (see page 2193), and SelectedImageIndex (see page 2194). When the ImageIndex (see page 2191) index is -1, it generally means that no image is used for the normal state. For the other state image indexes, -1 means to use the same image as the normal state.

The ImageMargins (see page 2192) and ImageAlignment (see page 2191) properties orient the image inside the button. The image is not sized to fit the button, and the button is not stretched.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageAlignment (see page 2191)

ImageIndex (see page 2191)

ImageMargins (see page 2192)

PressedImageIndex (see page 2193)

SelectedImageIndex (see page 2194)

29.1.6.6.12 TCustomButton.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the button component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel (see page 2187) buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel (see page 2187) button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick event handlers are required for the buttons.

The following constants are of TModalResult type and defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.
mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.
mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.
mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.
mrClose	mrYesToAll + 1	The user exited with the CLOSE button.

See Also

TModalResult

29.1.6.6.13 TCustomButton.PressedImageIndex Property

Image index for pressed button state.

Pascal

```
property PressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex PressedImageIndex;
```

Description

PressedImageIndex is an image index into the Images (see page 2192) property for the pressed button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

SelectedImageIndex (see page 2194)

29.1.6.6.14 TCustomButton.SelectedImageIndex Property

Image index for selected button state.

Pascal

```
property SelectedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex SelectedImageIndex;
```

Description

SelectedImageIndex is an image index into the Images (see page 2192) property for the selected button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2189)

HotImageIndex (see page 2190)

ImageIndex (see page 2191)

Images (see page 2192)

PressedImageIndex (see page 2193)

29.1.6.6.15 TCustomButton.Style Property

Style of a button.

Pascal

```
property Style: TButtonStyle;
```

C++

```
__property TButtonStyle Style;
```

Description

Style indicates a button's style, which is a TButtonStyle (see page 2181) type.

See Also

TButtonStyle

29.1.6.6.16 TCustomButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.6.7 TCustomButton Events

The events of the TCustomButton class are listed here.

Public Events

	Name	Description
	OnDropDownClick (see page 2195)	Down arrow part of Vista style split button is clicked.

29.1.6.7.1 TCustomButton.OnDropDownClick Event

Down arrow part of Vista style split button is clicked.

Pascal

```
property OnDropDownClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDownClick;
```

Description

OnDropDownClick occurs when the down arrow part of a Vista style split button is clicked. This only applies when the button's Style (see page 2194) property is **bsSplitButton**.

This is available on Windows Vista only.

See Also

DropDownMenu (see page 2189)

Style (see page 2194)

TButtonStyle

29.1.7 TCustomCheckBox Class

TCustomCheckBox is the ancestor of all check-box components.

Class Hierarchy

Pascal

```
TCustomCheckBox = class(TButtonControl);
```

C++

```
class TCustomCheckBox : public TButtonControl;
```

File

StdCtrls

Description

TCustomCheckBox is an abstract class from which check-box components, including TCheckBox (see page 2040) and TDBCheckBox, descend. Check boxes present the user with options that can be selected (checked) or deselected (unchecked).

See Also

TCheckBox (see page 2040)

TDBCheckBox

29.1.7.1 TCustomCheckBox Members

The following tables list the members exposed by TCustomCheckBox.

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	Click (see page 2197)	Click generates an OnClick event.
	CreateParams (see page 2198)	Initializes a window-creation parameter record.
	CreateWnd (see page 2198)	Creates a Windows control corresponding to the check box.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2199)	Returns the value of the Checked (see page 2037) property.
	SetChecked (see page 2199)	Sets the value of the Checked (see page 2037) property.
	Toggle (see page 2200)	Changes the state of the check box.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	Alignment (see page 2200)	Controls the position of the check box's caption.

	AllowGrayed (see page 2201)	Determines whether check box can be in a "grayed" state.
	Checked (see page 2037)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.
	State (see page 2201)	Indicates whether the check box is selected, deselected, or grayed.
	WordWrap (see page 2039)	Specifies whether the button text wraps to fit the width of the control.

Public Methods

	Name	Description
	Create (see page 2197)	Creates and initializes a TCustomCheckBox (see page 2196) instance.
	GetControlsAlignment (see page 2199)	Returns the alignment of the check box caption.

Published Properties

	Name	Description
	TabStop (see page 2202)	Determines if the user can tab to a control.

29.1.7.2 TCustomCheckBox.Create Constructor

Creates and initializes a TCustomCheckBox (see page 2196) instance.

Pascal

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

C++

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

Description

Call Create to generate a check box at runtime. Check boxes placed on forms at design time, Create is called automatically.

AOwner is a component, typically the form, that is responsible for freeing the check box instance.

See Also

Owner

29.1.7.3 TCustomCheckBox Methods

The methods of the TCustomCheckBox class are listed here.

Protected Methods

	Name	Description
	Click (see page 2197)	Click generates an OnClick event.
	CreateParams (see page 2198)	Initializes a window-creation parameter record.
	CreateWnd (see page 2198)	Creates a Windows control corresponding to the check box.
	GetChecked (see page 2199)	Returns the value of the Checked (see page 2037) property.
	SetChecked (see page 2199)	Sets the value of the Checked (see page 2037) property.
	Toggle (see page 2200)	Changes the state of the check box.

Public Methods

	Name	Description
	GetControlsAlignment (see page 2199)	Returns the alignment of the check box caption.

29.1.7.3.1 TCustomCheckBox.Click Method

Click generates an OnClick event.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

The protected Click method is called automatically when the check box is clicked with the mouse. In addition to generating an OnClick event, it sends a message to the parent control's window to indicate that the state of the check box has changed.

See Also

OnClick

Parent

29.1.7.3.2 TCustomCheckBox.CreateParams Method

Initializes a window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2198) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams fills in the parameter record that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

In TCustomCheckBox (see page 2196), CreateParams calls the inherited method, then makes specific adjustments for check boxes.

See Also

CreateWnd

CreateWindowHandle

29.1.7.3.3 TCustomCheckBox.CreateWnd Method

Creates a Windows control corresponding to the check box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the check box is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. In TCustomCheckBox (see page 2196), this method calls the inherited method to generate the underlying windows control, then sends a BM_SETCHECK message to set the check state of the control.

See Also

CreateParams (see page 2198)

29.1.7.3.4 TCustomCheckBox.GetChecked Method

Returns the value of the Checked (see page 2037) property.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the protected implementation of the Checked (see page 2037) property. Override this method and the SetChecked (see page 2199) method to change the Checked (see page 2037) property in descendant classes.

See Also

Checked

SetChecked (see page 2199)

29.1.7.3.5 TCustomCheckBox.GetControlsAlignment Method

Returns the alignment of the check box caption.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

GetControlsAlignment indicates the alignment of the check box caption. Usually, the caption is right-justified, but if the UseRightToLeftAlignment method returns true and the check mark is on the left, then GetControlsAlignment indicates that the caption should be left-justified.

See Also

Caption

alignment (see page 2200)

UseRightToLeftAlignment

29.1.7.3.6 TCustomCheckBox.SetChecked Method

Sets the value of the Checked (see page 2037) property.

Pascal

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

C++

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

Description

SetChecked is the protected implementation of the Checked (see page 2037) property. Override this method and the GetChecked (see page 2199) method to change the Checked (see page 2037) property in descendant classes.

See Also

Checked

GetChecked (see page 2199)

29.1.7.3.7 TCustomCheckBox.Toggle Method

Changes the state of the check box.

Pascal

```
procedure Toggle; virtual;
```

C++

```
virtual __fastcall Toggle();
```

Description

Use the Toggle method to select or deselect the check box programmatically. Toggle switches among the valid check-box states—from checked to unchecked to grayed, or from checked to unchecked.

See Also

AllowGrayed (see page 2201)

State (see page 2201)

29.1.7.4 TCustomCheckBox Properties

The properties of the TCustomCheckBox class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2200)	Controls the position of the check box's caption.
	AllowGrayed (see page 2201)	Determines whether check box can be in a "grayed" state.
	State (see page 2201)	Indicates whether the check box is selected, deselected, or grayed.

Published Properties

	Name	Description
	TabStop (see page 2202)	Determines if the user can tab to a control.

29.1.7.4.1 TCustomCheckBox.Alignment Property

Controls the position of the check box's caption.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

If Alignment is set to taRightJustify, the caption appears to the right of the check box. If alignment is set to taLeftJustify, the caption appears to the left of the check box.

29.1.7.4.2 TCustomCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked

State (see page 2201)

29.1.7.4.3 TCustomCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 2201)

Checked

29.1.7.4.4 TCustomCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

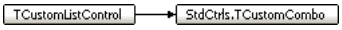
See Also

TabOrder

29.1.8 TCustomCombo Class

TCustomCombo is the base class for all combo boxes.

Class Hierarchy



Pascal

```
TCustomCombo = class(TCustomListControl);
```

C++

```
class TCustomCombo : public TCustomListControl;
```

File

StdCtrls

Description

Use TCustomCombo as a base class when defining a control that represents a specialized combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Typically, combo box classes do not descend directly from TCustomCombo. Rather, they descend from one of TCustomCombo's descendants, TCustomComboBox (see page 2241) or TCustomComboBoxEx. Use TCustomCombo as a base class for a simple combo box, or TCustomComboBoxEx as a base class for a combo box that supports the extended combo box features such as images on the list entries.

Do not create instances of TCustomCombo. To put a combo box in a form, use a descendant of TCustomCombo, such as TComboBox (see page 2106) or TComboBoxEx.

See Also

TComboBox (see page 2106)

TComboBoxEx

TDBComboBox

TColorBox

TDriveComboBox

TFilterComboBox

TCustomComboBox (see page 2241)

TCustomComboBoxEx

29.1.8.1 TCustomCombo Members

The following tables list the members exposed by TCustomCombo.

Protected Events

	Name	Description
	OnChange (see page 2237)	Occurs when the user changes the text displayed in the edit region.

	OnCloseUp (see page 2239)	Occurs when the drop-down list closes up due to some user action.
	OnDropDown (see page 2239)	Occurs when the user opens the drop-down list.
	OnSelect (see page 2241)	Occurs when the user selects a string in the drop-down list.

Protected Fields

	Name	Description
	FDefEditProc (see page 2206)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 2206)	This is FDefListProc, a member of class TCustomCombo.
	FDropHandle (see page 2207)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 2207)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 2207)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 2207)	This is FEditInstance, a member of class TCustomCombo.
	FFocusChanged (see page 2207)	This is FFocusChanged, a member of class TCustomCombo.
	FIsFocused (see page 2208)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 2208)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 2208)	This is FListInstance, a member of class TCustomCombo.
	FSaveIndex (see page 2208)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 2211)	Adjusts the size and position of the drop down list.
	Change (see page 2211)	Generates an OnChange (see page 2237) event.
	CloseUp (see page 2212)	Generates an OnCloseUp (see page 2239) event.
V	ComboWndProc (see page 2213)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateWnd (see page 2214)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 2214)	Destroys all of the windows controls that implement the combo box.
	DropDown (see page 2215)	Generates an OnDropDown (see page 2239) event.
V	EditWndProc (see page 2215)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2216)	Returns the value of the DroppedDown (see page 2225) property.
A	GetItemCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
A	GetItemHt (see page 2217)	Returns the value of the ItemHeight (see page 2226) property.
V	GetItemIndex (see page 2217)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 2217)	Indicates the class that implements the Items (see page 2227) property.
	GetSelLength (see page 2218)	Returns the value of the SelLength (see page 2230) property.
	GetSelStart (see page 2218)	Returns the value of the SelStart (see page 2233) property.
	ListWndProc (see page 2218)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2219)	Initializes the combo box after its form is loaded into memory.
	Select (see page 2219)	Generates an OnSelect (see page 2241) event.
V	SetDropDownCount (see page 2220)	Sets the value of the DropDownCount (see page 2225) property.
	SetDroppedDown (see page 2220)	Sets the value of the DroppedDown (see page 2225) property.
V	SetItemHeight (see page 2221)	Sets the value of the ItemHeight (see page 2226) property.
V	SetItemIndex (see page 2221)	Sets the value of the ItemIndex property.
V	SetItems (see page 2221)	Sets the value of the Items (see page 2227) property.
	SetMaxLength (see page 2222)	Sets the value of the MaxLength (see page 2229) property.
	SetSelLength (see page 2222)	Sets the value of the SelLength (see page 2230) property.
	SetSelStart (see page 2222)	Sets the value of the SelStart (see page 2233) property.
V	WndProc (see page 2223)	Processes messages received by the combo box.

Protected Properties

	Name	Description
	DropDownCount (see page 2225)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 2225)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2226)	Specifies the number of items in the drop-down list.

	ItemHeight (see page 2226)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 2229)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2229)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 2230)	Determines where a control looks for its color information.

Public Methods

	Name	Description
V	AddItem (see page 2209)	Adds an item to the drop-down list of items.
V	Clear (see page 2212)	Deletes all text from the edit box and all items from the Items (see page 2227) list.
V	ClearSelection (see page 2212)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 2213)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2205)	Creates and initializes a TCustomCombo (see page 2203) instance.
V	DeleteSelected (see page 2214)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2205)	Destroys the combo box and frees its memory.
V	Focused (see page 2215)	Indicates whether the combo box has input focus.
V	GetCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
V	SelectAll (see page 2220)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
R	Canvas (see page 2224)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2225)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2227)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2230)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2233)	Specifies the position of the first selected character in the edit portion of the combo box.
	TabStop (see page 2236)	Determines if the user can tab to a control.

29.1.8.2 TCustomCombo.Create Constructor

Creates and initializes a TCustomCombo (see page 2203) instance.

Pascal

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

C++

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

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component (typically a form) that is responsible for freeing the combo box instance.

See Also

Owner

29.1.8.3 TCustomCombo.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomCombo();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

29.1.8.4 TCustomCombo Fields

The fields of the TCustomCombo class are listed here.

Protected Fields

	Name	Description
	FDefEditProc (see page 2206)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 2206)	This is FDefListProc, a member of class TCustomCombo.
	FDropHandle (see page 2207)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 2207)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 2207)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 2207)	This is FEditInstance, a member of class TCustomCombo.
	FFocusChanged (see page 2207)	This is FFocusChanged, a member of class TCustomCombo.
	FIsFocused (see page 2208)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 2208)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 2208)	This is FListInstance, a member of class TCustomCombo.
	FSaveIndex (see page 2208)	This is FSaveIndex, a member of class TCustomCombo.

29.1.8.4.1 TCustomCombo.FDefEditProc Field

Pascal

```
FDefEditProc: TWindowProcPtr;
```

C++

```
TWindowProcPtr FDefEditProc;
```

Description

This is FDefEditProc, a member of class TCustomCombo.

29.1.8.4.2 TCustomCombo.FDefListProc Field

Pascal

```
FDefListProc: TWindowProcPtr;
```

C++

```
TWindowProcPtr FDefListProc;
```

Description

This is FDefListProc, a member of class TCustomCombo.

29.1.8.4.3 TCustomCombo.FDropHandle Field

Pascal

```
FDropHandle: HWnd;
```

C++

```
HWnd FDropHandle;
```

Description

This is FDropHandle, a member of class TCustomCombo.

29.1.8.4.4 TCustomCombo.FDroppingDown Field

Pascal

```
FDroppingDown: Boolean;
```

C++

```
Boolean FDroppingDown;
```

Description

This is FDroppingDown, a member of class TCustomCombo.

29.1.8.4.5 TCustomCombo.FEditHandle Field

Pascal

```
FEditHandle: HWnd;
```

C++

```
HWnd FEditHandle;
```

Description

This is FEditHandle, a member of class TCustomCombo.

29.1.8.4.6 TCustomCombo.FEditInstance Field

Pascal

```
FEditInstance: Pointer;
```

C++

```
void * FEditInstance;
```

Description

This is FEditInstance, a member of class TCustomCombo.

29.1.8.4.7 TCustomCombo.FFocusChanged Field

Pascal

```
FFocusChanged: Boolean;
```

C++

```
Boolean FFocusChanged;
```

Description

This is FFocusChanged, a member of class TCustomCombo.

29.1.8.4.8 TCustomCombo.FIsFocused Field

Pascal

```
FIsFocused: Boolean;
```

C++

```
Boolean FIsFocused;
```

Description

This is FIsFocused, a member of class TCustomCombo.

29.1.8.4.9 TCustomCombo.FListHandle Field

Pascal

```
FListHandle: HWnd;
```

C++

```
HWnd FListHandle;
```

Description

This is FListHandle, a member of class TCustomCombo.

29.1.8.4.10 TCustomCombo.FListInstance Field

Pascal

```
FListInstance: Pointer;
```

C++

```
void * FListInstance;
```

Description

This is FListInstance, a member of class TCustomCombo.

29.1.8.4.11 TCustomCombo.FSaveIndex Field

Pascal

```
FSaveIndex: Integer;
```

C++

```
int FSaveIndex;
```

Description

This is FSaveIndex, a member of class TCustomCombo.

29.1.8.5 TCustomCombo Methods

The methods of the TCustomCombo class are listed here.

Protected Methods

	Name	Description
	AdjustDropDown (see page 2211)	Adjusts the size and position of the drop down list.
	Change (see page 2211)	Generates an OnChange (see page 2237) event.
	CloseUp (see page 2212)	Generates an OnCloseUp (see page 2239) event.
	ComboWndProc (see page 2213)	Responds to Windows messages sent to the edit control or drop-down list.
	CreateWnd (see page 2214)	Creates the Windows control that represents the combo box.
	DestroyWindowHandle (see page 2214)	Destroys all of the windows controls that implement the combo box.
	DropDown (see page 2215)	Generates an OnDropDown (see page 2239) event.
	EditWndProc (see page 2215)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2216)	Returns the value of the DroppedDown (see page 2225) property.
	GetItemCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
	GetItemHt (see page 2217)	Returns the value of the ItemHeight (see page 2226) property.
	GetItemIndex (see page 2217)	Returns the value of the ItemIndex property.
	GetItemsClass (see page 2217)	Indicates the class that implements the Items (see page 2227) property.
	GetSelLength (see page 2218)	Returns the value of the SelLength (see page 2230) property.
	GetSelStart (see page 2218)	Returns the value of the SelStart (see page 2233) property.
	ListWndProc (see page 2218)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
	Loaded (see page 2219)	Initializes the combo box after its form is loaded into memory.
	Select (see page 2219)	Generates an OnSelect (see page 2241) event.
	SetDropDownCount (see page 2220)	Sets the value of the DropDownCount (see page 2225) property.
	SetDroppedDown (see page 2220)	Sets the value of the DroppedDown (see page 2225) property.
	SetItemHeight (see page 2221)	Sets the value of the ItemHeight (see page 2226) property.
	SetItemIndex (see page 2221)	Sets the value of the ItemIndex property.
	SetItems (see page 2221)	Sets the value of the Items (see page 2227) property.
	SetMaxLength (see page 2222)	Sets the value of the MaxLength (see page 2229) property.
	SetSelLength (see page 2222)	Sets the value of the SelLength (see page 2230) property.
	SetSelStart (see page 2222)	Sets the value of the SelStart (see page 2233) property.
	WndProc (see page 2223)	Processes messages received by the combo box.

Public Methods

	Name	Description
	AddItem (see page 2209)	Adds an item to the drop-down list of items.
	Clear (see page 2212)	Deletes all text from the edit box and all items from the Items (see page 2227) list.
	ClearSelection (see page 2212)	Deselects any selected items in the drop-down list.
	CopySelection (see page 2213)	Copies the selected item in the drop-down list to another list control.
	DeleteSelected (see page 2214)	Deletes the selected item from the drop-down list.
	Focused (see page 2215)	Indicates whether the combo box has input focus.
	GetCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
	SelectAll (see page 2220)	Selects the entire block of text in the edit box.

29.1.8.5.1 TCustomCombo.AddItem Method

Adds an item to the drop-down list of items.

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(UnicodeString Item, TObject * AObject);
```

Description

Call AddItem to add a string, with an associated object, to the drop-down list.

Item is the string to add to the drop-down list.

AObject is an object associated with that string. It can be accessed using the Objects property of the TStrings object that implements the Items (see page 2227) property.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;
```

```
    //adding items to the combo box
    comb.AddItem('firstChoice', nil);
    comb.AddItem('secondChoice', nil);
    comb.AddItem('thirdChoice', nil);

    //setting the default value
    comb.ItemIndex := 1;
  end;
```

See Also

Items (see page 2227)

Objects

29.1.8.5.2 TCustomCombo.AdjustDropDown Method

Adjusts the size and position of the drop down list.

Pascal

```
procedure AdjustDropDown; virtual;
```

C++

```
virtual __fastcall AdjustDropDown();
```

Description

TCustomCombo (see page 2203) calls AdjustDropDown when it drops down the drop-down list. This method adjusts the size and position of the drop-down list to reflect the values of the ItemHeight (see page 2226) and DropDownCount (see page 2225) properties.

See Also

ItemHeight (see page 2226)

DroppedDown (see page 2225)

DropDownCount (see page 2225)

29.1.8.5.3 TCustomCombo.Change Method

Generates an OnChange (see page 2237) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange (see page 2237) event. Override this method to add additional processing when the text changes, or to block the OnChange (see page 2237) event.

See Also

OnChange (see page 2237)

Changed

29.1.8.5.4 TCustomCombo.Clear Method

Deletes all text from the edit box and all items from the Items (see page 2227) list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the combo box at runtime.

See Also

Items (see page 2227)

Text

29.1.8.5.5 TCustomCombo.ClearSelection Method

Deselects any selected items in the drop-down list.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

ClearSelection checks whether an item in the drop-down list is currently selected. If so, it deselects the item (sets ItemIndex to –1), and changes the Text property to an empty string.

See Also

ItemIndex

Text

29.1.8.5.6 TCustomCombo.CloseUp Method

Generates an OnCloseUp (see page 2239) event.

Pascal

```
procedure CloseUp; dynamic;
```

C++

```
__fastcall CloseUp();
```

Description

CloseUp is called automatically when the DroppedDown (see page 2225) property changes to false. It generates an OnCloseUp (see page 2239) event.

Override CloseUp to implement additional class-specific responses when the drop-down list closes up or to suppress the event.

See Also

DroppedDown (see page 2225)

OnCloseUp (see page 2239)

29.1.8.5.7 TCustomCombo.ComboWndProc Method

Responds to Windows messages sent to the edit control or drop-down list.

Pascal

```
procedure ComboWndProc(var Message: TMessage; ComboWnd: HWnd; ComboProc: TWindowProcPtr);
virtual;
```

C++

```
virtual __fastcall ComboWndProc(TMessage Message, HWnd ComboWnd, TWindowProcPtr ComboProc);
```

Description

ComboWndProc handles the common processing of Windows messages sent to the edit region or list portion of the combo box. It is called by the EditWndProc (see page 2215) and ListWndProc (see page 2218) methods to take care of all processing that is not specific to the edit region or drop-down list.

The Message parameter describes the Windows message. The ComboWnd parameter is the windows handle for the edit region or list. The ComboProc parameter is a pointer to the default window procedure for the window that received the message. ComboWndProc provides default responses to focus events, mouse events, and key strokes.

Override ComboWndProc to change the way the combo box responds to Windows messages. When overriding ComboWndProc, be sure to call the parent class's method to process all unhandled messages.

Note: Do not confuse ComboWndProc with the WndProc (see page 2223) method. ComboWndProc responds to messages that are sent to the edit region or drop-down list. WndProc (see page 2223) responds to messages sent to the combo box itself.

See Also

WndProc (see page 2223)

EditWndProc (see page 2215)

ListWndProc (see page 2218)

29.1.8.5.8 TCustomCombo.CopySelection Method

Copies the selected item in the drop-down list to another list control.

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

CopySelection checks whether an item is selected in the drop-down list, and if so, adds it to the control specified by Destination, along with its associated object.

See Also

Items (see page 2227)

ItemIndex

29.1.8.5.9 TCustomCombo.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo (see page 2203) overrides the inherited method to ensure that the newly-created window and the TCustomCombo (see page 2203) properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo (see page 2203) instance.

See Also

CreateHandle

29.1.8.5.10 TCustomCombo.DeleteSelected Method

Deletes the selected item from the drop-down list.

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

DeleteSelected deletes the item specified by the ItemIndex property from the drop-down list.

See Also

ItemIndex

Items (see page 2227)

29.1.8.5.11 TCustomCombo.DestroyWindowHandle Method

Destroys all of the windows controls that implement the combo box.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

The DestroyWnd method calls DestroyWindowHandle to release the underlying Windows control. TCustomCombo (see page 2203) overrides the inherited method to reset the EditHandle (see page 2225) and ListHandle (see page 2229) properties once the underlying Windows controls are gone.

See Also

DestroyWnd

EditHandle (see page 2225)

ListHandle (see page 2229)

29.1.8.5.12 TCustomCombo.DropDown Method

Generates an OnDropDown (see page 2239) event.

Pascal

```
procedure DropDown; dynamic;
```

C++

```
__fastcall DropDown();
```

Description

DropDown is called automatically when the DroppedDown (see page 2225) property changes to true. It responds by generating an OnDropDown (see page 2239) event. To perform additional processing or block the OnDropDown (see page 2239) event, override DropDown.

See Also

OnDropDown (see page 2239)

DroppedDown (see page 2225)

29.1.8.5.13 TCustomCombo.EditWndProc Method

Responds to Windows messages that are sent to the edit control portion of the combo box.

Pascal

```
procedure EditWndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall EditWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditWndProc is the window procedure for the edit control portion of the combo box.

See Also

EditHandle (see page 2225)

WndProc (see page 2223)

ListWndProc (see page 2218)

ComboWndProc (see page 2213)

29.1.8.5.14 TCustomCombo.Focused Method

Indicates whether the combo box has input focus.

Pascal

```
function Focused: Boolean; override;
```

C++

```
virtual __fastcall Boolean Focused();
```

Description

Call Focused to determine whether the combo box is currently the recipient of keyboard input. When Focused returns true, the combo box is the active control and keystroke messages are sent to it. When Focused is false, the combo box is not active (although it may be selected).

See Also

ActiveControl

29.1.8.5.15 TCustomCombo.GetCount Method

Returns the value of the ItemCount (see page 2226) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount returns the number of items in the drop-down list.

See Also

ItemCount (see page 2226)

29.1.8.5.16 TCustomCombo.GetDroppedDown Method

Returns the value of the DroppedDown (see page 2225) property.

Pascal

```
function GetDroppedDown: Boolean;
```

C++

```
__fastcall Boolean GetDroppedDown();
```

Description

GetDroppedDown is the read implementation of the DroppedDown (see page 2225) property. It returns true if the drop-down list is showing.

See Also

DroppedDown (see page 2225)

29.1.8.5.17 TCustomCombo.GetItemCount Method

Returns the value of the ItemCount (see page 2226) property.

Pascal

```
function GetItemCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemCount() = 0;
```

Description

GetItemCount is the protected read access method for the ItemCount (see page 2226) property. In TCustomCombo (see page 2203), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the number of items in the drop-down list.

See Also

ItemCount (see page 2226)

29.1.8.5.18 TCustomCombo.GetItemHt Method

Returns the value of the ItemHeight (see page 2226) property.

Pascal

```
function GetItemHt: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemHt() = 0;
```

Description

GetItemHt is the protected read access method for the ItemHeight (see page 2226) property. In TCustomCombo (see page 2203), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the height of items in the drop-down list.

See Also

ItemHeight (see page 2226)

29.1.8.5.19 TCustomCombo.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the read implementation of the ItemIndex property. It returns the index of the selected item in the drop-down list. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

29.1.8.5.20 TCustomCombo.GetItemsClass Method

Indicates the class that implements the Items (see page 2227) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass() = 0;
```

Description

TCustomCombo (see page 2203) uses GetItemsClass internally to determine what class to instantiate when it implements the Items (see page 2227) property.

In TCustomCombo (see page 2203), GetItemsClass is abstract pure virtual, meaning it has no implementation. TCustomCombo (see page 2203) descendants override this method to indicate a TCustomComboBoxStrings (see page 2257) descendant that implements the Items (see page 2227) property.

See Also

Items (see page 2227)

29.1.8.5.21 TCustomCombo.GetSelLength Method

Returns the value of the SelLength (see page 2230) property.

Pascal

```
function GetSelLength: Integer;
```

C++

```
__fastcall int GetSelLength();
```

Description

GetSelLength is the read implementation of the SelLength (see page 2230) property. It returns the number of selected characters.

See Also

SelLength (see page 2230)

29.1.8.5.22 TCustomCombo.GetSelStart Method

Returns the value of the SelStart (see page 2233) property.

Pascal

```
function GetSelStart: Integer;
```

C++

```
__fastcall int GetSelStart();
```

Description

GetSelStart is the read implementation of the SelStart (see page 2233) property. It returns the position of the first selected character, or the position of the cursor if no characters are selected.

See Also

SelStart (see page 2233)

29.1.8.5.23 TCustomCombo.ListWndProc Method

Responds to Windows messages that are sent to the drop-down list portion of the combo box.

Pascal

```
procedure ListWndProc(var Message: TMessage);
```

C++

```
__fastcall ListWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListWndProc is the window procedure for the drop-down list.

See Also

ListHandle (see page 2229)

WndProc (see page 2223)

EditWndProc (see page 2215)

ComboWndProc (see page 2213)

29.1.8.5.24 TCustomCombo.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form from a stream.

TCustomCombo (see page 2203) overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded

ItemIndex

29.1.8.5.25 TCustomCombo.Select Method

Generates an OnSelect (see page 2241) event.

Pascal

```
procedure Select; dynamic;
```

C++

```
__fastcall Select();
```

Description

Select is called automatically when the selected item in the drop-down list changes. It responds by generating an OnSelect (see page 2241) event. To perform additional processing or block the OnSelect (see page 2241) event, override Select.

See Also

OnSelect (see page 2241)

ItemIndex

29.1.8.5.26 TCustomCombo.SelectAll Method

Selects the entire block of text in the edit box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the edit box. To select only part of the text, use the SelStart (see page 2233) and SelLength (see page 2230) properties.

See Also

SelStart (see page 2233)

SelLength (see page 2230)

29.1.8.5.27 TCustomCombo.SetDropDownCount Method

Sets the value of the DropDownCount (see page 2225) property.

Pascal

```
procedure SetDropDownCount(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetDropDownCount(const int Value);
```

Description

SetDropDownCount is the protected write implementation of the DropDownCount (see page 2225) property.

See Also

DropDownCount (see page 2225)

29.1.8.5.28 TCustomCombo.SetDroppedDown Method

Sets the value of the DroppedDown (see page 2225) property.

Pascal

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

C++

```
__fastcall SetDroppedDown(Boolean Value);
```

Description

SetDroppedDown is the protected write implementation of the DroppedDown (see page 2225) property.

See Also

DroppedDown (see page 2225)

29.1.8.5.29 TCustomCombo.SetItemHeight Method

Sets the value of the ItemHeight (see page 2226) property.

Pascal

```
procedure SetItemHeight(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetItemHeight(int Value);
```

Description

SetItemHeight is the protected write implementation of the ItemHeight (see page 2226) property. Although SetItemHeight never raises an exception, in some cases it does not change the value of the ItemHeight (see page 2226) property. ItemHeight (see page 2226) can only be changed for some TCustomCombo (see page 2203) descendants, and then only under specific circumstances.

For example, TComboBoxEx never allows the ItemHeight (see page 2226) property to change.

See Also

ItemHeight (see page 2226)

29.1.8.5.30 TCustomCombo.SetItemIndex Method

Sets the value of the ItemIndex property.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation of the ItemIndex property. It selects an item in the drop-down list.

Value is the item to select, where 0 indicates the first item, 1 indicates the second item, and so on. A negative Value causes all items to be unselected.

See Also

ItemIndex

29.1.8.5.31 TCustomCombo.SetItems Method

Sets the value of the Items (see page 2227) property.

Pascal

```
procedure SetItems(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetItems(const TStrings Value);
```

Description

SetItems is the protected write implementation for the Items (see page 2227) property. It copies the strings (and possibly their associated objects) from Value to the Items (see page 2227) property.

See Also

Items (see page 2227)

Assign

29.1.8.5.32 TCustomCombo.SetMaxLength Method

Sets the value of the MaxLength (see page 2229) property.

Pascal

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

C++

```
__fastcall SetMaxLength(int Value);
```

Description

SetMaxLength is the protected write implementation of the MaxLength (see page 2229) property.

See Also

MaxLength (see page 2229)

29.1.8.5.33 TCustomCombo.SetSelLength Method

Sets the value of the SelLength (see page 2230) property.

Pascal

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

C++

```
__fastcall SetSelLength(int Value);
```

Description

SetSelLength is the protected write implementation of the SelLength (see page 2230) property.

See Also

SelLength (see page 2230)

29.1.8.5.34 TCustomCombo.SetSelStart Method

Sets the value of the SelStart (see page 2233) property.

Pascal

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

C++

```
__fastcall SetSelStart(int Value);
```

Description

SetSelStart is the protected write implementation of the SelStart (see page 2233) property.

See Also

SelStart (see page 2233)

29.1.8.5.35 TCustomCombo.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 2215) or ListWndProc (see page 2218) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 2213)

EditWndProc (see page 2215)

ListWndProc (see page 2218)

29.1.8.6 TCustomCombo Properties

The properties of the TCustomCombo class are listed here.

Protected Properties

	Name	Description
	DropDownCount (see page 2225)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 2225)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2226)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 2226)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 2229)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2229)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 2230)	Determines where a control looks for its color information.

Public Properties

	Name	Description
R	Canvas (see page 2224)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2225)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2227)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2230)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2233)	Specifies the position of the first selected character in the edit portion of the combo box.
	TabStop (see page 2236)	Determines if the user can tab to a control.

29.1.8.6.1 TCustomCombo.Canvas Property

Provides access to the drawing surface of the combo box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use Canvas as the drawing surface when customizing the way the combo box paints itself. For example, when using a TCustomCombo (see page 2203) descendant that allows applications to draw combo box items in an OnDrawItem even, the OnDrawItem event handler uses Canvas for painting the customized items.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TComboBox *myComboBox;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TComboBox")
  {
    myComboBox = dynamic_cast<TComboBox *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myComboBox->Canvas->MoveTo(myComboBox->ClientRect.Left, myComboBox->ClientRect.Top);
    myComboBox->Canvas->LineTo(myComboBox->ClientRect.Right, myComboBox->ClientRect.Bottom);
    myComboBox = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TComboBox) then
  begin
    myComboBox:= (ListBox1.Items.Objects[Index] as TComboBox);
    with myComboBox.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myComboBox.Canvas.MoveTo(Left, Top);
      myComboBox.Canvas.LineTo(Right, Bottom);
    end;
  myComboBox:= nil;
  end;
```

See Also

OnDrawItem (see page 2256)

29.1.8.6.2 TCustomCombo.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items (see page 2227) property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items (see page 2227) property, the drop down list will display a scroll bar.

See Also

Items (see page 2227)

ItemCount (see page 2226)

29.1.8.6.3 TCustomCombo.DroppedDown Property

Indicates whether the drop-down list is currently displayed.

Pascal

```
property DroppedDown: Boolean;
```

C++

```
__property Boolean DroppedDown;
```

Description

Read (see page 3727) DroppedDown to determine whether the drop-down list is visible. Set DropDown (see page 2215) to drop the list down or close it up. DroppedDown is true when the drop-down list is open, and false when it is closed.

See Also

OnDropDown (see page 2239)

AutoDropDown (see page 2253)

29.1.8.6.4 TCustomCombo.EditHandle Property

Returns the window handle for the edit portion of the combo box.

Pascal

```
property EditHandle: HWnd;
```

C++

```
__property HWnd EditHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditHandle is the

window handle of the edit control portion. (The Handle property and the ListHandle (see page 2229) property give the handles for the combo box itself and the drop-down list, respectively.)

Use EditHandle when making Windows API calls addressed to the edit portion of the combo box.

See Also

Handle

ListHandle (see page 2229)

29.1.8.6.5 TCustomCombo.ItemCount Property

Specifies the number of items in the drop-down list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Read (see page 3727) ItemCount to get the number of items in the drop-down list. ItemCount is the number of all items, not just those that are visible at one time.

See Also

Items (see page 2227)

DropDownCount (see page 2225)

29.1.8.6.6 TCustomCombo.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read (see page 3727) ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox (see page 2106) lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 2227)

OnMeasureItem (see page 2257)

Style (see page 2255)

29.1.8.6.7 TCustomCombo.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read (see page 3727) Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", reinterpret_cast<TObject *>(vsIcon));
  ComboBox1->Items->AddObject("vsList", reinterpret_cast<TObject *>(vsList));
  ComboBox1->Items->AddObject("vsReport", reinterpret_cast<TObject *>(vsReport));
  ComboBox1->Items->AddObject("vsSmallIcon", reinterpret_cast<TObject *>(vsSmallIcon));
  // Display first item in the Combo Box
```

```
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
```

```
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
```

See Also

ItemIndex

TStrings

Assign

29.1.8.6.8 TCustomCombo.ListHandle Property

Provides access to the window handle of the list of items in the combo box.

Pascal

```
property ListHandle: HWnd;
```

C++

```
__property HWnd ListHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListHandle is the window handle of the drop-down list. (The Handle property and the EditHandle (see page 2225) property give the handles for the combo box itself and the edit portion, respectively.)

Use ListHandle when making Windows API calls addressed to the list portion of the combo box.

Note: ListHandle is only available when the combo box has a "simple" style (an edit box with a fixed list below rather than a drop-down list).

See Also

Handle

EditHandle (see page 2225)

29.1.8.6.9 TCustomCombo.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that

may be entered into an edit control.

See Also

Text

29.1.8.6.10 TCustomCombo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.8.6.11 TCustomCombo.SelLength Property

Specifies the length, in characters, of the selected text in the edit portion of the combo box.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

The selected portion of the text in the edit region of the combo box is the string of SelLength characters that starts at the character specified by the SelStart (see page 2233) property. The selected text appears highlighted in the edit region of the combo box.

To select a section of text in the edit region, first set the SelStart (see page 2233) property to position the cursor, and then set SelLength to the number of characters that should be selected. The SelText property is the substring that is selected.

Note: The SelLength property is only valid when the combo box has focus. When the combo box does not have focus, reading SelLength always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}

TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(String S)
{
```

```
  CharSetList->Add(S);
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TStringList> _CharSetListCleaner(CharSetList = new TStringList);
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
```

```
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelStart (see page 2233)

SelText (see page 2253)

Text

29.1.8.6.12 TCustomCombo.SelStart Property

Specifies the position of the first selected character in the edit portion of the combo box.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read (see page 3727) SelStart to determine where the selection starts in the Text property. The first character is designated with 0, the second with 1, and so on. When SelLength (see page 2230) is 0, SelStart is the position of the cursor.

Set SelStart to position the cursor in the edit region of the combo box. Setting SelStart to 0 positions the cursor before the first character, setting SelStart to 1 positions the cursor before the second character, and so on.

Note: Setting SelStart changes the SelLength (see page 2230) property to 0. When using SelStart and SelLength (see page 2230) to programmatically select a portion of the text in the edit region, set SelStart before setting SelLength (see page 2230).

Note: The SelStart property is only valid when the combo box has focus. When the combo box does not have focus, reading SelStart always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}

TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(String S)
{
  CharSetList->Add(S);
}
```

```
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TStringList> _CharSetListCleaner(CharSetList = new TStringList);
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
```

```
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelLength (see page 2230)

SelText (see page 2253)

Text

29.1.8.6.13 TCustomCombo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.8.7 TCustomCombo Events

The events of the TCustomCombo class are listed here.

Protected Events

	Name	Description
	OnChange (see page 2237)	Occurs when the user changes the text displayed in the edit region.

	OnCloseUp (see page 2239)	Occurs when the drop-down list closes up due to some user action.
	OnDropDown (see page 2239)	Occurs when the user opens the drop-down list.
	OnSelect (see page 2241)	Occurs when the user selects a string in the drop-down list.

29.1.8.7.1 TCustomCombo.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3789) an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
```

```
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
    end;
  ComboBox1.ItemIndex := 0;
  PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
```

```
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text

OnCloseUp (see page 2239)

29.1.8.7.2 TCustomCombo.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write (see page 3789) an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 2239)

DroppedDown (see page 2225)

29.1.8.7.3 TCustomCombo.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write (see page 3789) an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
```

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 2225)

OnCloseUp (see page 2239)

29.1.8.7.4 TCustomCombo.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write (see page 3789) an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 2251)

ItemIndex

Text

29.1.9 TCustomComboBox Class

TCustomComboBox is the base class for combobox components.

Class Hierarchy

Pascal

```
TCustomComboBox = class(TCustomCombo);
```

C++

```
class TCustomComboBox : public TCustomCombo;
```

File

StdCtrls

Description

Use TCustomComboBox as a base class when defining a control that represents a specialized Windows combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Note: To create a combo box class that uses the extended combo box features such as images associated with entries, do not use TCustomComboBox. Instead, use the TCustomComboBoxEx class instead.

Do not create instances of TCustomComboBox. To put a combo box in a form, use a descendant of TCustomComboBox, such as TComboBox (see page 2106).

See Also

TComboBox (see page 2106)

TDBComboBox

TDriveComboBox

TEdit (see page 2380)

TFilterComboBox

TListBox (see page 2572)

TColorBox

TCustomComboBoxEx

29.1.9.1 TCustomComboBox Members

The following tables list the members exposed by TCustomComboBox.

Protected Events

	Name	Description
	OnChange (see page 2237)	Occurs when the user changes the text displayed in the edit region.
	OnCloseUp (see page 2239)	Occurs when the drop-down list closes up due to some user action.
	OnDrawItem (see page 2256)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2239)	Occurs when the user opens the drop-down list.
	OnMeasureItem (see page 2257)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnSelect (see page 2241)	Occurs when the user selects a string in the drop-down list.

Protected Fields

	Name	Description
	FDefEditProc (see page 2206)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 2206)	This is FDefListProc, a member of class TCustomCombo.
	FDropHandle (see page 2207)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 2207)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 2207)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 2207)	This is FEditInstance, a member of class TCustomCombo.
	FFocusChanged (see page 2207)	This is FFocusChanged, a member of class TCustomCombo.
	FIsFocused (see page 2208)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 2208)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 2208)	This is FListInstance, a member of class TCustomCombo.
	FSaveIndex (see page 2208)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	AdjustDropDown (see page 2211)	Adjusts the size and position of the drop down list.
	Change (see page 2211)	Generates an OnChange (see page 2237) event.
	CloseUp (see page 2212)	Generates an OnCloseUp (see page 2239) event.
	ComboWndProc (see page 2213)	Responds to Windows messages sent to the edit control or drop-down list.

	CreateParams (see page 2245)	Initializes the window-creation parameter record.
	CreateWnd (see page 2245)	Creates the Windows control that represents the combo box.
	DestroyWindowHandle (see page 2214)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 2246)	Destroys the Windows control that represents the combo box.
	DoSetTextHint (see page 2246)	This is DoSetTextHint, a member of class TCustomComboBox.
	DrawItem (see page 2246)	Generates an OnDrawItem (see page 2256) event.
	DropDown (see page 2215)	Generates an OnDropDown (see page 2239) event.
	EditWndProc (see page 2215)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2216)	Returns the value of the DroppedDown (see page 2225) property.
	GetItemCount (see page 2247)	Returns the number of items in the Items (see page 2227) property list.
	GetItemHt (see page 2247)	Returns the value of the ItemHeight (see page 2226) property.
	GetItemIndex (see page 2217)	Returns the value of the ItemIndex property.
	GetItemsClass (see page 2247)	Returns the class used to implement the Items (see page 2227) property.
	GetSelLength (see page 2218)	Returns the value of the SelLength (see page 2230) property.
	GetSelStart (see page 2218)	Returns the value of the SelStart (see page 2233) property.
	GetSelText (see page 2248)	Returns the value of the SelText (see page 2253) property.
	KeyPress (see page 2248)	Responds to WM_CHAR messages sent to the combo box.
	ListWndProc (see page 2218)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
	Loaded (see page 2219)	Initializes the combo box after its form is loaded into memory.
	MeasureItem (see page 2249)	Generates an OnMeasureItem (see page 2257) event.
	Select (see page 2219)	Generates an OnSelect (see page 2241) event.
	SelectItem (see page 2249)	Selects a specified item from the drop-down list.
	SetDropDownCount (see page 2220)	Sets the value of the DropDownCount (see page 2225) property.
	SetDroppedDown (see page 2220)	Sets the value of the DroppedDown (see page 2225) property.
	SetItemHeight (see page 2221)	Sets the value of the ItemHeight (see page 2226) property.
	SetItemIndex (see page 2221)	Sets the value of the ItemIndex property.
	SetItems (see page 2221)	Sets the value of the Items (see page 2227) property.
	SetMaxLength (see page 2222)	Sets the value of the MaxLength (see page 2229) property.
	SetSelLength (see page 2222)	Sets the value of the SelLength (see page 2230) property.
	SetSelStart (see page 2222)	Sets the value of the SelStart (see page 2233) property.
	SetStyle (see page 2249)	Sets the Style (see page 2255) property.
	WndProc (see page 2250)	Processes messages received by the combo box.

Protected Properties

	Name	Description
	DropDownCount (see page 2225)	Specifies the maximum number of items displayed in the drop-down list.
	EditHandle (see page 2225)	Returns the window handle for the edit portion of the combo box.
	ItemCount (see page 2226)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 2226)	Specifies the height, in pixels, of the items in the drop-down list.
	ListHandle (see page 2229)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2229)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 2230)	Determines where a control looks for its color information.
	Sorted (see page 2254)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2255)	Determines the display style of the combo box.

Public Methods

	Name	Description
	AddItem (see page 2209)	Adds an item to the drop-down list of items.
	Clear (see page 2212)	Deletes all text from the edit box and all items from the Items (see page 2227) list.
	ClearSelection (see page 2212)	Deselects any selected items in the drop-down list.
	CopySelection (see page 2213)	Copies the selected item in the drop-down list to another list control.
	Create (see page 2244)	Creates and initializes a TCustomComboBox (see page 2241) instance.

	DeleteSelected (see page 2214)	Deletes the selected item from the drop-down list.
	Destroy (see page 2244)	Destroys the combo box and frees its memory.
	Focused (see page 2215)	Indicates whether the combo box has input focus.
	GetCount (see page 2216)	Returns the value of the ItemCount (see page 2226) property.
	SelectAll (see page 2220)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
	AutoCloseUp (see page 2250)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2251)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2252)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2253)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	Canvas (see page 2224)	Provides access to the drawing surface of the combo box.
	CharCase (see page 2253)	Determines the case of the text in the combo box.
	DroppedDown (see page 2225)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2227)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2230)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2233)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 2253)	Represents the selected text in the edit region.
	TabStop (see page 2236)	Determines if the user can tab to a control.
	TextHint (see page 2256)	This is TextHint, a member of class TCustomComboBox.

29.1.9.2 TCustomComboBox.Create Constructor

Creates and initializes a TCustomComboBox (see page 2241) instance.

Pascal

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

C++

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

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the combo box instance.

See Also

Owner

29.1.9.3 TCustomComboBox.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

29.1.9.4 TCustomComboBox Methods

The methods of the TCustomComboBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2245)	Initializes the window-creation parameter record.
	CreateWnd (see page 2245)	Creates the Windows control that represents the combo box.
	DestroyWnd (see page 2246)	Destroys the Windows control that represents the combo box.
	DoSetTextHint (see page 2246)	This is DoSetTextHint, a member of class TCustomComboBox.
	DrawItem (see page 2246)	Generates an OnDrawItem (see page 2256) event.
	GetItemCount (see page 2247)	Returns the number of items in the Items (see page 2227) property list.
	GetItemHt (see page 2247)	Returns the value of the ItemHeight (see page 2226) property.
	GetItemsClass (see page 2247)	Returns the class used to implement the Items (see page 2227) property.
	GetSelText (see page 2248)	Returns the value of the SelText (see page 2253) property.
	KeyPress (see page 2248)	Responds to WM_CHAR messages sent to the combo box.
	MeasureItem (see page 2249)	Generates an OnMeasureItem (see page 2257) event.
	SelectItem (see page 2249)	Selects a specified item from the drop-down list.
	SetStyle (see page 2249)	Sets the Style (see page 2255) property.
	WndProc (see page 2250)	Processes messages received by the combo box.

29.1.9.4.1 TCustomComboBox.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2245) method calls CreateParams to initialize the parameters it uses to create the window for the combo box. CreateParams overrides the inherited method to implement the Style (see page 2255) property.

See Also

CreateWnd (see page 2245)

Style (see page 2255)

29.1.9.4.2 TCustomComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle

29.1.9.4.3 TCustomComboBox.DestroyWnd Method

Destroys the Windows control that represents the combo box.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. TCustomComboBox (see page 2241) calls DestroyWnd to destroy this Windows control when necessary.

DestroyWnd saves the contents of Items (see page 2227) list before calling the inherited DestroyWnd method. The saved list is used if the window is recreated.

See Also

CreateWnd (see page 2245)

29.1.9.4.4 TCustomComboBox.DoSetTextHint Method

Pascal

```
procedure DoSetTextHint; virtual;
```

C++

```
virtual __fastcall DoSetTextHint();
```

Description

This is DoSetTextHint, a member of class TCustomComboBox.

29.1.9.4.5 TCustomComboBox.DrawItem Method

Generates an OnDrawItem (see page 2256) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

When Style (see page 2255) is csOwnerDrawFixed or csOwnerDrawVariable, DrawItem is called whenever an item in the list needs to be drawn. If an OnDrawItem (see page 2256) event handler is assigned, DrawItem generates an OnDrawItem (see page 2256) event, passing the Index, Rect, and State parameters to the event handler. Otherwise, OnDrawItem (see page 2256) fills the rectangle passed in Rect and draws any text associated with the indexed item, ignoring the State parameter.

Override DrawItem to provide class-specific drawing code for the items in the list.

See Also

OnDrawItem (see page 2256)

Style (see page 2255)

29.1.9.4.6 TCustomComboBox.GetItemCount Method

Returns the number of items in the Items (see page 2227) property list.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

GetItemCount is used internally to quickly determine the number of items in the drop-down list of the combo box.

See Also

Items (see page 2227)

29.1.9.4.7 TCustomComboBox.GetItemHt Method

Returns the value of the ItemHeight (see page 2226) property.

Pascal

```
function GetItemHt: Integer; override;
```

C++

```
virtual __fastcall int GetItemHt();
```

Description

GetItemHt is the protected read implementation of the ItemHeight (see page 2226) property.

See Also

ItemHeight

29.1.9.4.8 TCustomComboBox.GetItemsClass Method

Returns the class used to implement the Items (see page 2227) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; override;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass();
```

Description

Combo box controls use GetItemsClass internally to determine what class to use when implementing the Items (see page 2227) property. This allows the TCustomCombo (see page 2203) base class to implement the Items (see page 2227) property for both TCustomComboBox (see page 2241) descendants and TCustomComboBoxEx descendants.

See Also

Items (see page 2227)

29.1.9.4.9 TCustomComboBox.GetSelText Method

Returns the value of the SelText (see page 2253) property.

Pascal

```
function GetSelText: string;
```

C++

```
__fastcall UnicodeString GetSelText();
```

Description

GetSelText is the protected read implementation of the SelText (see page 2253) property.

See Also

SelText (see page 2253)

29.1.9.4.10 TCustomComboBox.KeyPress Method

Responds to WM_CHAR messages sent to the combo box.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress processes standard keystroke messages for the combo box control. If the KeyPress method does not handle the keystroke, it calls the inherited method, which generates an OnKeyPress event.

Key is the character represented by the key that was pressed. If the KeyPress method changes Key to 0, no further processing occurs for the keystroke.

See Also

KeyDown

KeyUp

OnKeyPress

29.1.9.4.11 TCustomComboBox.MeasureItem Method

Generates an OnMeasureItem (see page 2257) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is the protected implementation method for the OnMeasureItem (see page 2257) event. Override MeasureItem to provide class-specific code to specify the size of list items when Style (see page 2255) is csOwnerDrawVariable.

See Also

OnMeasureItem (see page 2257)

Style (see page 2255)

29.1.9.4.12 TCustomComboBox.SelectItem Method

Selects a specified item from the drop-down list.

Pascal

```
function SelectItem(const AnItem: string): Boolean;
```

C++

```
__fastcall Boolean SelectItem(const UnicodeString AnItem);
```

Description

SelectItem selects the first item in the drop-down list that matches the string specified by AnItem. If the list does not contain a string that matches AnItem, SelectItem does nothing.

See Also

Items (see page 2227)

ItemIndex

29.1.9.4.13 TCustomComboBox.SetStyle Method

Sets the Style (see page 2255) property.

Pascal

```
procedure SetStyle(Value: TComboBoxStyle); virtual;
```

C++

```
virtual __fastcall SetStyle(TComboBoxStyle Value);
```

Description

SetStyle is the protected implementation method for the Style (see page 2255) property. Override SetStyle to change the implementation of the Style (see page 2255) property.

See Also

Style (see page 2255)

29.1.9.4.14 TCustomComboBox.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 2215) or ListWndProc (see page 2218) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 2213)

System::TObject::Dispatch (see page 3379)

WndProc

EditWndProc (see page 2215)

ListWndProc (see page 2218)

29.1.9.5 TCustomComboBox Properties

The properties of the TCustomComboBox class are listed here.

Protected Properties

	Name	Description
	Sorted (see page 2254)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2255)	Determines the display style of the combo box.

Public Properties

	Name	Description
	AutoCloseUp (see page 2250)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2251)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2252)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2253)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	CharCase (see page 2253)	Determines the case of the text in the combo box.
	SelText (see page 2253)	Represents the selected text in the edit region.
	TextHint (see page 2256)	This is TextHint, a member of class TCustomComboBox.

29.1.9.5.1 TCustomComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 2253)

29.1.9.5.2 TCustomComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay (see page 2252) milliseconds between the characters typed and the autocomplete operation.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

AutoCompleteDelay (see page 2252)

29.1.9.5.3 TCustomComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2251) is true.

See Also

AutoComplete (see page 2251)

29.1.9.5.4 TCustomComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 2251)

OnDropDown (see page 2239)

DroppedDown (see page 2225)

29.1.9.5.5 TCustomComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text

29.1.9.5.6 TCustomComboBox.SelText Property

Represents the selected text in the edit region.

Pascal

```
property SelText: string;
```

C++

```
__property UnicodeString SelText;
```

Description

Read (see page 3727) SelText to obtain the text that is selected in the edit region of the combo box. When the combo box is

the active control, SelText appears highlighted. Set SelText to replace the current selection with a different substring. If no text is selected when a value is assigned to SelText, the SelText string is inserted in the text at the cursor.

Note: The SelText property is only valid when the combo box has focus. When the combo box does not have focus, reading SelText always yields an empty string and setting it inserts the text at the cursor rather than replacing the selected text.

See Also

SelLength (see page 2230)

SelStart (see page 2233)

29.1.9.5.7 TCustomComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items (see page 2227) list alphabetically. New (see page 3712) items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
```

```
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items

Using TDBListBox and TDBComboBox

29.1.9.5.8 TCustomComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem (see page 2256) event occurs when the list

items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem (see page 2257) event occurs as well.

See Also

Height

ItemHeight

OnDrawItem (see page 2256)

OnMeasureItem (see page 2257)

29.1.9.5.9 TCustomComboBox.TextHint Property

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

This is TextHint, a member of class TCustomComboBox.

29.1.9.6 TCustomComboBox Events

The events of the TCustomComboBox class are listed here.

Protected Events

	Name	Description
	OnDrawItem (see page 2256)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnMeasureItem (see page 2257)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

29.1.9.6.1 TCustomComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write (see page 3789) an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas (see page 2224) using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style (see page 2255) is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas

Items

OnMeasureItem (see page 2257)

Style (see page 2255)

29.1.9.6.2 TCustomComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style (see page 2255) is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem (see page 2256). Write (see page 3789) an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight

Items

OnDrawItem (see page 2256)

Style (see page 2255)

29.1.10 TCustomComboBoxStrings Class

TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.

Class Hierarchy

Pascal

```
TCustomComboBoxStrings = class(TStrings);
```

C++

```
class TCustomComboBoxStrings : public TStrings;
```

File

StdCtrls

Description

TCustomComboBoxStrings is the base class for objects that represent the items in the drop-down list of a combo box. Each type of combo box (TComboBox (see page 2106) or TComboBoxEx) uses its own TCustomComboBoxStrings descendant to manage the list of items.

Do not create instances of TCustomComboBoxStrings. This class is designed to work only with a combo box. TCustomCombo (see page 2203) creates the TCustomComboBoxStrings descendant it uses to implement its Items property.

See Also

TCustomCombo (see page 2203)

TComboBox (see page 2106)

TComboBoxEx

29.1.10.1 TCustomComboBoxStrings Members

The following tables list the members exposed by TCustomComboBoxStrings.

Protected Methods

	Name	Description
	Get (see page 2260)	Returns a string given its index.
	GetCount (see page 2261)	Returns the number of strings in the list
	GetObject (see page 2261)	Returns the object associated with the item at a specified index.
	PutObject (see page 2262)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 2262)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	ComboBox (see page 2263)	Identifies the combo box whose items this object represents.

Public Methods

	Name	Description
	Clear (see page 2259)	Deletes all the strings from the list.
	Create (see page 2258)	Constructs an object and initializes its data before the object is first used.
	Delete (see page 2260)	Removes the item specified by the Index parameter.
	Destroy (see page 2259)	Destroys the TPersistent instance and frees its memory.
	IndexOf (see page 2261)	Returns the position of a string in the list.

29.1.10.2 TCustomComboBoxStrings.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TCustomComboBoxStrings();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

29.1.10.3 TCustomComboBoxStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBoxStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

System::TObject::Destroy (see page 3366)

29.1.10.4 TCustomComboBoxStrings Methods

The methods of the TCustomComboBoxStrings class are listed here.

Protected Methods

	Name	Description
	Get (see page 2260)	Returns a string given its index.
	GetCount (see page 2261)	Returns the number of strings in the list
	GetObject (see page 2261)	Returns the object associated with the item at a specified index.
	PutObject (see page 2262)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 2262)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Clear (see page 2259)	Deletes all the strings from the list.
	Delete (see page 2260)	Removes the item specified by the Index parameter.
	IndexOf (see page 2261)	Returns the position of a string in the list.

29.1.10.4.1 TCustomComboBoxStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the drop-down list of the combo box. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 2260)

Objects

Strings

29.1.10.4.2 TCustomComboBoxStrings.Delete Method

Removes the item specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single item from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string in the drop-down list, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear (see page 2259)

IndexOf (see page 2261)

InsertObject

Objects

Strings

29.1.10.4.3 TCustomComboBoxStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property. It fetches the specified string from the combo box identified by the ComboBox (see page 2263) property.

Index identifies the string to return, where 0 is the index of the first string, 1 is the index of the second string, and so on.

See Also

Strings

IndexOf (see page 2261)

ComboBox (see page 2263)

29.1.10.4.4 TCustomComboBoxStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property. It fetches the number of items from the combo box identified by the ComboBox (see page 2263) property.

See Also

Count

ComboBox (see page 2263)

29.1.10.4.5 TCustomComboBoxStrings.GetObject Method

Returns the object associated with the item at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property. It fetches the associated object for the identified item from the combo box identified by the ComboBox (see page 2263) property.

Index is the index of the item with which the object is associated.

See Also

Objects

ComboBox (see page 2263)

29.1.10.4.6 TCustomComboBoxStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const UnicodeString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. The string must be an exact match (S can't be the prefix of a string in the combo box).

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second

string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

IndexOfName

IndexOfObject

Strings

29.1.10.4.7 TCustomComboBoxStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property. It associates the object specified by AObject with the item identified by Index.

Index identifies an item in the drop-down list of the combo box, where 0 is the index of the first item, 1 is the index of the second item, and so on.

AObject is the object to associated with the specified item.

See Also

Objects

29.1.10.4.8 TCustomComboBoxStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TCustomComboBoxStrings (see page 2257) object is not already in the middle of an update, SetUpdateState is called with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, SetUpdateState is called with Updating set to false.

SetUpdateState refreshes the combo box when Updating is false, so that the combo box reflects the new set of strings.

See Also

BeginUpdate

EndUpdate

29.1.10.5 TCustomComboBoxStrings Properties

The properties of the TCustomComboBoxStrings class are listed here.

Protected Properties

	Name	Description
	ComboBox (see page 2263)	Identifies the combo box whose items this object represents.

29.1.10.5.1 TCustomComboBoxStrings.ComboBox Property

Identifies the combo box whose items this object represents.

Pascal

```
property ComboBox: TCustomCombo;
```

C++

```
__property TCustomCombo ComboBox;
```

Description

TCustomComboBoxStrings (see page 2257) uses ComboBox internally so that the strings in the list and the items in the drop-down list of the combo box remain synchronized.

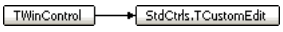
See Also

Items (see page 2227)

29.1.11 TCustomEdit Class

TCustomEdit is the base class from which all edit-box and memo controls are derived.

Class Hierarchy



Pascal

```
TCustomEdit = class(TWinControl);
```

C++

```
class TCustomEdit : public TWinControl;
```

File

StdCtrls

Description

TCustomEdit encapsulates the behavior common to all components for editing text by introducing methods and properties that provide:

Basic text editing functions such as selecting text, modifying selected text, and case conversions.

Ability to respond to changes in the contents of the text.

Access control of the text for making it read-only or introducing a password character to hide the actual value.

Do not create instances of TCustomEdit. Use TCustomEdit as a base class when declaring control objects that permit the user to

enter or modify text. Properties and methods of TCustomEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TCustomMaskEdit

TCustomMemo (see page 2361)

TEdit (see page 2380)

29.1.11.1 TCustomEdit Members

The following tables list the members exposed by TCustomEdit.

Protected Events

	Name	Description
	OnChange (see page 2299)	Occurs when the text for the edit control may have changed.

Protected Methods

	Name	Description
	Change (see page 2266)	Generates an OnChange (see page 2299) event.
V	CreateParams (see page 2269)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2270)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2270)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2273)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2273)	Sets the MaxLength (see page 2286) property.
V	DoSetTextHint (see page 2273)	This is DoSetTextHint, a member of class TCustomEdit.
V	GetSelLength (see page 2274)	Returns the SelLength (see page 2289) property.
V	GetSelStart (see page 2274)	Returns the SelStart (see page 2293) property.
V	GetSelText (see page 2275)	Returns the SelText (see page 2296) property.
	SetAlignment (see page 2279)	Sets the Alignment (see page 2283) property.
V	SetAutoSize (see page 2279)	Sets the AutoSize (see page 2284) property.
V	SetSelLength (see page 2280)	Sets the SelLength (see page 2289) property.
V	SetSelStart (see page 2280)	Sets the SelStart (see page 2293) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2284)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2284)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2285)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2286)	Determines the case of the text within the edit control.
	HideSelection (see page 2286)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2286)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2287)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2288)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2288)	Determines where a control looks for its color information.
	PasswordChar (see page 2289)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Methods

	Name	Description
	Clear (see page 2267)	Deletes all text from the edit control.
	ClearSelection (see page 2267)	Removes the selected text from the edit control.
	ClearUndo (see page 2268)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2268)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2265)	Creates an instance of TCustomEdit (see page 2263).
	CutToClipboard (see page 2271)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2272)	Provides message handling for all messages that the control does not fully process itself.
	GetControlsAlignment (see page 2274)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2275)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2276)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2278)	Selects all text in the edit control.
	SetSelText (see page 2281)	Sets the SelText (see page 2296) property.
	SetSelTextBuf (see page 2281)	Replaces the selected text with a null-terminated string.
	Undo (see page 2282)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	Alignment (see page 2283)	Determines how the text is aligned within the text edit control.
	CanUndo (see page 2285)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2287)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2289)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2289)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2293)	Specifies the position of the first selected character in the text.
	SelText (see page 2296)	Specifies the selected portion of the edit control's text.
	Text (see page 2297)	Contains a text string associated with the control.
	TextHint (see page 2299)	A hint or message to be displayed when the Text (see page 2297) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2297)	Determines if the user can tab to a control.

29.1.11.2 TCustomEdit.Create Constructor

Creates an instance of TCustomEdit (see page 2263).

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomEdit (see page 2263).

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner

29.1.11.3 TCustomEdit Methods

The methods of the TCustomEdit class are listed here.

Protected Methods

	Name	Description
	Change (see page 2266)	Generates an OnChange (see page 2299) event.
V	CreateParams (see page 2269)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2270)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2270)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2273)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2273)	Sets the MaxLength (see page 2286) property.
V	DoSetTextHint (see page 2273)	This is DoSetTextHint, a member of class TCustomEdit.
V	GetSelLength (see page 2274)	Returns the SelLength (see page 2289) property.
V	GetSelStart (see page 2274)	Returns the SelStart (see page 2293) property.
V	GetSelText (see page 2275)	Returns the SelText (see page 2296) property.
	SetAlignment (see page 2279)	Sets the Alignment (see page 2283) property.
V	SetAutoSize (see page 2279)	Sets the AutoSize (see page 2284) property.
V	SetSelLength (see page 2280)	Sets the SelLength (see page 2289) property.
V	SetSelStart (see page 2280)	Sets the SelStart (see page 2293) property.

Public Methods

	Name	Description
V	Clear (see page 2267)	Deletes all text from the edit control.
	ClearSelection (see page 2267)	Removes the selected text from the edit control.
	ClearUndo (see page 2268)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2268)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	CutToClipboard (see page 2271)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 2272)	Provides message handling for all messages that the control does not fully process itself.
V	GetControlsAlignment (see page 2274)	Indicates how text is aligned within the control.
V	GetSelTextBuf (see page 2275)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2276)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2278)	Selects all text in the edit control.
	SetSelText (see page 2281)	Sets the SelText (see page 2296) property.
	SetSelTextBuf (see page 2281)	Replaces the selected text with a null-terminated string.
	Undo (see page 2282)	Backs out all changes in the undo buffer.

29.1.11.3.1 TCustomEdit.Change Method

Generates an OnChange (see page 2299) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Applications do not call the Change method directly. It is triggered automatically by Windows events. Change is exposed as a protected method so that derived classes can make additional responses when the text changes beyond invoking the OnChange (see page 2299) event handler. If Derived classes that override the Change method do not call the inherited method, the OnChange (see page 2299) event handler will not be called.

See Also

Handle

OnChange (see page 2299)

Text

29.1.11.3.2 TCustomEdit.Clear Method

Deletes all text from the edit control.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to replace the Text (see page 2297) property with an empty string. Clear removes all text, not just the selected text. To remove just the selected text, use ClearSelection (see page 2267).

See Also

ClearSelection (see page 2267)

Text

29.1.11.3.3 TCustomEdit.ClearSelection Method

Removes the selected text from the edit control.

Pascal

```
procedure ClearSelection;
```

C++

```
__fastcall ClearSelection();
```

Description

Use ClearSelection to delete the selected text from the edit control. If no text is selected, ClearSelection does nothing. If all of the text is selected, ClearSelection clears all text, like the Clear (see page 2267) method.

Note: Calling ClearSelection does the same thing as sending the edit control a WM_CLEAR message.

See Also

Clear (see page 2267)

CutToClipboard (see page 2271)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.11.3.4 TCustomEdit.ClearUndo Method

Clears the undo buffer so that no changes to the text can be backed out.

Pascal

```
procedure ClearUndo;
```

C++

```
__fastcall ClearUndo();
```

Description

Use ClearUndo to commit changes to the Text (see page 2297) property. After calling ClearUndo, the CanUndo (see page 2285) property is false and the Undo (see page 2282) method does nothing.

See Also

Undo (see page 2282)

CanUndo (see page 2285)

Text

29.1.11.3.5 TCustomEdit.CopyToClipboard Method

Copies the selected text in the edit control to the Clipboard in CF_TEXT format.

Pascal

```
procedure CopyToClipboard;
```

C++

```
__fastcall CopyToClipboard();
```

Description

Use CopyToClipboard to replace the contents of the Clipboard with the selected text. CopyToClipboard does not clear the Clipboard if no text is selected. If no text is selected, CopyToClipboard does nothing.

Note: Calling CopyToClipboard does the same thing as sending the edit control a WM_COPY message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

CutToClipboard (see page 2271)

GetSelTextBuf (see page 2275)

PasteFromClipboard (see page 2276)

SelText (see page 2296)

Displaying and Editing Text in a Memo Control

29.1.11.3.6 TCustomEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2270) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 2270). TCustomEdit (see page 2263) overrides the inherited method to specify a scrollable single-line edit control

that reflects the current values of the BorderStyle (see page 2285), HideSelection (see page 2286), PasswordChar (see page 2289), and ReadOnly (see page 2289) properties.

See Also

BorderStyle (see page 2285)

CreateWindowHandle (see page 2270)

CreateWnd (see page 2270)

HideSelection (see page 2286)

PasswordChar (see page 2289)

ReadOnly (see page 2289)

29.1.11.3.7 TCustomEdit.CreateWindowHandle Method

Creates a Windows control to represent the control.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

CreateWindowHandle calls the inherited CreateWindowHandle method after working around a Win95 bug involving setting the read-only style on machines with Far East locales.

See Also

CreateParams (see page 2269)

CreateWnd (see page 2270)

DestroyWnd (see page 2273)

CreateWindowHandle

29.1.11.3.8 TCustomEdit.CreateWnd Method

Creates a Windows control corresponding to the edit control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the edit control is created, and whenever a change to the edit control requires the window to be recreated. After calling the inherited method, CreateWnd sends the newly created window messages to enforce the MaxLength (see page 2286) and PasswordChar (see page 2289) properties.

See Also

CreateParams (see page 2269)

CreateWindowHandle

DestroyWnd (see page 2273)

MaxLength (see page 2286)

PasswordChar (see page 2289)

CreateWnd

29.1.11.3.9 TCustomEdit.CutToClipboard Method

Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.

Pascal

```
procedure CutToClipboard;
```

C++

```
__fastcall CutToClipboard();
```

Description

Use CutToClipboard to replace the contents of the Clipboard with the selected text while clearing the selected text. If no text is selected, CutToClipboard does nothing.

Note: Calling CopyToClipboard (see page 2268) does the same thing as sending the edit control a WM_CUT message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
```

```
  control, the formatting information would be copied as well.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
    RichEdit1.CutToClipboard;
    Edit1.Clear;
    Edit1.PasteFromClipboard;
    RichEdit1.SetFocus;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const
    Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

ClearSelection (see page 2267)

CopyToClipboard (see page 2268)

GetSelTextBuf (see page 2275)

PasteFromClipboard (see page 2276)

SelText (see page 2296)

Displaying and Editing Text in a Memo Control

29.1.11.3.10 TCustomEdit.DefaultHandler Method

Provides message handling for all messages that the control does not fully process itself.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to change the default message handling for the edit control. The Message parameter can be cast to a TMessage type, to obtain the WParam, LParam, and Result of the message. If the Result of the message is non-zero, the message has already been handled. Set the Result field to a non-zero value to prevent further processing of the message by the inherited method.

Note: In Delphi, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method if that ancestor does not specify a handler for the message being handled.

See Also

WindowProc

DefWndProc

TMessage

29.1.11.3.11 TCustomEdit.DestroyWnd Method

Destroys the Windows control that corresponds to the edit control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the edit box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. When this happens, DestroyWnd is called.

Before the window is destroyed, DestroyWnd saves a copy of any properties that are needed to recreate the window after it is destroyed.

See Also

CreateWnd (see page 2270)

29.1.11.3.12 TCustomEdit.DoSetMaxLength Method

Sets the MaxLength (see page 2286) property.

Pascal

```
procedure DoSetMaxLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall DoSetMaxLength(int Value);
```

Description

DoSetMaxLength is the property access function for the MaxLength (see page 2286) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the MaxLength (see page 2286) property.

DoSetMaxLength sends an EM_LIMITTEXT message to the window of the edit control.

See Also

MaxLength (see page 2286)

29.1.11.3.13 TCustomEdit.DoSetTextHint Method

Pascal

```
procedure DoSetTextHint(const Value: string); virtual;
```

C++

```
virtual __fastcall DoSetTextHint(const UnicodeString Value);
```

Description

This is DoSetTextHint, a member of class TCustomEdit.

29.1.11.3.14 TCustomEdit.GetControlsAlignment Method

Indicates how text is aligned within the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the alignment of the control. For controls that do not have an Alignment property, GetControlsAlignment returns taLeftJustify.

Note: GetControlsAlignment returns the alignment of text before it is altered by the BiDiMode property. To check if the alignment should be reversed to reflect the BiDiMode property, call UseRightToLeftAlignment.

See Also

BiDiMode

Text

Caption

UseRightToLeftAlignment

29.1.11.3.15 TCustomEdit.GetSelLength Method

Returns the SelLength (see page 2289) property.

Pascal

```
function GetSelLength: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelLength();
```

Description

GetSelLength is the property access function for the SelLength (see page 2289) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the SelLength (see page 2289) property.

GetSelLength sends an EM_GETSEL message to the window of the edit control and returns the difference between the starting position and the ending position.

See Also

SelLength (see page 2289)

SetSelLength (see page 2280)

29.1.11.3.16 TCustomEdit.GetSelStart Method

Returns the SelStart (see page 2293) property.

Pascal

```
function GetSelStart: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelStart();
```

Description

GetSelStart is the property access method for the SelStart (see page 2293) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the SelStart (see page 2293) property.

GetSelStart sends an EM_GETSEL message to the window of the edit control and returns the starting position.

See Also

SelStart (see page 2293)

SetSelStart (see page 2280)

29.1.11.3.17 TCustomEdit.GetSelText Method

Returns the SelText (see page 2296) property.

Pascal

```
function GetSelText: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetSelText();
```

Description

GetSelText is the property access method for the SelText (see page 2296) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the SelText (see page 2296) property.

See Also

SelText (see page 2296)

29.1.11.3.18 TCustomEdit.GetSelTextBuf Method

Copies the selected text into a buffer and returns the number of characters copied.

Pascal

```
function GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetSelTextBuf(const char * Buffer, int BufSize);
```

Description

Use GetSelTextBuf to copy the selected text into a character buffer. If there is no selection, the buffer receives an empty string. If the selection contains more than (BufSize - 1) characters, only the first (BufSize - 1) characters are copied. GetSelTextBuf returns the number of characters that were actually copied into the buffer.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
Char *Buffer;
```

```
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  static std::auto_ptr<Char> _BufferCleaner(Buffer = new Char[Size]);
  Edit1->GetSelTextBuf(Buffer, Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
  Inc(Size); {Add room for null character}
  GetMem(Buffer, Size); {Creates Buffer dynamic variable}
  Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.SelText;
end;
```

See Also

CopyToClipboard (see page 2268)

GetTextBuf

SelLength (see page 2289)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.11.3.19 TCustomEdit.PasteFromClipboard Method

Pastes the contents of the Clipboard into edit control, replacing the current selection.

Pascal

```
procedure PasteFromClipboard;
```

C++

```
__fastcall PasteFromClipboard();
```

Description

Use PasteFromClipboard to replace the selected text with the contents of the Clipboard, or, if no text is selected, to insert the contents of the Clipboard at the cursor. If the Clipboard is empty, or if it does not contain anything in CF_TEXT format, PasteFromClipboard does nothing.

Note: Calling PasteFromClipboard does the same thing as sending the edit control a WM_PASTE message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
```

```
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

CutToClipboard (see page 2271)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

Displaying and Editing Text in a Memo Control

29.1.11.3.20 TCustomEdit.SelectAll Method

Selects all text in the edit control.

Pascal

```
procedure SelectAll;
```

C++

```
__fastcall SelectAll();
```

Description

Use SelectAll to select all text in the edit control. To select only part of the text, use the SelStart (see page 2293) and SelLength (see page 2289) properties.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
```

```
  button on a form. When the user clicks the button, text is
  cut from the rich edit control and pasted into the edit box.
  Note that only the text is pasted. If the rich edit control
  includes any formatting information that is not pasted into
  the edit control.  If the destination where a rich edit
  control, the formatting information would be copied as well.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
    RichEdit1.CutToClipboard;
    Edit1.Clear;
    Edit1.PasteFromClipboard;
    RichEdit1.SetFocus;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const
    Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

Text

29.1.11.3.21 TCustomEdit.SetAlignment Method

Sets the Alignment (see page 2283) property.

Pascal

```
procedure SetAlignment(Value: TAlignment);
```

C++

```
__fastcall SetAlignment(TAlignment Value);
```

Description

SetAlignment is exposed as a protected method so derived classes can customize their implementation of the Alignment (see page 2283) property.

See Also

Alignment (see page 2283)

29.1.11.3.22 TCustomEdit.SetAutoSize Method

Sets the AutoSize (see page 2284) property.

Pascal

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

C++

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

Description

SetAutoSize is the protected write implementation of the AutoSize (see page 2284) property.

TCustomEdit (see page 2263) overrides the inherited method to update the height of the edit control to reflect the current font when the value of AutoSize (see page 2284) changes to true.

See Also

AutoSize (see page 2284)

29.1.11.3.23 TCustomEdit.SetSelLength Method

Sets the SelLength (see page 2289) property.

Pascal

```
procedure SetSelLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelLength(int Value);
```

Description

SetSelLength is the property access method for the SelLength (see page 2289) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the SelLength (see page 2289) property.

SetSelLength sends an EM_SETSEL message to the window of the edit control, specifying a selection that begins at SelStart (see page 2293) and ends at SelStart (see page 2293) + Value.

See Also

GetSelLength (see page 2274)

SelLength (see page 2289)

SelStart (see page 2293)

29.1.11.3.24 TCustomEdit.SetSelStart Method

Sets the SelStart (see page 2293) property.

Pascal

```
procedure SetSelStart(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelStart(int Value);
```

Description

SetSelStart is the property access method for the SelStart (see page 2293) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2263) can customize the implementation of the SelStart (see page 2293) property.

SetSelLength (see page 2280) sends an EM_SETSEL message to the window of the edit control to position the cursor on the position indicated by Value.

See Also

GetSelStart (see page 2274)

SelStart (see page 2293)

29.1.11.3.25 TCustomEdit.SetSelText Method

Sets the SelText (see page 2296) property.

Pascal

```
procedure SetSelText(const Value: string);
```

C++

```
__fastcall SetSelText(const UnicodeString Value);
```

Description

SetSelText is the property access method for the SelText (see page 2296) property.

See Also

GetSelTextBuf (see page 2275)

SelText (see page 2296)

29.1.11.3.26 TCustomEdit.SetSelTextBuf Method

Replaces the selected text with a null-terminated string.

Pascal

```
procedure SetSelTextBuf(Buffer: PChar);
```

C++

```
__fastcall SetSelTextBuf(const char * Buffer);
```

Description

Use SetSelTextBuf to replace the current selection by the contents of the null-terminated string pointed to by Buffer. If no text is selected, the contents of Buffer are inserted at the cursor. SetSelTextBuf does the same thing as setting the SelText (see page 2296) property, except that it takes a pointer to a char type rather than a string.

C++ Examples:

```
/*
This example inserts the contents of temp_string at the
current location of the cursor.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   const AnsiString Path = "../readme.txt";
   Memo1->Lines->LoadFromFile(Path);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  WideChar *temp_string = L"I am now here";
  Memo1->SetSelTextBuf(temp_string);
}
```

Delphi Examples:

```
{
This example inserts the contents of temp_string at the
current location of the cursor.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  temp_string: array[0..15] of Char;
begin
  temp_string := 'I am now here';
  Memo1.SetSelTextBuf(temp_string);
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'readme.txt';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

GetSelTextBuf (see page 2275)

PasteFromClipboard (see page 2276)

SelText (see page 2296)

SetTextBuf

29.1.11.3.27 TCustomEdit.Undo Method

Backs out all changes in the undo buffer.

Pascal

```
procedure Undo;
```

C++

```
__fastcall Undo();
```

Description

Call Undo to cancel all changes made to the Text (see page 2297) property since the last call to the ClearUndo (see page 2268) method. If ClearUndo (see page 2268) was never called, Undo rolls back all changes.

To determine whether there are any changes in the undo buffer, check the CanUndo (see page 2285) property.

See Also

ClearUndo (see page 2268)

CanUndo (see page 2285)

Text

29.1.11.4 TCustomEdit Properties

The properties of the TCustomEdit class are listed here.

Protected Properties

	Name	Description
	AutoSelect (see page 2284)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2284)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2285)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2286)	Determines the case of the text within the edit control.

	HideSelection (see page 2286)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2286)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2287)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2288)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2288)	Determines where a control looks for its color information.
	PasswordChar (see page 2289)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Properties

	Name	Description
	Alignment (see page 2283)	Determines how the text is aligned within the text edit control.
R	CanUndo (see page 2285)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2287)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2289)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2289)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2293)	Specifies the position of the first selected character in the text.
	SelText (see page 2296)	Specifies the selected portion of the edit control's text.
	Text (see page 2297)	Contains a text string associated with the control.
	TextHint (see page 2299)	A hint or message to be displayed when the Text (see page 2297) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2297)	Determines if the user can tab to a control.

29.1.11.4.1 TCustomEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2279)

Text

Displaying and Editing Fields in an Edit Box

29.1.11.4.2 TCustomEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 2278)

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.11.4.3 TCustomEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
```

```
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 2285)

ClientHeight

ControlStyle

Height

29.1.11.4.4 TCustomEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

29.1.11.4.5 TCustomEdit.CanUndo Property

Indicates whether the edit control contains changes that can be backed out.

Pascal

```
property CanUndo: Boolean;
```

C++

```
__property Boolean CanUndo;
```

Description

Read (see page 3727) CanUndo to determine whether the user has made any changes to the text that can be undone by calling the Undo (see page 2282) method. This is useful when enabling or disabling menu items that correspond to these commands.

See Also

Undo (see page 2282)

ClearUndo (see page 2268)

Text

29.1.11.4.6 TCustomEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

29.1.11.4.7 TCustomEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.11.4.8 TCustomEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

29.1.11.4.9 TCustomEdit.Modified Property

Indicates whether the user edited the text of the edit control.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use Modified to determine whether the user changed the Text (see page 2297) property of the edit control. Modified is only reset to **False** when you assign a value to the Text (see page 2297) property. In particular, it is not reset when the control receives focus.

See Also

Change (see page 2266)

OnChange (see page 2299)

SetTextBuf

Text

29.1.11.4.10 TCustomEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly: Boolean;
```

C++

```
__property Boolean NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

29.1.11.4.11 TCustomEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

29.1.11.4.12 TCustomEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.11.4.13 TCustomEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text (see page 2297) property reflects the actual characters that are typed.

See Also

Text

29.1.11.4.14 TCustomEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2268)

GetSelTextBuf (see page 2275)

29.1.11.4.15 TCustomEdit.SelLength Property

Specifies the number of characters (bytes) that are selected.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

Read (see page 3727) SelLength to determine the length, in bytes, of the selected text. This is the same as the number of

characters, unless you are using a multi-byte character set. Set SelLength to change the selection to consist of the first SelLength bytes starting at SelStart (see page 2293).

Note: Setting SelLength to a value greater than the number of characters from SelStart (see page 2293) to the end of the text results in the selection of all characters from SelStart (see page 2293) to the end of the text. Reading SelLength immediately after setting it to a value greater than the number of available characters returns the number of characters actually selected, not the value that was just set.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection. Notice that the
"Match whole word only" and "Match case" work.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes = TSearchTypes();
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
```

```
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
```

```
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following OnFind event handler searches a memo component
  for the text specified in the FindText property of a find
  dialog component. If found, the first occurrence of the text
  in Memo1 is selected. The code uses the Pos function to
  compare strings, and stores the number of characters to skip
  when determining the selection position in the SkipChars
  variable. Because there is no handling of case, whole word,
  or search direction in this algorithm, it is assumed that
  the Options property of FindDialog1 is set to
  [frHideMatchCase, frHideWholeWord, frHideUpDown].
  }
  procedure TForm1.FindDialog1Find(Sender: TObject);
  var
    I, J, PosReturn, SkipChars: Integer;
  begin
    for I := 0 to Memo1.Lines.Count do
    begin
      PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
      if PosReturn <> 0 then {found!}
      begin
        SkipChars := 0;
        for J := 0 to I - 1 do
          SkipChars := SkipChars + Length(Memo1.Lines[J]);
        SkipChars := SkipChars + (I*2);
        SkipChars := SkipChars + PosReturn - 1;
        Memo1.SetFocus;
        Memo1.SelStart := SkipChars;
        Memo1.SelLength := Length(FindDialog1.FindText);
        Break;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'country.list';
  begin
    Memo1.Lines.LoadFromFile(Path);
  end;
```

See Also

SelStart (see page 2293)

SelText (see page 2296)

29.1.11.4.16 TCustomEdit.SelStart Property

Specifies the position of the first selected character in the text.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read (see page 3727) SelStart to determine the position of the first selected character, where 0 indicates the first character. If there is no selected text, SelStart indicates the position of the cursor. Set SelStart to remove the current selection and position the cursor just before the indicated character.

To select a particular range of the text, first set SelStart to position the cursor, and then set SelLength (see page 2289) to extend the selection.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection. Notice that the
"Match whole word only" and "Match case" work.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes = TSearchTypes();
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
```

```
      mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
      mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
    FoundAt :=
      FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end
    else Beep;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

SelLength (see page 2289)

SelText (see page 2296)

29.1.11.4.17 TCustomEdit.SelText Property

Specifies the selected portion of the edit control's text.

Pascal

```
property SelText: string;
```

C++

```
__property UnicodeString SelText;
```

Description

Read (see page 3727) SelText to determine the value of the selected text. Set SelText to replace the selected text with a new string. If there is no selection, but the edit control has focus, set SelText to insert a new string into the text at the cursor.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
Char *Buffer;

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  static std::auto_ptr<Char> _BufferCleaner(Buffer = new Char[Size]);
  Edit1->GetSelTextBuf(Buffer, Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
```

```
    Inc(Size); {Add room for null character}
    GetMem(Buffer, Size); {Creates Buffer dynamic variable}
    Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
    Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
    FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
  end;

  {
  Note: the same thing could be accomplished more simply as follows:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Edit2.Text := Edit1.SelText;
  end;
```

See Also

ClearSelection (see page 2267)

GetSelTextBuf (see page 2275)

SelLength (see page 2289)

SelStart (see page 2293)

SetSelTextBuf (see page 2281)

Text

29.1.11.4.18 TCustomEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.11.4.19 TCustomEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2368)

29.1.11.4.20 TCustomEdit.TextHint Property

A hint or message to be displayed when the Text (see page 2297) property is empty.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

29.1.11.5 TCustomEdit Events

The events of the TCustomEdit class are listed here.

Protected Events

	Name	Description
	OnChange (see page 2299)	Occurs when the text for the edit control may have changed.

29.1.11.5.1 TCustomEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3789) an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified (see page 2287) property to see if a change actually occurred. The Text (see page 2297) property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2266)

Text

29.1.12 TCustomGroupBox Class

TCustomGroupBox is the base class for all group-box components.

Class Hierarchy

Pascal

```
TCustomGroupBox = class(TCustomControl);
```

C++

```
class TCustomGroupBox : public TCustomControl;
```

File

StdCtrls

Description

Group boxes are visual components that act as containers for sets of related controls, such as radio buttons. Group boxes may also contain other group boxes. TCustomGroupBox is the class from which all group-box components—including TGroupBox (see page 2451), TRadioGroup, and TDBRadioGroup—descend.

See Also

TCustomGroupBox

TDBRadioGroup

TRadioGroup

TGroupBox (see page 2451)

TCustomControl

TWidgetControl

29.1.12.1 TCustomGroupBox Members

The following tables list the members exposed by TCustomGroupBox.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2301)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2301)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2302)	Renders the onscreen image of the group box.

Public Methods

	Name	Description
	Create (see page 2300)	Creates and initializes a TCustomGroupBox (see page 2299) instance.

29.1.12.2 TCustomGroupBox.Create Constructor

Creates and initializes a TCustomGroupBox (see page 2299) instance.

Pascal

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

C++

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

Description

Call Create to instantiate a group box at runtime. Group boxes placed on forms at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the group box. It becomes the value of the Owner property.

See Also

Owner

29.1.12.3 TCustomGroupBox Methods

The methods of the TCustomGroupBox class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2301)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2301)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2302)	Renders the onscreen image of the group box.

29.1.12.3.1 TCustomGroupBox.AdjustClientRect Method

Adjusts the ClientRect property for idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the group box needs accurate information on where to place child controls within the client area. It indents the client rectangle slightly so that child controls are not placed on the frame (or caption) of the group box.

See Also

ClientRect

Caption

29.1.12.3.2 TCustomGroupBox.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a group box creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

Specify the parameter record as the value of the Params parameter. The TCreateParams type is a data structure holding information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TCustomGroupBox (see page 2299) adjusts the window style by turning off the CS_HREDRAW and CS_VREDRAW flags.

See Also

CreateWnd

CreateWindowHandle

29.1.12.3.3 TCustomGroupBox.Paint Method

Renders the onscreen image of the group box.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

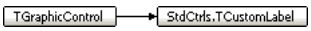
Description

Paint draws a border around the group box and fills in the caption.

29.1.13 TCustomLabel Class

TCustomLabel is the base class for non-windowed controls that display text on a form.

Class Hierarchy



Pascal

```
TCustomLabel = class(TGraphicControl);
```

C++

```
class TCustomLabel : public TGraphicControl;
```

File

StdCtrls

Description

Use TCustomLabel as a base class when defining objects that display text that users can't edit. TCustomLabel introduces several new properties and methods to control the appearance of the text.

TCustomLabel is not a descendant of TWinControl, so it does not have its own window and can't receive direct input from the keyboard. To define an object that responds to direct keyboard input in addition to displaying text, use TCustomStaticText (see page 2375) as a base class.

Do not create instances of TCustomLabel. To put a label on a form, use a TLabel (see page 2522) object.

See Also

TCustomStaticText (see page 2375)

TLabel (see page 2522)

29.1.13.1 TCustomLabel Members

The following tables list the members exposed by TCustomLabel.

Protected Methods

	Name	Description
	AdjustBounds (see page 2304)	Resizes the label when the text changes.
	DoDrawText (see page 2304)	Renders the text in the client area.
	GetLabelText (see page 2305)	Returns the value of the Caption (see page 2310) property.
	Loaded (see page 2305)	Adjusts the size of the label when it is first loaded. .
	Notification (see page 2305)	Responds to notifications when objects are about to be created or destroyed.
	Paint (see page 2306)	Draws the image of the label.
	SetAutoSize (see page 2306)	Sets the AutoSize (see page 2307) property.

Protected Properties

	Name	Description
	Alignment (see page 2307)	Controls the horizontal placement of the text within the label.
	AutoSize (see page 2307)	Determines whether the size of the label automatically resizes to accommodate the text.
	EllipsisPosition (see page 2313)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 2313)	Designates a windowed control associated with the label.
	Layout (see page 2314)	Specifies the vertical placement of the text within the label.
	ShowAccelChar (see page 2314)	Determines how an ampersand in the label text is displayed.
	Transparent (see page 2315)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WordWrap (see page 2316)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Methods

	Name	Description
	Create (see page 2303)	Creates and initializes an instance of TCustomLabel (see page 2302).

Public Properties

	Name	Description
	Canvas (see page 2308)	Provides the drawing surface used by the graphic control.
	Caption (see page 2310)	Specifies a text string that identifies the control to the user.
	GlowSize (see page 2314)	This is GlowSize, a member of class TCustomLabel.

29.1.13.2 TCustomLabel.Create Constructor

Creates and initializes an instance of TCustomLabel (see page 2302).

Pascal

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

C++

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

Description

Call Create to create an instance of TCustomLabel (see page 2302) at runtime. Labels placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the label.

See Also

Owner

29.1.13.3 TCustomLabel Methods

The methods of the TCustomLabel class are listed here.

Protected Methods

	Name	Description
	AdjustBounds (see page 2304)	Resizes the label when the text changes.
	DoDrawText (see page 2304)	Renders the text in the client area.
V	GetLabelText (see page 2305)	Returns the value of the Caption (see page 2310) property.
V	Loaded (see page 2305)	Adjusts the size of the label when it is first loaded. .
V	Notification (see page 2305)	Responds to notifications when objects are about to be created or destroyed.
V	Paint (see page 2306)	Draws the image of the label.
V	SetAutoSize (see page 2306)	Sets the AutoSize (see page 2307) property.

29.1.13.3.1 TCustomLabel.AdjustBounds Method

Resizes the label when the text changes.

Pascal

```
procedure AdjustBounds; dynamic;
```

C++

```
__fastcall AdjustBounds();
```

Description

When the text in the label or the font changes, AdjustBounds is implicitly called. It redraws the text in the client area, and then resizes the label if the AutoSize (see page 2307) property is true.

When WordWrap (see page 2316) is true, the width of the label is fixed. If AutoSize (see page 2307) is also true, changes to the text cause the label to change in height. When AutoSize (see page 2307) is true and WordWrap (see page 2316) is false, the height of the label is determined by the font, and changes to the text cause the label to change in width.

Override AdjustBounds to make additional adjustments when the text changes.

See Also

DoDrawText (see page 2304)

AutoSize (see page 2307)

29.1.13.3.2 TCustomLabel.DoDrawText Method

Renders the text in the client area.

Pascal

```
procedure DoDrawText(var Rect: TRect; Flags: Longint); dynamic;
```

C++

```
__fastcall DoDrawText(TRect Rect, Longint Flags);
```

Description

When the label needs to draw itself, DoDrawText is implicitly called. DoDrawText uses the Windows API function, DrawText, to render the caption in the client area.

The first parameter, Rect, specifies the boundaries of the text within the client area. The second parameter, Flags, specifies how the text is aligned within those boundaries, how tab characters are handled, etc. See the Microsoft Windows API DrawText function for all the possible values of Flags.

Override DoDrawText to render text differently.

See Also

Caption

AdjustBounds (see page 2304)

29.1.13.3.3 TCustomLabel.GetLabelText Method

Returns the value of the Caption (see page 2310) property.

Pascal

```
function GetLabelText: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetLabelText();
```

Description

Call GetLabelText to obtain the string that appears as the text of the label.

See Also

Caption

29.1.13.3.4 TCustomLabel.Loaded Method

Adjusts the size of the label when it is first loaded. .

Pascal

```
procedure Loaded; override;
```

C++

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

Description

The protected Loaded method is called immediately after a saved version of the form that contains the label is loaded into memory. TCustomLabel (see page 2302) uses this method to adjust the size of the label to the text it contains if the AutoSize (see page 2307) property is true.

See Also

AutoSize (see page 2307)

29.1.13.3.5 TCustomLabel.Notification Method

Responds to notifications when objects are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the windowed control specified by the FocusControl (see page 2313) property is about to be destroyed. If the windowed control is going away, Notification sets the FocusControl (see page 2313) property to nil (Delphi) or NULL (C++).

See Also

FocusControl (see page 2313)

29.1.13.3.6 TCustomLabel.Paint Method

Draws the image of the label.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically when the label receives a WM_PAINT message from its Parent. Paint writes the text returned by the GetLabelText (see page 2305) method.

See Also

GetLabelText (see page 2305)

29.1.13.3.7 TCustomLabel.SetAutoSize Method

Sets the AutoSize (see page 2307) property.

Pascal

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

C++

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

Description

SetAutoSize is the protected access method for the AutoSize (see page 2307) property. It is exposed as a protected method so descendants can override it to change the implementation of the AutoSize (see page 2307) property.

See Also

AutoSize (see page 2307)

29.1.13.4 TCustomLabel Properties

The properties of the TCustomLabel class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2307)	Controls the horizontal placement of the text within the label.
	AutoSize (see page 2307)	Determines whether the size of the label automatically resizes to accommodate the text.
	EllipsisPosition (see page 2313)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 2313)	Designates a windowed control associated with the label.
	Layout (see page 2314)	Specifies the vertical placement of the text within the label.
	ShowAccelChar (see page 2314)	Determines how an ampersand in the label text is displayed.
	Transparent (see page 2315)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WordWrap (see page 2316)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Properties

	Name	Description
	Canvas (see page 2308)	Provides the drawing surface used by the graphic control.
	Caption (see page 2310)	Specifies a text string that identifies the control to the user.
	GlowSize (see page 2314)	This is GlowSize, a member of class TCustomLabel.

29.1.13.4.1 TCustomLabel.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap (see page 2316) property is true and the label includes more than one line of text.

See Also

ClientRect

GetLabelText (see page 2305)

Layout (see page 2314)

WordWrap (see page 2316)

29.1.13.4.2 TCustomLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text.

When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap (see page 2316) is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap (see page 2316) is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
```

See Also

BoundsRect

GetLabelText (see page 2305)

Font

WordWrap (see page 2316)

Layout (see page 2314)

29.1.13.4.3 TCustomLabel.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent

control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = dynamic_cast<TLabel *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

29.1.13.4.4 TCustomLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}
```

```
TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
```

```
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

29.1.13.4.5 TCustomLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition (see page 2908) constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 2316)

TypeName

29.1.13.4.6 TCustomLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the

label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar (see page 2314) property to true.

See Also

ShowAccelChar (see page 2314)

GetLabelText (see page 2305)

29.1.13.4.7 TCustomLabel.GlowSize Property

Pascal

```
property GlowSize: Integer;
```

C++

```
__property int GlowSize;
```

Description

This is GlowSize, a member of class TCustomLabel.

29.1.13.4.8 TCustomLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment (see page 2307) property.

See Also

Alignment (see page 2307)

ClientRect

29.1.13.4.9 TCustomLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl (see page 2313) property is set, the windowed control specified by the FocusControl (see page 2313) property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl (see page 2313) property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 2313)

GetLabelText (see page 2305)

29.1.13.4.10 TCustomLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color

ParentColor

29.1.13.4.11 TCustomLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth

GetLabelText (see page 2305)

29.1.14 TCustomListBox Class

TCustomListBox is the base class for listbox objects that are wrappers for list box controls.

Class Hierarchy

Pascal

```
TCustomListBox = class(TCustomMultiSelectListControl);
```

C++

```
class TCustomListBox : public TCustomMultiSelectListControl;
```

File

StdCtrls

Description

TCustomListBox encapsulates behavior common to all list box controls. List box objects are used to display a scrollable list from which users can select one or more items. Use a list box control to limit the user to selecting from a list of items. Items (see page 2346) can be added at design time or at runtime.

As a wrapper for a list box TCustomListBox introduces methods for:

Displaying, sorting, selecting, manipulating, and querying items in the list.

Creating owner-draw style list boxes that can display graphical images such as icons and bitmaps.

Creating multiple-column and multiple-selection lists.

By default, TCustomListBox is a standard list box that displays items as text strings. These items are contained in a TStrings object, and the methods of the TStrings object are used to add, insert or delete a string. The following code demonstrates adding a new string to the list box:

MyListBox.Items.Add('New (see page 3712) item'); //Delphi

MyListBox->Items (see page 2346)->Add("New (see page 3712) item"); // C++

Descendants of TCustomListBox inherit the basic functionality of TCustomListBox. Some descendants also have specialized behavior and can be linked to file, directory, drive, or database information. Use these list boxes to easily access and list this type of information.

A combo box is a list box with an edit control and an optional drop down style for the list. Use a combo box to allow users to type their selection into an edit box or to save space on a form by having the list drop down.

For lists of items that require a matrix or expandable tree-like structure, use list views or tree views. These controls allow the grouping of items under column headings or in expandable and contractible rows.

See Also

TStrings

29.1.14.1 TCustomListBox Members

The following tables list the members exposed by TCustomListBox.

Protected Events

	Name	Description
	OnData (see page 2356)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2356)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2357)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDrawItem (see page 2357)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnMeasureItem (see page 2359)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Protected Fields

	Name	Description
	FMoving (see page 2319)	This is FMoving, a member of class TCustomListBox.

Protected Methods

	Name	Description
V	CreateParams (see page 2322)	Initializes the window-creation parameter record passed in the Params parameter.
V	CreateWnd (see page 2323)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2324)	Deletes a string from the list box.
V	DestroyWnd (see page 2324)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2325)	Returns the index of a string in the list box.
	DoGetData (see page 2325)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2326)	Returns the object associated with an item in a virtual list box.
V	DragCanceled (see page 2326)	Sends a mouse message canceling the drag-and-drop operation.
V	DrawItem (see page 2326)	Generates an OnDrawItem (see page 2357) event.
	GetItemData (see page 2327)	Returns a 32-bit value associated with the specified item in a list box.
V	GetItemIndex (see page 2328)	Returns the value of the ItemIndex property.
V	GetSelCount (see page 2328)	Returns the number of selected items.
	GetSelected (see page 2328)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2328)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2329)	Associates an integer with a specified item in the list box.
V	KeyPress (see page 2332)	Responds when the user presses a key.
V	LoadRecreateItems (see page 2333)	This is LoadRecreateItems, a member of class TCustomListBox.
V	MeasureItem (see page 2333)	Generates an OnMeasureItem (see page 2359) event.

	ResetContent (see page 2334)	Removes all of the items from a list box.
V	SaveRecreateItems (see page 2334)	This is SaveRecreateItems, a member of class TCustomListBox.
	SetItemData (see page 2334)	Sets the 32-bit value associated with the specified item in a list box.
V	SetItemIndex (see page 2335)	Gives focus to the item with a specified index.
V	SetMultiSelect (see page 2335)	Sets the MultiSelect property.
V	WndProc (see page 2335)	Handles Windows messages.

Protected Properties

	Name	Description
	BorderStyle (see page 2341)	Determines whether the list box has a border.
	Columns (see page 2344)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	ExtendedSelect (see page 2345)	Determines if the user can select a sequential range of items in the list box.
	IntegralHeight (see page 2345)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2346)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 2349)	Determines where a control looks for its color information.
	Sorted (see page 2352)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2353)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 2354)	Specifies the size of the tabs in the list box.

Public Methods

	Name	Description
V	AddItem (see page 2320)	Adds an item to the list box
V	Clear (see page 2321)	Deletes all items from the list box.
V	ClearSelection (see page 2321)	Deselects all selected items in the list box, if possible.
V	CopySelection (see page 2322)	Copies the selected items to another list-type control
V	Create (see page 2318)	Instantiates a list box control.
V	DeleteSelected (see page 2324)	Deletes the selected items from the list box
V	Destroy (see page 2319)	Destroys the instance of a list box control.
V	GetCount (see page 2327)	Returns the value of the Count (see page 2344) property.
	ItemAtPos (see page 2329)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2331)	Returns the rectangle that surrounds the item specified in the Item parameter.
V	SelectAll (see page 2334)	Selects the entire block of text in the list box.

Public Properties

	Name	Description
	AutoComplete (see page 2340)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2341)	Specifies the delay between a key press and an attempt to autocomplete the field.
R	Canvas (see page 2341)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2357) event of an owner-draw list box.
	Count (see page 2344)	Indicates the number of items in the list box.
	Items (see page 2346)	Contains the strings that appear in the list box.
	ScrollWidth (see page 2349)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	Selected (see page 2350)	Indicates whether a particular item is selected.
	TopIndex (see page 2354)	Specifies the index number of the item that appears at the top of the list box.

Published Properties

	Name	Description
	TabStop (see page 2354)	Determines if the user can tab to a control.

29.1.14.2 TCustomListBox.Create Constructor

Instantiates a list box control.

Pascal

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

C++

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

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner

29.1.14.3 TCustomListBox.Destroy Destructor

Destroys the instance of a list box control.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomListBox();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free checks that the list box reference is not nil before it calls Destroy.

Destroy frees the memory associated with the objects for the Canvas (see page 2341) and Items (see page 2346) properties. Then it calls the inherited destructor to destroy the list box object.

See Also

System::TObject::Free (see page 3382)

29.1.14.4 TCustomListBox Fields

The fields of the TCustomListBox class are listed here.

Protected Fields

	Name	Description
	FMoving (see page 2319)	This is FMoving, a member of class TCustomListBox.

29.1.14.4.1 TCustomListBox.FMoving Field

Pascal

```
FMoving: Boolean;
```

C++

```
Boolean FMoving;
```

Description

This is FMoving, a member of class TCustomListBox.

29.1.14.5 TCustomListBox Methods

The methods of the TCustomListBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2322)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 2323)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2324)	Deletes a string from the list box.
	DestroyWnd (see page 2324)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2325)	Returns the index of a string in the list box.
	DoGetData (see page 2325)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2326)	Returns the object associated with an item in a virtual list box.
	DragCanceled (see page 2326)	Sends a mouse message canceling the drag-and-drop operation.
	DrawItem (see page 2326)	Generates an OnDrawItem (see page 2357) event.
	GetItemData (see page 2327)	Returns a 32-bit value associated with the specified item in a list box.
	GetItemIndex (see page 2328)	Returns the value of the ItemIndex property.
	GetSelCount (see page 2328)	Returns the number of selected items.
	GetSelected (see page 2328)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2328)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2329)	Associates an integer with a specified item in the list box.
	KeyPress (see page 2332)	Responds when the user presses a key.
	LoadRecreateItems (see page 2333)	This is LoadRecreateItems, a member of class TCustomListBox.
	MeasureItem (see page 2333)	Generates an OnMeasureItem (see page 2359) event.
	ResetContent (see page 2334)	Removes all of the items from a list box.
	SaveRecreateItems (see page 2334)	This is SaveRecreateItems, a member of class TCustomListBox.
	SetItemData (see page 2334)	Sets the 32-bit value associated with the specified item in a list box.
	SetItemIndex (see page 2335)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 2335)	Sets the MultiSelect property.
	WndProc (see page 2335)	Handles Windows messages.

Public Methods

	Name	Description
	AddItem (see page 2320)	Adds an item to the list box
	Clear (see page 2321)	Deletes all items from the list box.
	ClearSelection (see page 2321)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 2322)	Copies the selected items to another list-type control
	DeleteSelected (see page 2324)	Deletes the selected items from the list box
	GetCount (see page 2327)	Returns the value of the Count (see page 2344) property.
	ItemAtPos (see page 2329)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2331)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 2334)	Selects the entire block of text in the list box.

29.1.14.5.1 TCustomListBox.AddItem Method

Adds an item to the list box

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(UnicodeString Item, TObject * AObject);
```

Description

Call AddItem to add a new item to the end of the list.

Item is the text of the item to add.

AObject is the object associated with the new item.

See Also

AddObject

Items (see page 2346)

29.1.14.5.2 TCustomListBox.Clear Method

Deletes all items from the list box.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to delete all of the items in the list box at once. Although the items in a standard list box are of type TStrings, use the Clear method of the list box rather than calling the Clear method of the Items (see page 2346) property. This allows descendants of TCustomListBox (see page 2316) to perform any other necessary clean-up in the Clear method, in addition to deleting the items from the Items (see page 2346) property. Thus, applications should use

```
ListBox1.Clear;
ListBox1->Clear();
```

rather than

```
ListBox1.Items.Clear;
ListBox1->Items->Clear();
```

See Also

Clear

Items (see page 2346)

29.1.14.5.3 TCustomListBox.ClearSelection Method

Deselects all selected items in the list box, if possible.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

Use ClearSelection to deselect all the currently selected items in the list box. When the list box receives focus, after ClearSelection is called, the last item that was selected is the item that receives the current focus in the list box. Although this

item receives focus, the item itself is not selected.

See Also

DeleteSelected (see page 2324)

extendedselect (see page 2345)

MultiSelect

SelCount

Selected (see page 2350)

ItemIndex

29.1.14.5.4 TCustomListBox.CopySelection Method

Copies the selected items to another list-type control

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

Call CopySelection to copy all the items selected in the list box to a combo box, list view, or another list box. If the destination control is not sorted, the selected items are added to the end of its list.

Destination is the list box, combo box, or list view to which the selected items are copied.

See Also

MoveSelection

extendedselect (see page 2345)

MultiSelect

SelCount

Selected (see page 2350)

ItemIndex

29.1.14.5.5 TCustomListBox.CreateParams Method

Initializes the window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is used to pass information, in its Params parameter, to CreateWindowHandle when the list box control is created.

TCustomListBox (see page 2316) overrides CreateParams to customize the representation of the window. It uses information from the list box Columns (see page 2344), ExtendedSelect (see page 2345), IntegralHeight (see page 2345),

MultiSelect, Sorted (see page 2352) and Style (see page 2353) properties and sets WS_HSCROLL, WS_VSCROLL, LBS_HASSTRINGS and LBS_NOTIFY Windows styles for creation of the list box.

When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

See Also

Columns (see page 2344)

CreateWindowHandle

CreateWnd

extendedselect (see page 2345)

IntegralHeight (see page 2345)

MultiSelect

Sorted (see page 2352)

Style (see page 2353)

29.1.14.5.6 TCustomListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams (see page 2322) to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 2344)

CreateHandle

CreateParams (see page 2322)

CreateWindowHandle

CreateWnd

ItemIndex

Items (see page 2346)

TabWidth (see page 2354)

TopIndex (see page 2354)

29.1.14.5.7 TCustomListBox.DeleteSelected Method

Deletes the selected items from the list box

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

Call DeleteSelected to delete the selected items from the list box.

See Also

ClearSelection

MoveSelection

CopySelection (see page 2322)

extendedselect (see page 2345)

MultiSelect

SelCount

Selected (see page 2350)

ItemIndex

29.1.14.5.8 TCustomListBox.DeleteString Method

Deletes a string from the list box.

Pascal

```
procedure DeleteString(Index: Integer); dynamic;
```

C++

```
__fastcall DeleteString(int Index);
```

Description

DeleteString deletes the string, specified by the Index parameter, from the list box control.

29.1.14.5.9 TCustomListBox.DestroyWnd Method

Saves the list box items in a string list before freeing the underlying Windows control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd saves the list box text string items in the Items (see page 2346) property to internal storage in a TStringList object that it creates. It then calls the inherited DestroyWnd method to destroy the windowed control's window handle and free

associated device contexts.

See Also

DestroyWnd

Items (see page 2346)

TStringList

29.1.14.5.10 TCustomListBox.DoFindData Method

Returns the index of a string in the list box.

Pascal

```
function DoFindData(const Data: String): Integer;
```

C++

```
__fastcall int DoFindData(const UnicodeString Data);
```

Description

This protected method is called automatically when the list box needs to indicate the index for an item.

If the list box is virtual (that is, if the value of the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw), DoFindData generates an OnDataFind (see page 2356) event so that the index can be provided dynamically. If the list box is not virtual (any other Style (see page 2353)), DoFindData locates the specified string from the Items (see page 2346) property and returns its index.

Data is the text of the item to locate.

See Also

OnDataFind (see page 2356)

Style (see page 2353)

Items (see page 2346)

29.1.14.5.11 TCustomListBox.DoGetData Method

Returns a string in the list box given its index.

Pascal

```
function DoGetData(const Index: Integer): String;
```

C++

```
__fastcall UnicodeString DoGetData(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the text of an item. DoGetData is never called unless the list box is virtual (that is, unless the value of the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw).

DoGetData generates an OnData (see page 2356) event and returns the string that is supplied dynamically by the event handler.

See Also

OnData (see page 2356)

Style (see page 2353)

Items (see page 2346)

29.1.14.5.12 TCustomListBox.DoGetDataObject Method

Returns the object associated with an item in a virtual list box.

Pascal

```
function DoGetDataObject(const Index: Integer): TObject;
```

C++

```
__fastcall TObject * DoGetDataObject(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the object associated with an item. DoGetDataObject is never called unless the list box is virtual (that is, unless the value of the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw).

DoGetDataObject generates an OnDataObject (see page 2357) event and returns the object that is supplied dynamically by the event handler.

See Also

OnDataObject (see page 2357)

Style (see page 2353)

Items (see page 2346)

29.1.14.5.13 TCustomListBox.DragCanceled Method

Sends a mouse message canceling the drag-and-drop operation.

Pascal

```
procedure DragCanceled; override;
```

C++

```
virtual __fastcall DragCanceled();
```

Description

DragCanceled is called in response to the cancellation of a drag-and-drop operation that had started. Users can cancel the operation by dropping the item over a component that does not accept the item, or by pressing Esc while dragging.

The OnEndDrag event occurs after the call to DragCanceled.

See Also

OnEndDrag

29.1.14.5.14 TCustomListBox.DrawItem Method

Generates an OnDrawItem (see page 2357) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem (see page 2357) event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

Style (see page 2353)

29.1.14.5.15 TCustomListBox.GetCount Method

Returns the value of the Count (see page 2344) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 2344) property.

See Also

Count (see page 2344)

29.1.14.5.16 TCustomListBox.GetItemData Method

Returns a 32-bit value associated with the specified item in a list box.

Pascal

```
function GetItemData(Index: Integer): TCustomLongData; dynamic;
```

C++

```
__fastcall TCustomLongData GetItemData(int Index);
```

Description

GetItemData retrieves the data associated with a particular item in the list box. This value can be displayed as that list box item, or can be a value (such as a pointer) associated with the list box item.

See Also

SetItemData (see page 2334)

29.1.14.5.17 TCustomListBox.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the protected read implementation of the ItemIndex property. It returns the index of the first selected item in the list box.

The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

29.1.14.5.18 TCustomListBox.GetSelCount Method

Returns the number of selected items.

Pascal

```
function GetSelCount: Integer; override;
```

C++

```
virtual __fastcall int GetSelCount();
```

Description

GetSelCount is the protected read implementation of the SelCount property.

See Also

SelCount

29.1.14.5.19 TCustomListBox.GetSelected Method

Indicates whether a particular item is selected.

Pascal

```
function GetSelected(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean GetSelected(int Index);
```

Description

GetSelected is the protected read implementation of the Selected (see page 2350) property. It returns true if the item specified by Index is selected, false if it is not.

29.1.14.5.20 TCustomListBox.InternalGetItemData Method

Retrieves the data associated with an item in the list box.

Pascal

```
function InternalGetItemData(Index: Integer): TCustomLongData; dynamic;
```

C++

```
__fastcall TCustomLongData InternalGetItemData(int Index);
```

Description

InternalGetItemData is the low-level protected method for fetching the data associated with an item in the list box. It retrieves a reference pointer to the object associated with the specified item, as an integer.

Index is the index of the item whose data is to be fetched.

See Also

GetItemData (see page 2327)

29.1.14.5.21 TCustomListBox.InternalSetItemData Method

Associates an integer with a specified item in the list box.

Pascal

```
procedure InternalSetItemData(Index: Integer; AData: TCustomLongData); dynamic;
```

C++

```
__fastcall InternalSetItemData(int Index, TCustomLongData AData);
```

Description

InternalSetItemData is the low-level protected method for associating data with an item in the list box. It associates the item that has the index specified by Index with the integer specified by AData.

See Also

SetItemData (see page 2334)

29.1.14.5.22 TCustomListBox.ItemAtPos Method

Returns the index of the list box item indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; Existing: Boolean): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean Existing);
```

Description

Use ItemAtPos to detect if an item exists at a particular point in the control.

The Pos (see page 3720) parameter is the point in the control in window coordinates. If Pos (see page 3720) is beyond the last item in the list box, the value of the Existing variable determines the returned value. If Existing is set to true, ItemAtPos returns -1, indicating that no item exists at that point. If Existing is set to false, ItemAtPos returns the index of the last item in the list box plus one.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
```

```
  the OnMouseUp event handler for the list box.  The following
  code fills a list box with the names of all components on the
  form when the user clicks the button. References to the
  components themselves are inserted along with the names. The
  components are all inserted at the front of the list, so that
  the last component added to the form is the first component
  in the list.  When the user right-clicks the name of an
  object in the list, the component's coordinates are displayed
  on the status bar. Note that because we are using the right-
  click, the item need not be selected.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    for (int i = 0; i < ComponentCount; i++)
      ListBox1->Items->InsertObject(0,
                                    Components[i]->Name,
                                    dynamic_cast<TObject *>(Components[i]));
  }

  void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y)
  {
    if (Button == mbRight)
    {
      TClass ClassRef;
      int Index = ListBox1->ItemAtPos(Point(X,Y), true);
      // only components that are controls have a position
      // make sure the component is a control
      for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
           ClassRef != NULL;
           ClassRef = ClassRef->ClassParent())
        if (String(ClassRef->ClassName()) == "TControl")
        {
          TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
          StatusBar1->SimpleText =
            TheObject->Name + " is at (" +
            IntToStr(TheObject->Left) + ", " +
            IntToStr(TheObject->Top) + ")";
          break;
        }
      if (ClassRef == NULL) // if it wasn't a control
        MessageBeep(0);
    }
  }
```

Delphi Examples:

```
  {
  For the following example, add a button, a status bar, and a
  list box to the form. Set the SimplePanel property of the
  status bar to true, using the object inspector. Also, populate
  the OnMouseUp event handler for the list box.  The following
  code fills a list box with the names of all components on the
  form when the user clicks the button. References to the
  components themselves are inserted along with the names. The
  components are all inserted at the front of the list, so that
  the last component added to the form is the first component
  in the list.  When the user right-clicks the name of an
  object in the list, the component's coordinates are displayed
  on the status bar. Note that because we are using the right-
  click, the item need not be selected.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    for I := 0 to Form1.ComponentCount-1 do
```

```
      ListBox1.Items.InsertObject(
        0, Form1.Components[I].Name,
        Form1.Components[I] as TObject);
  end;

  procedure TForm1.ListBox1MouseUp(
    Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X,Y: integer);
  var
    APoint: TPoint;
    Index: integer;
    TheObject: TControl;
  begin
    if Button = mbRight then
    begin
    APoint.X := X;
    APoint.Y := Y;
    Index := ListBox1.ItemAtPos(APoint, True);
    if (ListBox1.Items.Objects[Index] is TControl) then
    begin
      TheObject :=
        (ListBox1.Items.Objects[Index] as TControl);
      StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                               IntToStr(TheObject.Left) + ', ' +
                               IntToStr(TheObject.Top) + ') ';
    end
    else
      Beep;
    end;
  end;
```

See Also

ItemIndex

Items (see page 2346)

29.1.14.5.23 TCustomListBox.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to get the coordinates of a particular item in the list box. For example, ItemRect is used internally by the ItemAtPos (see page 2329) method.

The Item parameter is the ItemIndex of the item whose position is queried.

C++ Examples:

```
/*
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Hello");
```

```
  ListBox1->Items->Add("New");
  ListBox1->Items->Add("World");
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  TRect ItemBounds;
  if (Button == mbLeft)
  {
    ItemBounds = ListBox1->ItemRect(ListBox1->ItemIndex);
    if (Y > ItemBounds.Top &&
        Y < ItemBounds.Bottom &&
        X > ItemBounds.Left &&
        X < (ItemBounds.Left + ItemBounds.Right)/2)
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with ListBox1 do
  begin
    Items.Add('Hello');
    Items.Add('New');
    Items.Add('World');
  end;
end;

procedure TForm1.ListBox1MouseUp (Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
var
  ListBoxItem: TRect;
begin
  if Button = mbLeft then
  begin
    ListBoxItem := ListBox1.ItemRect(ListBox1.ItemIndex);
    if (Y > ListBoxItem.Top) and
       (Y < ListBoxItem.Bottom) and
       (X > ListBoxItem.Left) and
       (X < (ListBoxItem.Left + ListBoxItem.Right) div 2) then
      Beep;
  end;
 end;
```

See Also

ItemAtPos (see page 2329)

ItemIndex

29.1.14.5.24 TCustomListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete (see page 2340) property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress

29.1.14.5.25 TCustomListBox.LoadRecreateItems Method

Pascal

```
procedure LoadRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall LoadRecreateItems(TStrings RecreateItems);
```

Description

This is LoadRecreateItems, a member of class TCustomListBox.

29.1.14.5.26 TCustomListBox.MeasureItem Method

Generates an OnMeasureItem (see page 2359) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is called whenever a visual aspect of a variable owner-draw list box changes and the value of the Style (see page 2353) property is csOwnerDrawVariable.

Before drawing its items, a variable owner-draw list box calls MeasureItem once for each visible item, passing the index of the item to measure, and the default height of the item. Since Height is passed by reference (a var parameter), MeasureItem can increase or reduce the height of each item as needed.

By default, the MeasureItem method does nothing except call any event handler attached to the OnMeasureItem (see page 2359) event. Override MeasureItem to change or add functionality to the default behavior.

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

ItemIndex

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

Style (see page 2353)

29.1.14.5.27 TCustomListBox.ResetContent Method

Removes all of the items from a list box.

Pascal

```
procedure ResetContent; dynamic;
```

C++

```
__fastcall ResetContent();
```

Description

ResetContent is used to remove the items from the list box. ResetContent is used, for example, by the Clear (see page 2321) method.

See Also

Clear (see page 2321)

29.1.14.5.28 TCustomListBox.SaveRecreateItems Method

Pascal

```
procedure SaveRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall SaveRecreateItems(TStrings RecreateItems);
```

Description

This is SaveRecreateItems, a member of class TCustomListBox.

29.1.14.5.29 TCustomListBox.SelectAll Method

Selects the entire block of text in the list box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the list box. To select only part of the text, use the SelStart and SelLength properties.

29.1.14.5.30 TCustomListBox.SetItemData Method

Sets the 32-bit value associated with the specified item in a list box.

Pascal

```
procedure SetItemData(Index: Integer; AData: TCustomLongData); dynamic;
```

C++

```
__fastcall SetItemData(int Index, TCustomLongData AData);
```

Description

Use SetItemData to set the data associated with a particular item in a list box. This value can be displayed as that list box item, or it can be a value (such as a pointer) represented by the list box item.

See Also

GetItemData (see page 2327)

29.1.14.5.31 TCustomListBox.SetItemIndex Method

Gives focus to the item with a specified index.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation for the ItemIndex property. Value is the index of the item that should receive focus.

See Also

ItemIndex

29.1.14.5.32 TCustomListBox.SetMultiSelect Method

Sets the MultiSelect property.

Pascal

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

C++

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

Description

SetMultiSelect is the protected write implementation for the MultiSelect property. Value is true if the list box should allow more than one item to be selected, false if only one item can be selected at a time.

See Also

MultiSelect

29.1.14.5.33 TCustomListBox.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the inherited WndProc to provide specialized responses to messages that allows the list box to handle auto

drag mode itself. Then it calls the inherited WndProc to handle all other messages.

When overriding the WndProc method, be sure to call the inherited WndProc at the end to dispatch any other messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
```

```
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
```

```
  a TListBox descendant to respond to a user-defined message
  called WM_STYLEMESSAGE. The subclassed window procedure can
  be turned on or off by pressing a radio button.
  }

  type
    TMyListBoxDescendant = class(TlistBox)
      procedure SubClassWndProc(var Message: TMessage);
      procedure ToggleSubClass(On: Boolean);
      procedure OnDrawItemProc(
        Control: TWinControl;
        Index: Integer;
        Rect:TRect;
        State: TOwnerDrawState);
    end;
    TForm1 = class(TForm)
      SubClassRadioGroup1: TRadioGroup;
      Button1: TButton;
      ImageList1: TImageList;
      Button2: TButton;
      procedure SubClassRadioGroup1Click(Sender: TObject);
      procedure FormCreate(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
      procedure Button1Click(Sender: TObject);
      procedure Button2Click(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;
    MyListBoxDescendant1: TMyListBoxDescendant;
    bitmap0: TBitmap;

  implementation

  {$R *.dfm}

  const WM_STYLEMESSAGE = WM_USER + 2000;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbOwnerDrawFixed),
      0);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbStandard),
      0);
  end;

  procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
  begin
    if (Message.Msg = WM_STYLEMESSAGE) then
      Style:= TListBoxStyle(Message.WParam)
    else
      WndProc(Message);
```

```
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

WndProc

29.1.14.6 TCustomListBox Properties

The properties of the TCustomListBox class are listed here.

Protected Properties

	Name	Description
	BorderStyle (see page 2341)	Determines whether the list box has a border.
	Columns (see page 2344)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	ExtendedSelect (see page 2345)	Determines if the user can select a sequential range of items in the list box.
	IntegralHeight (see page 2345)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2346)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 2349)	Determines where a control looks for its color information.
	Sorted (see page 2352)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2353)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 2354)	Specifies the size of the tabs in the list box.

Public Properties

	Name	Description
	AutoComplete (see page 2340)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2341)	Specifies the delay between a key press and an attempt to autocomplete the field.
R	Canvas (see page 2341)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2357) event of an owner-draw list box.
	Count (see page 2344)	Indicates the number of items in the list box.
	Items (see page 2346)	Contains the strings that appear in the list box.
	ScrollWidth (see page 2349)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	Selected (see page 2350)	Indicates whether a particular item is selected.
	TopIndex (see page 2354)	Specifies the index number of the item that appears at the top of the list box.

Published Properties

	Name	Description
	TabStop (see page 2354)	Determines if the user can tab to a control.

29.1.14.6.1 TCustomListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

29.1.14.6.2 TCustomListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2340) is true.

See Also

AutoComplete (see page 2340)

29.1.14.6.3 TCustomListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D

29.1.14.6.4 TCustomListBox.Canvas Property

Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2357) event of an owner-draw list box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas is only useful for applications that control the drawing of list box items, that is, for owner-draw list boxes. Canvas is used to draw graphical images that can be used as items in a list box.

By default a TControlCanvas object is created for the Canvas property in the list box constructor. To create an owner-draw list box that uses a canvas, set the Style (see page 2353) property accordingly.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TListBox *myListBox;
```

```
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TListBox")
  {
    myListBox = dynamic_cast<TListBox *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myListBox->Canvas->MoveTo(myListBox->ClientRect.Left, myListBox->ClientRect.Top);
    myListBox->Canvas->LineTo(myListBox->ClientRect.Right, myListBox->ClientRect.Bottom);
    myListBox = NULL;
  }
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TListBox) then
  begin
    myListBox:= (ListBox1.Items.Objects[Index] as TListBox);
    with myListBox.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myListBox.Canvas.MoveTo(Left, Top);
      myListBox.Canvas.LineTo(Right, Bottom);
    end;
  myListBox:= nil;
  end;
```

See Also

ItemHeight (see page 2346)

OnDrawItem (see page 2357)

Style (see page 2353)

TCanvas

TControlCanvas

29.1.14.6.5 TCustomListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 2346)

Width

29.1.14.6.6 TCustomListBox.Count Property

Indicates the number of items in the list box.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of items in the list box. You can only set Count when the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw, in which case you must set this property.

See Also

Style (see page 2353)

29.1.14.6.7 TCustomListBox.ExtendedSelect Property

Determines if the user can select a sequential range of items in the list box.

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

ExtendedSelect works with the MultiSelect property. If ExtendedSelect is true and MultiSelect is true, the user can select an item then hold down the Shift key and select another and all the items in between the two selected items also become selected.

If the user doesn't hold down the Shift or Ctrl key while selecting a second item, the first selected item becomes unselected— in other words, the user must use the Ctrl key to select multiple noncontiguous items, or the Shift key to select a range of items.

If ExtendedSelect is false and MultiSelect is true, the user can select multiple items without using the Shift or Ctrl key, but they can't select a range of items in one operation.

If MultiSelect is false, the setting of ExtendedSelect has no effect as the user will not be able to select more than one item at a time in the list box.

See Also

MultiSelect

29.1.14.6.8 TCustomListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style (see page 2353) property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight (see page 2346) property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style (see page 2353) property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 2346)

Items (see page 2346)

Style (see page 2353)

Using TDBListBox and TDBComboBox

29.1.14.6.9 TCustomListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style (see page 2353) property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight (see page 2345) property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style (see page 2353) property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 2326)

IntegralHeight (see page 2345)

Items (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

Style (see page 2353)

Using TDBListBox and TDBComboBox

29.1.14.6.10 TCustomListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").t_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).t_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
```

```
    for I := 0 to ListBox1.Items.Count -1 do
      ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    Listbox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;
```

See Also

Clear (see page 2321)

ItemIndex

OnDrawItem (see page 2357)

Selected (see page 2350)

TStrings

29.1.14.6.11 TCustomListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.14.6.12 TCustomListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 2344)

ClientWidth

29.1.14.6.13 TCustomListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected [Index: Integer]: Boolean;
```

C++

```
__property Boolean Selected[int Index];
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").t_str(),
```

```
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).t_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
```

See Also

extendedselect (see page 2345)

Items (see page 2346)

MultiSelect

SelCount

29.1.14.6.14 TCustomListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
```

```
  This example uses an edit box, a list box, and two buttons
  on a form. The buttons are named Add and Sort. When the user
  clicks the Add button, the text in the edit box is added to
  the list in the list box. Before the user clicks the sort
  button, any new items are added to the end of the list.
  After the user clicks the Sort button, the list in the list
  box is sorted and remains sorted, even if additional strings
  are added.  If the user clicks the sort button again, the
  list box remains sorted, but any new strings are again added
  to the end of the list.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;


  procedure TForm1.AddClick(Sender: TObject);
  begin
    ListBox1.Items.Add(Edit1.Text);
  end;

  procedure TForm1.SortClick(Sender: TObject);
  begin
    ListBox1.Sorted := True;
  end;
```

See Also

Items (see page 2346)

Style (see page 2353)

29.1.14.6.15 TCustomListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

Items (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

OnData (see page 2356)

OnDataFind (see page 2356)

OnDataObject (see page 2357)

29.1.14.6.16 TCustomListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.14.6.17 TCustomListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

29.1.14.6.18 TCustomListBox.TopIndex Property

Specifies the index number of the item that appears at the top of the list box.

Pascal

```
property TopIndex: Integer;
```

C++

```
__property int TopIndex;
```

Description

Use TopIndex property to find or set the first item displayed at the top of the list box. TopIndex can be used, for example, to change the topmost item to a different item in the list.

C++ Examples:

```
/*
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
that item appears in the edit box. The index value displayed
is 2, indicating the third item in the list (the first item
in the list has an index value of 0):
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->TopIndex = 2;
  Edit1->Text = IntToStr(ListBox1->TopIndex);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int Number = 1; Number <= 20; Number++)
    ListBox1->Items->Add("Item " + IntToStr(Number));
}
```

Delphi Examples:

```
{
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
that item appears in the edit box. The index value displayed
is 2, indicating the third item in the list (the first item
in the list has an index value of 0):
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.TopIndex := 2;
  Edit1.Text := IntToStr(ListBox1.TopIndex);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  Number: Integer;
begin
  for Number := 1 to 20 do
    ListBox1.Items.Add('Item ' + IntToStr(Number));
end;
```

See Also

ItemIndex

Items (see page 2346)

Sorted (see page 2352)

29.1.14.7 TCustomListBox Events

The events of the TCustomListBox class are listed here.

Protected Events

	Name	Description
	OnData (see page 2356)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2356)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2357)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDrawItem (see page 2357)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnMeasureItem (see page 2359)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

29.1.14.7.1 TCustomListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 2356)

OnDataObject (see page 2357)

Style (see page 2353)

Count (see page 2344)

29.1.14.7.2 TCustomListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style (see page 2353) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 2356)

OnDataObject (see page 2357)

Style (see page 2353)

Count (see page 2344)

29.1.14.7.3 TCustomListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 2356)

OnDataFind (see page 2356)

Style (see page 2353)

Count (see page 2344)

29.1.14.7.4 TCustomListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style (see page 2353) values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight (see page 2346) property for fixed owner-draw list boxes or by the response to the OnMeasureItem (see page 2359) event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
```

```
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
```

```
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

MeasureItem (see page 2333)

OnMeasureItem (see page 2359)

Style (see page 2353)

29.1.14.7.5 TCustomListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style (see page 2353) property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent (see page 2918) which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the

control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem (see page 2357) event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Making these static keeps them around until the project terminates.
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.
```

```
}
```

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

Style (see page 2353)

29.1.15 TCustomMemo Class

TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2649) and TDBMemo.

Class Hierarchy

Pascal

```
TCustomMemo = class(TCustomEdit);
```

C++

```
class TCustomMemo : public TCustomEdit;
```

File

StdCtrls

Description

TCustomMemo introduces new properties and methods to expand the capabilities of TCustomEdit (see page 2263) for handling multiline edit boxes.

Do not create instances of TCustomMemo. Use TCustomMemo as a base class when declaring control objects that implement a multiline edit control. Properties and methods of TCustomMemo provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TDBMemo

TMemo (see page 2649)

29.1.15.1 TCustomMemo Members

The following tables list the members exposed by TCustomMemo.

Protected Events

	Name	Description
	OnChange (see page 2299)	Occurs when the text for the edit control may have changed.

Protected Methods

	Name	Description
	Change (see page 2266)	Generates an OnChange (see page 2299) event.
	CreateParams (see page 2364)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWindowHandle (see page 2365)	Creates a window based on the window-creation parameter record.
	CreateWnd (see page 2270)	Creates a Windows control corresponding to the edit control.
	DestroyWnd (see page 2273)	Destroys the Windows control that corresponds to the edit control.
	DoSetMaxLength (see page 2273)	Sets the MaxLength (see page 2286) property.
	DoSetTextHint (see page 2273)	This is DoSetTextHint, a member of class TCustomEdit.
	GetCaretPos (see page 2365)	Returns the value of the CaretPos (see page 2368) property.
	GetSelLength (see page 2274)	Returns the SelLength (see page 2289) property.
	GetSelStart (see page 2274)	Returns the SelStart (see page 2293) property.
	GetSelText (see page 2275)	Returns the SelText (see page 2296) property.
	KeyPress (see page 2366)	Ensures that return characters are not entered into the text when WantReturns (see page 2372) is false.
	Loaded (see page 2366)	Initializes the memo control when it is first loaded into memory.
	SetAlignment (see page 2279)	Sets the Alignment (see page 2283) property.
	SetAutoSize (see page 2279)	Sets the AutoSize (see page 2284) property.
	SetCaretPos (see page 2366)	Sets the CaretPos (see page 2368) property.
	SetLines (see page 2367)	Sets the Lines (see page 2368) property.
	SetScrollBars (see page 2367)	Sets the ScrollBars (see page 2369) property.
	SetSelLength (see page 2280)	Sets the SelLength (see page 2289) property.
	SetSelStart (see page 2280)	Sets the SelStart (see page 2293) property.
	SetWordWrap (see page 2367)	Sets the WordWrap (see page 2372) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2284)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2284)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2285)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2286)	Determines the case of the text within the edit control.
	HideSelection (see page 2286)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2286)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2287)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2288)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2288)	Determines where a control looks for its color information.
	PasswordChar (see page 2289)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	ScrollBars (see page 2369)	Determines whether the memo control has scroll bars.
	WantReturns (see page 2372)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2372)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2372)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Methods

	Name	Description
	Clear (see page 2267)	Deletes all text from the edit control.
	ClearSelection (see page 2267)	Removes the selected text from the edit control.
	ClearUndo (see page 2268)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2268)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2363)	Creates and initializes an instance of TCustomMemo (see page 2361).

	CutToClipboard (see page 2271)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 2272)	Provides message handling for all messages that the control does not fully process itself.
V	Destroy (see page 2364)	Destroys an instance of TCustomMemo (see page 2361).
V	GetControlsAlignment (see page 2274)	Indicates how text is aligned within the control.
V	GetSelTextBuf (see page 2275)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2276)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2278)	Selects all text in the edit control.
	SetSelText (see page 2281)	Sets the SelText (see page 2296) property.
	SetSelTextBuf (see page 2281)	Replaces the selected text with a null-terminated string.
	Undo (see page 2282)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	Alignment (see page 2283)	Determines how the text is aligned within the text edit control.
R	CanUndo (see page 2285)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 2368)	Indicates the position of the caret in the client area of the memo.
	Lines (see page 2368)	Contains the individual lines of text in the memo control.
	Modified (see page 2287)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2289)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2289)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2293)	Specifies the position of the first selected character in the text.
	SelText (see page 2296)	Specifies the selected portion of the edit control's text.
	Text (see page 2297)	Contains a text string associated with the control.
	TextHint (see page 2299)	A hint or message to be displayed when the Text (see page 2297) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2297)	Determines if the user can tab to a control.

29.1.15.2 TCustomMemo.Create Constructor

Creates and initializes an instance of TCustomMemo (see page 2361).

Pascal

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

C++

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

Description

Call Create to instantiate a TCustomMemo (see page 2361) object.

AOwner is the component, typically the form, that is responsible for freeing the memo object. This becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before making any additional initializations.

See Also

Owner

29.1.15.3 TCustomMemo.Destroy Destructor

Destroys an instance of TCustomMemo (see page 2361).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomMemo();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCustomMemo (see page 2361) reference is not nil, and only then calls Destroy.

Destroy frees the TStrings object that implements the Lines (see page 2368) property before calling the inherited destructor.

See Also

System::TObject::Free (see page 3382)

Lines (see page 2368)

TStrings

29.1.15.4 TCustomMemo Methods

The methods of the TCustomMemo class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2364)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWindowHandle (see page 2365)	Creates a window based on the window-creation parameter record.
	GetCaretPos (see page 2365)	Returns the value of the CaretPos (see page 2368) property.
	KeyPress (see page 2366)	Ensures that return characters are not entered into the text when WantReturns (see page 2372) is false.
	Loaded (see page 2366)	Initializes the memo control when it is first loaded into memory.
	SetCaretPos (see page 2366)	Sets the CaretPos (see page 2368) property.
	SetLines (see page 2367)	Sets the Lines (see page 2368) property.
	SetScrollBars (see page 2367)	Sets the ScrollBars (see page 2369) property.
	SetWordWrap (see page 2367)	Sets the WordWrap (see page 2372) property.

29.1.15.4.1 TCustomMemo.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2270) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 2365). TCustomMemo (see page 2361) overrides the inherited method to specify a scrollable multiline edit control.

The Params parameter is updated to reflect the current values of the Alignment (see page 2283), ScrollBars (see page 2369), and WordWrap (see page 2372) properties, in addition to the BorderStyle (see page 2285), HideSelection (see page 2286), PasswordChar (see page 2289), and ReadOnly (see page 2289) properties that the inherited method specifies.

See Also

Alignment

BorderStyle (see page 2285)

CreateWindowHandle (see page 2365)

CreateWnd (see page 2270)

HideSelection (see page 2286)

PasswordChar (see page 2289)

ReadOnly (see page 2289)

ScrollBars (see page 2369)

WordWrap (see page 2372)

29.1.15.4.2 TCustomMemo.CreateWindowHandle Method

Creates a window based on the window-creation parameter record.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 2270) method calls CreateWindowHandle to create the edit control window once it has been specified in the window-creation parameters. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. CreateWindowHandle overrides the inherited method to interpret the Caption in Params as the initial text rather than as the window caption.

See Also

CreateParams (see page 2364)

CreateWnd (see page 2270)

29.1.15.4.3 TCustomMemo.GetCaretPos Method

Returns the value of the CaretPos (see page 2368) property.

Pascal

```
function GetCaretPos: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetCaretPos();
```

Description

CaretPos (see page 2368) represents the coordinates of the cursor. GetCaretPos is the protected implementation of the CaretPos (see page 2368) property. Override this method to change the way the caret position is computed.

See Also

CaretPos (see page 2368)

29.1.15.4.4 TCustomMemo.KeyPress Method

Ensures that return characters are not entered into the text when WantReturns (see page 2372) is false.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the user presses a key. After calling the inherited KeyPress, which calls the OnKeyPress event handler, KeyPress discards return characters when WantReturns (see page 2372) is false.

See Also

WantReturns (see page 2372)

29.1.15.4.5 TCustomMemo.Loaded Method

Initializes the memo control when it is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Loaded overrides the inherited method so that the Modified (see page 2287) property is set to false when the form containing the memo is first loaded into memory. Derived classes can override the Loaded method to perform additional initializations after the memo's form is loaded.

See Also

Modified (see page 2287)

29.1.15.4.6 TCustomMemo.SetCaretPos Method

Sets the CaretPos (see page 2368) property.

Pascal

```
procedure SetCaretPos(const Value: TPoint); virtual;
```

C++

```
virtual __fastcall SetCaretPos(const TPoint Value);
```

Description

SetCaretPos is the property write access method for the CaretPos (see page 2368) property. The CaretPos (see page 2368) property returns the coordinates of the cursor. SetCaretPos sets the cursor position to Value.

See Also

CaretPos (see page 2368)

29.1.15.4.7 TCustomMemo.SetLines Method

Sets the Lines (see page 2368) property.

Pascal

```
procedure SetLines(Value: TStrings);
```

C++

```
__fastcall SetLines(TStrings Value);
```

Description

SetLines is the protected write implementation for the Lines (see page 2368) property.

SetLines is exposed as a protected method so derived classes can customize their implementation of the Lines (see page 2368) property.

See Also

Lines (see page 2368)

29.1.15.4.8 TCustomMemo.SetScrollBars Method

Sets the ScrollBars (see page 2369) property.

Pascal

```
procedure SetScrollBars(Value: TScrollStyle);
```

C++

```
__fastcall SetScrollBars(TScrollStyle Value);
```

Description

SetScrollBars is the property access method for the ScrollBars (see page 2369) property.

SetScrollBars is exposed as a protected method so derived classes can customize their implementation of the ScrollBars (see page 2369) property.

See Also

ScrollBars (see page 2369)

29.1.15.4.9 TCustomMemo.SetWordWrap Method

Sets the WordWrap (see page 2372) property.

Pascal

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

C++

```
__fastcall SetWordWrap(Boolean Value);
```

Description

SetWordWrap is the property access method for the WordWrap (see page 2372) property.

See Also

WordWrap (see page 2372)

29.1.15.5 TCustomMemo Properties

The properties of the TCustomMemo class are listed here.

Protected Properties

	Name	Description
	ScrollBars (see page 2369)	Determines whether the memo control has scroll bars.
	WantReturns (see page 2372)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2372)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2372)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Properties

	Name	Description
	CaretPos (see page 2368)	Indicates the position of the caret in the client area of the memo.
	Lines (see page 2368)	Contains the individual lines of text in the memo control.

29.1.15.5.1 TCustomMemo.CaretPos Property

Indicates the position of the caret in the client area of the memo.

Pascal

```
property CaretPos: TPoint;
```

C++

```
__property TPoint CaretPos;
```

Description

Use CaretPos to determine the coordinates of the cursor. CaretPos defines a location onscreen, in characters, with the origin in the top left corner of the memo. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

To determine the position of the caret in terms of characters of text only (rather than X/Y location), use the SelStart (see page 2293) property.

See Also

ClientRect

ClientOrigin

SelStart (see page 2293)

29.1.15.5.2 TCustomMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text (see page 2297) property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings

SelText (see page 2296)

SetLines (see page 2367)

Text

WordWrap (see page 2372)

29.1.15.5.3 TCustomMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
```

```
  style of the RichEdit. Toggling the Check Box turns
  HideScrollBars on and off.  Clicking the button loads the
  Rich Edit with a large amount of contiguous text.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    RichEdit1.Lines.Clear;
    for I := 0 to 500 do
    begin
      RichEdit1.Text :=
        RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
      if (i mod 4 = 0) then
        RichEdit1.Text := RichEdit1.Text + Chr(13);
    end;
    // move text cursor to top of editend;
    RichEdit1.SelStart := 0;
  end;

  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    RichEdit1.HideScrollbars := CheckBox1.Checked;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  const
    ScrollBarA: array[0..3] of TScrollStyle = (
      ssBoth,
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

SetScrollBars (see page 2367)

WordWrap (see page 2372)

Displaying and Editing Text in a Memo Control

29.1.15.5.4 TCustomMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text

WantTabs (see page 2372)

WordWrap (see page 2372)

29.1.15.5.5 TCustomMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text

WantReturns (see page 2372)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

29.1.15.5.6 TCustomMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

Lines (see page 2368)

ScrollBars (see page 2369)

Text

WantReturns (see page 2372)

Displaying and Editing Text in a Memo Control

29.1.16 TCustomStaticText Class

TCustomStaticText is the base class for TStaticText (see page 2853).

Class Hierarchy

Pascal

```
TCustomStaticText = class(TWinControl);
```

C++

```
class TCustomStaticText : public TWinControl;
```

File

StdCtrls

Description

TCustomStaticText is the base class for TStaticText (see page 2853), a windowed control that displays text on a form. This component functions like TLabel (see page 2522), except that it has a window handle.

TCustomStaticText cannot receive focus or contain other controls.

See Also

TLabel (see page 2522)

TStaticText (see page 2853)

29.1.16.1 TCustomStaticText Members

The following tables list the members exposed by TCustomStaticText.

Protected Methods

	Name	Description
	CreateParams (see page 2376)	Initializes a window-creation parameter data structure.
	Loaded (see page 2377)	Initializes the control after it is loaded from a stream.
	Notification (see page 2377)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2378)	This is SetAutoSize, a member of class TCustomStaticText.

Protected Properties

	Name	Description
	Alignment (see page 2378)	Determines how text is aligned within the component.
	AutoSize (see page 2379)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderStyle (see page 2379)	Determines the appearance of the control's border.
	FocusControl (see page 2379)	Links the component with another windowed control.
	ShowAccelChar (see page 2380)	Enables accelerator characters.

	Transparent (see page 2380)	Specifies whether the background of the image obscures objects below the image object.

Public Methods

	Name	Description
	Create (see page 2376)	Creates and initializes a TCustomStaticText (see page 2375) instance.

29.1.16.2 TCustomStaticText.Create Constructor

Creates and initializes a TCustomStaticText (see page 2375) instance.

Pascal

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

C++

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

Description

Create calls the inherited constructor, then initializes the TCustomStaticText (see page 2375) instance

See Also

autosize (see page 2379)

ControlStyle

ShowAccelChar (see page 2380)

29.1.16.3 TCustomStaticText Methods

The methods of the TCustomStaticText class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2376)	Initializes a window-creation parameter data structure.
	Loaded (see page 2377)	Initializes the control after it is loaded from a stream.
	Notification (see page 2377)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2378)	This is SetAutoSize, a member of class TCustomStaticText.

29.1.16.3.1 TCustomStaticText.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information

needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

29.1.16.3.2 TCustomStaticText.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

29.1.16.3.3 TCustomStaticText.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

29.1.16.3.4 TCustomStaticText.SetAutoSize Method

Pascal

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

C++

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

Description

This is SetAutoSize, a member of class TCustomStaticText.

29.1.16.4 TCustomStaticText Properties

The properties of the TCustomStaticText class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2378)	Determines how text is aligned within the component.
	AutoSize (see page 2379)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderStyle (see page 2379)	Determines the appearance of the control's border.
	FocusControl (see page 2379)	Links the component with another windowed control.
	ShowAccelChar (see page 2380)	Enables accelerator characters.
	Transparent (see page 2380)	Specifies whether the background of the image obscures objects below the image object.

29.1.16.4.1 TCustomStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font

Caption

29.1.16.4.2 TCustomStaticText.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

29.1.16.4.3 TCustomStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

29.1.16.4.4 TCustomStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText (see page 2375) instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption

FocusControl (see page 2313)

TWinControl

29.1.16.4.5 TCustomStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption

29.1.16.4.6 TCustomStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText (see page 2375) object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

29.1.17 TEdit Class

TEdit is a wrapper for a Windows single-line edit control.

Class Hierarchy

Pascal

```
TEdit = class(TCustomEdit);
```

C++

```
class TEdit : public TCustomEdit;
```

File

StdCtrls

Description

Use a TEdit object to put a standard Windows edit control on a form. Edit controls are used to retrieve text that users type. Edit controls can also display text to the user.

When only displaying text to the user, choose an edit control to allow users to select text and copy it to the Clipboard. Choose a label object if the selection capabilities of an edit control are not needed.

TEdit implements the generic behavior introduced in TCustomEdit (see page 2263). TEdit publishes many of the properties inherited from TCustomEdit (see page 2263), but does not introduce any new behavior. For specialized edit controls, use other descendant classes of TCustomEdit (see page 2263) or derive from it.

See Also

TDBEdit

TDBMemo

TInplaceEdit

TLabel (see page 2522)

TMaskEdit

TMemo (see page 2649)

TRichEdit

Displaying Field Component Values in Standard Controls

Edit Controls

29.1.17.1 TEdit Members

The following tables list the members exposed by TEdit.

Protected Methods

	Name	Description
	Change (see page 2266)	Generates an OnChange (see page 2299) event.
V	CreateParams (see page 2269)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2270)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2270)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2273)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2273)	Sets the MaxLength (see page 2286) property.
V	DoSetTextHint (see page 2273)	This is DoSetTextHint, a member of class TCustomEdit.
V	GetSelLength (see page 2274)	Returns the SelLength (see page 2289) property.
V	GetSelStart (see page 2274)	Returns the SelStart (see page 2293) property.
V	GetSelText (see page 2275)	Returns the SelText (see page 2296) property.
	SetAlignment (see page 2279)	Sets the Alignment (see page 2283) property.
V	SetAutoSize (see page 2279)	Sets the AutoSize (see page 2284) property.
V	SetSelLength (see page 2280)	Sets the SelLength (see page 2289) property.
V	SetSelStart (see page 2280)	Sets the SelStart (see page 2293) property.

Public Methods

	Name	Description
	Clear (see page 2267)	Deletes all text from the edit control.
	ClearSelection (see page 2267)	Removes the selected text from the edit control.
	ClearUndo (see page 2268)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2268)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2265)	Creates an instance of TCustomEdit (see page 2263).
	CutToClipboard (see page 2271)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2272)	Provides message handling for all messages that the control does not fully process itself.
	GetControlsAlignment (see page 2274)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2275)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2276)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2278)	Selects all text in the edit control.
	SetSelText (see page 2281)	Sets the SelText (see page 2296) property.
	SetSelTextBuf (see page 2281)	Replaces the selected text with a null-terminated string.
	Undo (see page 2282)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	CanUndo (see page 2285)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2287)	Indicates whether the user edited the text of the edit control.
	SelLength (see page 2289)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2293)	Specifies the position of the first selected character in the text.
	SelText (see page 2296)	Specifies the selected portion of the edit control's text.

Published Events

	Name	Description
	OnChange (see page 2419)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2419)	Occurs when the user clicks the control.
	OnContextPopup (see page 2421)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2422)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2422)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2425)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2428)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2429)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2430)	Occurs when a control receives the input focus.
	OnExit (see page 2432)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2435)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2438)	Occurs when a key is pressed.
	OnKeyUp (see page 2440)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2441)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2442)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2445)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2445)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2445)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 2447)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2450)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2450)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2385)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2388)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2389)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 2389)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2390)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 2391)	Specifies which edges of the control are beveled.
	BevelInner (see page 2391)	Specifies the cut of the inner bevel.
	BevelKind (see page 2392)	Specifies the control's bevel style.
	BevelOuter (see page 2392)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2392)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2393)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2393)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2394)	Determines the case of the text within the edit control.
	Color (see page 2394)	Specifies the background color of the control.
	Constraints (see page 2395)	Specifies the size constraints for the control.
	Ctl3D (see page 2395)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2396)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2398)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2398)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2398)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2400)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2400)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2402)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2402)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2403)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 2403)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2404)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2404)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2404)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2405)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2405)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2406)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2406)	Determines where a control looks for its font information.
	ParentShowHint (see page 2406)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 2407)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 2407)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2412)	Determines whether the user can change the text of the edit control.

	ShowHint (see page 2413)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2414)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2415)	Determines if the user can tab to a control.
	Text (see page 2415)	Contains a text string associated with the control.
	TextHint (see page 2416)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 2417)	Determines whether the component appears onscreen.

29.1.17.2 TEdit Properties

The properties of the TEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 2385)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2388)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2389)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 2389)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2390)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 2391)	Specifies which edges of the control are beveled.
	BevelInner (see page 2391)	Specifies the cut of the inner bevel.
	BevelKind (see page 2392)	Specifies the control's bevel style.
	BevelOuter (see page 2392)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2392)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2393)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2393)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2394)	Determines the case of the text within the edit control.
	Color (see page 2394)	Specifies the background color of the control.
	Constraints (see page 2395)	Specifies the size constraints for the control.
	Ctl3D (see page 2395)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2396)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2398)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2398)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2398)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2400)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2400)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2402)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2402)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2403)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 2403)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2404)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2404)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2404)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2405)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2405)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 2406)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2406)	Determines where a control looks for its font information.
	ParentShowHint (see page 2406)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 2407)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 2407)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2412)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 2413)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2414)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2415)	Determines if the user can tab to a control.
	Text (see page 2415)	Contains a text string associated with the control.
	TextHint (see page 2416)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 2417)	Determines whether the component appears onscreen.

29.1.17.2.1 TEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
```

```
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

29.1.17.2.2 TEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control

taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2279)

Text

Displaying and Editing Fields in an Edit Box

29.1.17.2.3 TEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.17.2.4 TEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 2278)

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.17.2.5 TEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 2285)

ClientHeight

ControlStyle

Height

29.1.17.2.6 TEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

29.1.17.2.7 TEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

29.1.17.2.8 TEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

29.1.17.2.9 TEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

29.1.17.2.10 TEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

29.1.17.2.11 TEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.17.2.12 TEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If

BorderStyle is bsNone, there will be no border.

29.1.17.2.13 TEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

29.1.17.2.14 TEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.17.2.15 TEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.17.2.16 TEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.17.2.17 TEdit.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
```

```
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.17.2.18 TEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.17.2.19 TEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.17.2.20 TEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.17.2.21 TEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.17.2.22 TEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.17.2.23 TEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.17.2.24 TEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

29.1.17.2.25 TEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

29.1.17.2.26 TEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

29.1.17.2.27 TEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly: Boolean;
```

C++

```
__property Boolean NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

29.1.17.2.28 TEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

29.1.17.2.29 TEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.17.2.30 TEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.17.2.31 TEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.17.2.32 TEdit.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.17.2.33 TEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.17.2.34 TEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.17.2.35 TEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

29.1.17.2.36 TEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
```

```
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.17.2.37 TEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then

be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2268)

GetSelTextBuf (see page 2275)

29.1.17.2.38 TEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.17.2.39 TEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the

control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.17.2.40 TEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.17.2.41 TEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
```

```
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2368)

29.1.17.2.42 TEdit.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

29.1.17.2.43 TEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.17.3 TEdit Events

The events of the TEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 2419)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2419)	Occurs when the user clicks the control.
	OnContextPopup (see page 2421)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2422)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2422)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2425)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2428)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2429)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2430)	Occurs when a control receives the input focus.
	OnExit (see page 2432)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2435)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2438)	Occurs when a key is pressed.
	OnKeyUp (see page 2440)	Occurs when the user releases a key that has been pressed.

	OnMouseActivate (see page 2441)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2442)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2445)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2445)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2445)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2447)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2450)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2450)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.17.3.1 TEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2266)

Text

29.1.17.3.2 TEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).

- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.17.3.3 TEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.17.3.4 TEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.17.3.5 TEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.17.3.6 TEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
```

```
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.17.3.7 TEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.17.3.8 TEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.17.3.9 TEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.17.3.10 TEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

29.1.17.3.11 TEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  Behavior is dependent on keyboard driver(USB, serial, etc).
  */
  void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
        TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      FormColor = Form1->Color;
      Form1->Color = clAqua;
      if (ComboBox1->Text == "KeyDown")
        ComboBox1->Text = "KeyDown2";
      else
        ComboBox1->Text = "KeyDown";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }

  void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      Form1->Color = FormColor;
      if (ComboBox1->Text == "KeyUp")
        ComboBox1->Text = "KeyUp2";
      else
        ComboBox1->Text = "KeyUp";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }
  /*
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  */
  #include <Printers.hpp>

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
     KeyPreview = True;

  }

  void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

  {
    if (Key == VK_ESCAPE && Printer()->Printing)
    {
      Printer()->Abort();
      MessageDlg("Printing aborted", mtInformation,
                 TMsgDlgButtons() << mbOK,0);
    }
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
    Memo1->Parent = Form1;
    Memo1->Visible = True;
    Memo1->Width = 700;
    if (OpenDialog1->Execute())
    {
      Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
```

```
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.17.3.12 TEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.17.3.13 TEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.17.3.14 TEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.17.3.15 TEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.17.3.16 TEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.17.3.17 TEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.17.3.18 TEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.17.3.19 TEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.17.3.20 TEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.17.3.21 TEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.18 TGroupBox Class

TGroupBox represents a Windows group box.

Class Hierarchy

Pascal

```
TGroupBox = class(TCustomGroupBox);
```

C++

```
class TGroupBox : public TCustomGroupBox;
```

File

StdCtrls

Description

The TGroupBox component represents a standard Windows group box, used to group related controls on a form. When another control component is placed within a group box, the group box becomes the parent of that component.

See Also

TRadioGroup

TDBRadioGroup

TControl

Group Boxes and Radio Groups

Grouping Controls

29.1.18.1 TGroupBox Members

The following tables list the members exposed by TGroupBox.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2301)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2301)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2302)	Renders the onscreen image of the group box.

Public Methods

	Name	Description
	Create (see page 2300)	Creates and initializes a TCustomGroupBox (see page 2299) instance.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2482)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2482)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 2483)	Occurs when the user clicks the control.
	OnContextPopup (see page 2485)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2486)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2486)	Occurs when another control is docked to the control.
	OnDockOver (see page 2488)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2490)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2493)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2496)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2497)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2498)	Occurs when a control receives the input focus.
	OnExit (see page 2500)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2503)	Returns the control's docking information.
	OnMouseActivate (see page 2505)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2506)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2508)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2508)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2509)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2511)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2513)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2514)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2514)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 2454)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2457)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2458)	Specifies the bi-directional mode for the control.
	Caption (see page 2458)	Specifies a text string that identifies the control to the user.
	Color (see page 2461)	Specifies the background color of the control.
	Constraints (see page 2462)	Specifies the size constraints for the control.
	Ctl3D (see page 2462)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2463)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 2464)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2465)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2466)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2466)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2468)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2468)	Controls the attributes of text written on or in the control.
	Padding (see page 2469)	Specifies the padding of a control.
	ParentBackground (see page 2469)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2470)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2470)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2471)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2471)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2471)	Determines where a control looks for its font information.
	ParentShowHint (see page 2472)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2472)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2477)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2479)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2479)	Determines if the user can tab to a control.
	Visible (see page 2480)	Determines whether the component appears onscreen.

29.1.18.2 TGroupBox Properties

The properties of the TGroupBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2454)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2457)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2458)	Specifies the bi-directional mode for the control.
	Caption (see page 2458)	Specifies a text string that identifies the control to the user.
	Color (see page 2461)	Specifies the background color of the control.
	Constraints (see page 2462)	Specifies the size constraints for the control.
	Ctl3D (see page 2462)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2463)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2464)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2465)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2466)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2466)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2468)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2468)	Controls the attributes of text written on or in the control.
	Padding (see page 2469)	Specifies the padding of a control.
	ParentBackground (see page 2469)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2470)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2470)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2471)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2471)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

	ParentFont (see page 2471)	Determines where a control looks for its font information.
	ParentShowHint (see page 2472)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2472)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2477)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2479)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2479)	Determines if the user can tab to a control.
	Visible (see page 2480)	Determines whether the component appears onscreen.

29.1.18.2.1 TGroupBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
```

```
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

29.1.18.2.2 TGroupBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.18.2.3 TGroupBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.18.2.4 TGroupBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
```

```
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

29.1.18.2.5 TGroupBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.18.2.6 TGroupBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.18.2.7 TGroupBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.18.2.8 TGroupBox.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
end;
```

29.1.18.2.9 TGroupBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.18.2.10 TGroupBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.18.2.11 TGroupBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.18.2.12 TGroupBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
```

```
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.18.2.13 TGroupBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.18.2.14 TGroupBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
```

```
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.18.2.15 TGroupBox.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

29.1.18.2.16 TGroupBox.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

29.1.18.2.17 TGroupBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.18.2.18 TGroupBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.18.2.19 TGroupBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.18.2.20 TGroupBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.18.2.21 TGroupBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.18.2.22 TGroupBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.18.2.23 TGroupBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
```

```
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.18.2.24 TGroupBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.18.2.25 TGroupBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.18.2.26 TGroupBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.18.2.27 TGroupBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.18.3 TGroupBox Events

The events of the TGroupBox class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2482)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2482)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 2483)	Occurs when the user clicks the control.
	OnContextPopup (see page 2485)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2486)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2486)	Occurs when another control is docked to the control.
	OnDockOver (see page 2488)	Occurs when another control is dragged over the control.

	OnDragDrop (see page 2490)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2493)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2496)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2497)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2498)	Occurs when a control receives the input focus.
	OnExit (see page 2500)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2503)	Returns the control's docking information.
	OnMouseActivate (see page 2505)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2506)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2508)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2508)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2509)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2511)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2513)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2514)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2514)	Occurs when the application tries to undock a control that is docked to the windowed control.

29.1.18.3.1 TGroupBox.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

29.1.18.3.2 TGroupBox.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

29.1.18.3.3 TGroupBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).

- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.18.3.4 TGroupBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.18.3.5 TGroupBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.18.3.6 TGroupBox.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}
```

```
void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
```

```
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

29.1.18.3.7 TGroupBox.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}
```

```
void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
```

```
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

29.1.18.3.8 TGroupBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}
```

```
void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
```

```
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.18.3.9 TGroupBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
```

```
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
  This code is an OnDragOver event handler that won't allow a
  label control to be dropped on a panel control and stops the
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.18.3.10 TGroupBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.18.3.11 TGroupBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.18.3.12 TGroupBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.18.3.13 TGroupBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

29.1.18.3.14 TGroupBox.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

29.1.18.3.15 TGroupBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.18.3.16 TGroupBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.18.3.17 TGroupBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.18.3.18 TGroupBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.18.3.19 TGroupBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.18.3.20 TGroupBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.18.3.21 TGroupBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.18.3.22 TGroupBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.18.3.23 TGroupBox.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
```

```
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
```

```
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

29.1.19 TImageMargins Class

Margins of image on button.

Class Hierarchy

Pascal

```
TImageMargins = class(TPersistent);
```

C++

```
class TImageMargins : public TPersistent;
```

File

StdCtrls

Description

A button image may be positioned using margins. TImageMargins defines the margins of an image on a TButton (see page 1961). The margin values are in the Bottom (see page 2520), Top (see page 2521), Left (see page 2520) and Right (see page 2520) properties.

The OnChange (see page 2521) event is triggered when a margin changes.

The TButton.ImageMargins property contains a TImageMargins instance.

See Also

Bottom (see page 2520)

ImageMargins (see page 1984)

Left (see page 2520)

Right (see page 2520)

Top (see page 2521)

OnChange (see page 2521)

29.1.19.1 TImageMargins Members

The following tables list the members exposed by TImageMargins.

Protected Methods

	Name	Description
	Change (see page 2519)	This is Change, a member of class TImageMargins.

Public Events

	Name	Description
	OnChange (see page 2521)	Margin changed.

Public Methods

	Name	Description
	Assign (see page 2518)	Copies the contents of another, similar object.

Published Properties

	Name	Description
	Bottom (see page 2520)	Bottom margin of image on button.
	Left (see page 2520)	Left margin of image on button.
	Right (see page 2520)	Right margin of image on button.
	Top (see page 2521)	Top margin of image on button.

29.1.19.2 TImageMargins Methods

The methods of the TImageMargins class are listed here.

Protected Methods

	Name	Description
	Change (see page 2519)	This is Change, a member of class TImageMargins.

Public Methods

	Name	Description
	Assign (see page 2518)	Copies the contents of another, similar object.

29.1.19.2.1 TImageMargins.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError (see page 3985)

29.1.19.2.2 TImageMargins.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TImageMargins.

29.1.19.3 TImageMargins Properties

The properties of the TImageMargins class are listed here.

Published Properties

	Name	Description
	Bottom (see page 2520)	Bottom margin of image on button.
	Left (see page 2520)	Left margin of image on button.
	Right (see page 2520)	Right margin of image on button.
	Top (see page 2521)	Top margin of image on button.

29.1.19.3.1 TImageMargins.Bottom Property

Bottom margin of image on button.

Pascal

```
property Bottom: Integer;
```

C++

```
__property int Bottom;
```

Description

A button image may be positioned using margins. Bottom is the bottom margin of an image on a TButton (see page 1961).

See Also

ImageMargins (see page 1984)

Left (see page 2520)

Right (see page 2520)

Top (see page 2521)

29.1.19.3.2 TImageMargins.Left Property

Left margin of image on button.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

A button image may be positioned using margins. Left is the left margin of an image on a TButton (see page 1961).

See Also

Bottom (see page 2520)

ImageMargins (see page 1984)

Right (see page 2520)

Top (see page 2521)

29.1.19.3.3 TImageMargins.Right Property

Right margin of image on button.

Pascal

```
property Right: Integer;
```

C++

```
__property int Right;
```

Description

A button image may be positioned using margins. Right is the right margin of an image on a TButton (see page 1961).

See Also

Bottom (see page 2520)

ImageMargins (see page 1984)

Left (see page 2520)

Top (see page 2521)

29.1.19.3.4 TImageMargins.Top Property

Top margin of image on button.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

A button image may be positioned using margins. Top is the top margin of an image on a TButton (see page 1961).

See Also

Bottom (see page 2520)

ImageMargins (see page 1984)

Left (see page 2520)

Right (see page 2520)

29.1.19.4 TImageMargins Events

The events of the TImageMargins class are listed here.

Public Events

	Name	Description
	OnChange (see page 2521)	Margin changed.

29.1.19.4.1 TImageMargins.OnChange Event

Margin changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

OnChange occurs when a margin changes.

See Also

ImageMargins (see page 1984)

29.1.20 TLabel Class

TLabel is a nonwindowed control that displays text on a form.

Class Hierarchy

Pascal

```
TLabel = class(TCustomLabel);
```

C++

```
class TLabel : public TCustomLabel;
```

File

StdCtrls

Description

Use TLabel to add text that the user can't edit to a form. This text can be used to label another control, and can set focus to that control when the user types an accelerator key.

Because TLabel is not a descendant of TWinControl, it does not have its own window and can't receive direct input from the keyboard. To add an object to a form that can respond to keyboard input (other than setting focus to another object when an accelerator key is typed) in addition to displaying text, use TStaticText (see page 2853).

To add an object to a form that displays text that a user can scroll or edit, use TEdit (see page 2380).

See Also

TEdit (see page 2380)

TStaticText (see page 2853)

Labels

Display Controls

29.1.20.1 TLabel Members

The following tables list the members exposed by TLabel.

Protected Methods

	Name	Description
	AdjustBounds (see page 2304)	Resizes the label when the text changes.
	DoDrawText (see page 2304)	Renders the text in the client area.

	GetLabelText (see page 2305)	Returns the value of the Caption (see page 2310) property.
	Loaded (see page 2305)	Adjusts the size of the label when it is first loaded. .
	Notification (see page 2305)	Responds to notifications when objects are about to be created or destroyed.
	Paint (see page 2306)	Draws the image of the label.
	SetAutoSize (see page 2306)	Sets the AutoSize (see page 2307) property.

Public Methods

	Name	Description
	Create (see page 2303)	Creates and initializes an instance of TCustomLabel (see page 2302).

Public Properties

	Name	Description
	Canvas (see page 2308)	Provides the drawing surface used by the graphic control.

Published Events

	Name	Description
	OnClick (see page 2552)	Occurs when the user clicks the control.
	OnContextPopup (see page 2554)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2554)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2555)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2558)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2561)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2561)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2562)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2563)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2566)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2566)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2566)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2568)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2571)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2571)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2525)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2528)	Controls the horizontal placement of the text within the label.
	Anchors (see page 2528)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2529)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 2530)	Specifies the bi-directional mode for the control.
	Caption (see page 2530)	Specifies a text string that identifies the control to the user.
	Color (see page 2533)	Specifies the background color of the control.
	Constraints (see page 2534)	Specifies the size constraints for the control.
	DragCursor (see page 2534)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2535)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2535)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 2537)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 2537)	Controls whether the control responds to mouse, keyboard, and timer events.

	FocusControl (see page 2538)	Designates a windowed control associated with the label.
	Font (see page 2538)	Controls the attributes of text written on or in the control.
	GlowSize (see page 2539)	This is GlowSize, a member of class TLabel.
	Layout (see page 2539)	Specifies the vertical placement of the text within the label.
	ParentBiDiMode (see page 2540)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2540)	Determines where a control looks for its color information.
	ParentFont (see page 2540)	Determines where a control looks for its font information.
	ParentShowHint (see page 2541)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2542)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2547)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 2547)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 2549)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 2549)	Determines whether the component appears onscreen.
	WordWrap (see page 2551)	Specifies whether the label text wraps when it is too long for the width of the label.

29.1.20.2 TLabel Properties

The properties of the TLabel class are listed here.

Published Properties

	Name	Description
	Align (see page 2525)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2528)	Controls the horizontal placement of the text within the label.
	Anchors (see page 2528)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2529)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 2530)	Specifies the bi-directional mode for the control.
	Caption (see page 2530)	Specifies a text string that identifies the control to the user.
	Color (see page 2533)	Specifies the background color of the control.
	Constraints (see page 2534)	Specifies the size constraints for the control.
	DragCursor (see page 2534)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2535)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2535)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 2537)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 2537)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2538)	Designates a windowed control associated with the label.
	Font (see page 2538)	Controls the attributes of text written on or in the control.
	GlowSize (see page 2539)	This is GlowSize, a member of class TLabel.
	Layout (see page 2539)	Specifies the vertical placement of the text within the label.
	ParentBiDiMode (see page 2540)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2540)	Determines where a control looks for its color information.
	ParentFont (see page 2540)	Determines where a control looks for its font information.
	ParentShowHint (see page 2541)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2542)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2547)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 2547)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 2549)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 2549)	Determines whether the component appears onscreen.
	WordWrap (see page 2551)	Specifies whether the label text wraps when it is too long for the width of the label.

29.1.20.2.1 TLabel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
```

```
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.20.2.2 TLabel.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap property is true and the label includes more than one line of text.

See Also

ClientRect

GetLabelText (see page 2305)

Layout (see page 2314)

WordWrap (see page 2316)

29.1.20.2.3 TLabel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.20.2.4 TLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text. When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
```

```
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
```

See Also

BoundsRect

GetLabelText (see page 2305)

Font

WordWrap (see page 2316)

Layout (see page 2314)

29.1.20.2.5 TLabel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.20.2.6 TLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
```

```
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
```

```
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

29.1.20.2.7 TLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.20.2.8 TLabel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.20.2.9 TLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.20.2.10 TLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.20.2.11 TLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.20.2.12 TLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 2316)

TypeName

29.1.20.2.13 TLabel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.20.2.14 TLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar property to true.

See Also

ShowAccelChar (see page 2314)

GetLabelText (see page 2305)

29.1.20.2.15 TLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.20.2.16 TLabel.GlowSize Property

Pascal

```
property GlowSize: Integer;
```

C++

```
__property int GlowSize;
```

Description

This is GlowSize, a member of class TLabel.

29.1.20.2.17 TLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment property.

See Also

Alignment (see page 2307)

ClientRect

29.1.20.2.18 TLabel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.20.2.19 TLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.20.2.20 TLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.20.2.21 TLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.20.2.22 TLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.20.2.23 TLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 2313)

GetLabelText (see page 2305)

29.1.20.2.24 TLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.20.2.25 TLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color

ParentColor

29.1.20.2.26 TLabel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.20.2.27 TLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth

GetLabelText (see page 2305)

29.1.20.3 TLabel Events

The events of the TLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 2552)	Occurs when the user clicks the control.
	OnContextPopup (see page 2554)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2554)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2555)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2558)	Occurs when the user drags an object over a control.

	OnEndDock (see page 2561)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2561)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2562)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2563)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2566)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2566)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2566)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2568)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2571)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2571)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.20.3.1 TLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.20.3.2 TLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.20.3.3 TLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.20.3.4 TLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.20.3.5 TLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.20.3.6 TLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.20.3.7 TLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.20.3.8 TLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
```

```
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.20.3.9 TLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.20.3.10 TLabel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.20.3.11 TLabel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.20.3.12 TLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.20.3.13 TLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.20.3.14 TLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.20.3.15 TLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.21 TListBox Class

TListBox displays a collection of items in a scrollable list.

Class Hierarchy

Pascal

```
TListBox = class(TCustomListBox);
```

C++

```
class TListBox : public TCustomListBox;
```

File

StdCtrls

Description

Use TListBox to display a scrollable list of items that users can select, add, or delete. TListBox is a wrapper for the Windows listbox control. For specialized list boxes, use other descendant classes of TCustomListBox (see page 2316) or derive your own class from TCustomListBox (see page 2316).

TListBox implements the generic behavior introduced in TCustomListBox (see page 2316). TListBox publishes many of the properties inherited from TCustomListBox (see page 2316), but does not introduce any new behavior.

See Also

List Controls

List Boxes and Check-list Boxes

29.1.21.1 TListBox Members

The following tables list the members exposed by TListBox.

Protected Fields

	Name	Description
	FMoving (see page 2319)	This is FMoving, a member of class TCustomListBox.

Protected Methods

	Name	Description
	CreateParams (see page 2322)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 2323)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2324)	Deletes a string from the list box.
	DestroyWnd (see page 2324)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2325)	Returns the index of a string in the list box.
	DoGetData (see page 2325)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2326)	Returns the object associated with an item in a virtual list box.
	DragCanceled (see page 2326)	Sends a mouse message canceling the drag-and-drop operation.
	DrawItem (see page 2326)	Generates an OnDrawItem (see page 2357) event.
	GetItemData (see page 2327)	Returns a 32-bit value associated with the specified item in a list box.
	GetItemIndex (see page 2328)	Returns the value of the ItemIndex property.
	GetSelCount (see page 2328)	Returns the number of selected items.
	GetSelected (see page 2328)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2328)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2329)	Associates an integer with a specified item in the list box.
	KeyPress (see page 2332)	Responds when the user presses a key.
	LoadRecreateItems (see page 2333)	This is LoadRecreateItems, a member of class TCustomListBox.
	MeasureItem (see page 2333)	Generates an OnMeasureItem (see page 2359) event.
	ResetContent (see page 2334)	Removes all of the items from a list box.
	SaveRecreateItems (see page 2334)	This is SaveRecreateItems, a member of class TCustomListBox.
	SetItemData (see page 2334)	Sets the 32-bit value associated with the specified item in a list box.
	SetItemIndex (see page 2335)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 2335)	Sets the MultiSelect property.
	WndProc (see page 2335)	Handles Windows messages.

Public Methods

	Name	Description
	AddItem (see page 2320)	Adds an item to the list box
	Clear (see page 2321)	Deletes all items from the list box.
	ClearSelection (see page 2321)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 2322)	Copies the selected items to another list-type control
	Create (see page 2318)	Instantiates a list box control.
	DeleteSelected (see page 2324)	Deletes the selected items from the list box
	Destroy (see page 2319)	Destroys the instance of a list box control.
	GetCount (see page 2327)	Returns the value of the Count (see page 2344) property.
	ItemAtPos (see page 2329)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2331)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 2334)	Selects the entire block of text in the list box.

Public Properties

	Name	Description
	Canvas (see page 2341)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2357) event of an owner-draw list box.
	Count (see page 2344)	Indicates the number of items in the list box.
	Selected (see page 2350)	Indicates whether a particular item is selected.
	TopIndex (see page 2354)	Specifies the index number of the item that appears at the top of the list box.

Published Events

	Name	Description
	OnClick (see page 2612)	Occurs when the user clicks the control.

	OnContextPopup (see page 2614)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2615)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2615)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2616)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2616)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2616)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2619)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2623)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2625)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2625)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2626)	Occurs when a control receives the input focus.
	OnExit (see page 2629)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2631)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2635)	Occurs when a key is pressed.
	OnKeyUp (see page 2636)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2638)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2639)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2640)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2643)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2643)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2643)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2646)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2648)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2648)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2576)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2580)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2580)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2580)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 2581)	Specifies which edges of the control are beveled.
	BevelInner (see page 2581)	Specifies the cut of the inner bevel.
	BevelKind (see page 2582)	Specifies the control's bevel style.
	BevelOuter (see page 2582)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2582)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2583)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2583)	Determines whether the list box has a border.
	Color (see page 2584)	Specifies the background color of the control.
	Columns (see page 2585)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2585)	Specifies the size constraints for the control.
	Ctl3D (see page 2586)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2586)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2588)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2588)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 2589)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2590)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 2591)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 2591)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2592)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2593)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2593)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2594)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2594)	Contains the strings that appear in the list box.
	MultiSelect (see page 2597)	Specifies whether the user can select more than one item.
	ParentBiDiMode (see page 2597)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2598)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2598)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2599)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2599)	Determines where a control looks for its font information.
	ParentShowHint (see page 2599)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2600)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 2605)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 2605)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2607)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2608)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2609)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2609)	Determines if the user can tab to a control.
	TabWidth (see page 2610)	Specifies the size of the tabs in the list box.
	Visible (see page 2610)	Determines whether the component appears onscreen.

29.1.21.2 TListBox Properties

The properties of the TListBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2576)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2580)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2580)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2580)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 2581)	Specifies which edges of the control are beveled.
	BevelInner (see page 2581)	Specifies the cut of the inner bevel.
	BevelKind (see page 2582)	Specifies the control's bevel style.
	BevelOuter (see page 2582)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2582)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2583)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2583)	Determines whether the list box has a border.
	Color (see page 2584)	Specifies the background color of the control.
	Columns (see page 2585)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2585)	Specifies the size constraints for the control.
	Ctl3D (see page 2586)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2586)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	DragCursor (see page 2588)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2588)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2589)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2590)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 2591)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 2591)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2592)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2593)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2593)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2594)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2594)	Contains the strings that appear in the list box.
	MultiSelect (see page 2597)	Specifies whether the user can select more than one item.
	ParentBiDiMode (see page 2597)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2598)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2598)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2599)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2599)	Determines where a control looks for its font information.
	ParentShowHint (see page 2599)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2600)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 2605)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 2605)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2607)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2608)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2609)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2609)	Determines if the user can tab to a control.
	TabWidth (see page 2610)	Specifies the size of the tabs in the list box.
	Visible (see page 2610)	Determines whether the component appears onscreen.

29.1.21.2.1 TListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.21.2.2 TListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.21.2.3 TListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

29.1.21.2.4 TListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2580) is true.

See Also

AutoComplete (see page 2580)

29.1.21.2.5 TListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

29.1.21.2.6 TListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

29.1.21.2.7 TListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

29.1.21.2.8 TListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

29.1.21.2.9 TListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

29.1.21.2.10 TListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.21.2.11 TListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D

29.1.21.2.12 TListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.21.2.13 TListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 2346)

Width

29.1.21.2.14 TListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.21.2.15 TListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.21.2.16 TListBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.21.2.17 TListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.21.2.18 TListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.21.2.19 TListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.21.2.20 TListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.21.2.21 TListBox.ExtendedSelect Property

Makes list item multi-select operate with Shift and CTRL keys.

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

ExtendedSelect only has effect when the MultiSelect (see page 2597) is true.

When ExtendedSelect is true, the Shift key (in conjunction with the left mouse button, or arrow keys) is used to select a contiguous sequence of list items. And the Ctrl key is used (in conjunction with the left mouse buttton) to select non-contiguous list items.

When ExtendedSelect is false, Multi-Select operates very differently. The Shift and Ctrl keys are not recognised. List items are always selected individually in any sequence. Reselecting a selected list item will deselect it. Use the arrow and spacebar keys, or the left mouse button to make selections.

See Also

MultiSelect (see page 2597)

29.1.21.2.22 TListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.21.2.23 TListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the

control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

29.1.21.2.24 TListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

29.1.21.2.25 TListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 2346)

Items (see page 2346)

Style (see page 2353)

Using TDBListBox and TDBComboBox

29.1.21.2.26 TListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 2326)

IntegralHeight (see page 2345)

Items (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

Style (see page 2353)

Using TDBListBox and TDBComboBox

29.1.21.2.27 TListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").t_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).t_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
```

```
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

Clear (see page 2321)

ItemIndex

OnDrawItem (see page 2357)

Selected (see page 2350)

TStrings

29.1.21.2.28 TListBox.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect;
```

C++

```
__property MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount property indicates the number of selected items. When MultiSelect is false, the user can only select one item, which is indicated by the ItemIndex property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl descendant, override the protected SetMultiSelect method.

See Also

SelCount

ItemIndex

29.1.21.2.29 TListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.21.2.30 TListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.21.2.31 TListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.21.2.32 TListBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.21.2.33 TListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.21.2.34 TListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.21.2.35 TListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
```

```
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}
```

```
void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
```

```
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
```

```
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.21.2.36 TListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 2344)

ClientWidth

29.1.21.2.37 TListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
```

```
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.21.2.38 TListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
```

```
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

See Also

Items (see page 2346)

Style (see page 2353)

29.1.21.2.39 TListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

Items (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

OnMeasureItem (see page 2359)

OnData (see page 2356)

OnDataFind (see page 2356)

OnDataObject (see page 2357)

29.1.21.2.40 TListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.21.2.41 TListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.21.2.42 TListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

29.1.21.2.43 TListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.21.3 TListBox Events

The events of the TListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 2612)	Occurs when the user clicks the control.
	OnContextPopup (see page 2614)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2615)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2615)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2616)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2616)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2616)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2619)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2623)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2625)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2625)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2626)	Occurs when a control receives the input focus.
	OnExit (see page 2629)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2631)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2635)	Occurs when a key is pressed.
	OnKeyUp (see page 2636)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2638)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2639)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2640)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2643)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2643)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2643)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2646)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2648)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2648)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.21.3.1 TListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that

action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.21.3.2 TListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property

of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.21.3.3 TListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 2356)

OnDataObject (see page 2357)

Style (see page 2353)

Count (see page 2344)

29.1.21.3.4 TListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 2356)

OnDataObject (see page 2357)

Style (see page 2353)

Count (see page 2344)

29.1.21.3.5 TListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 2356)

OnDataFind (see page 2356)

Style (see page 2353)

Count (see page 2344)

29.1.21.3.6 TListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.21.3.7 TListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
```

```
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.21.3.8 TListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
  property OnDragOver;
```

C++

```
  __property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
```

```
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.21.3.9 TListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight property for fixed owner-draw list boxes or by the response to the OnMeasureItem event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
```

```
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

MeasureItem (see page 2333)

OnMeasureItem (see page 2359)

Style (see page 2353)

29.1.21.3.10 TListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.21.3.11 TListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.21.3.12 TListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.21.3.13 TListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

29.1.21.3.14 TListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
```

```
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
```

```
  Behavior is dependent on keyboard driver(USB, serial, etc).
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.21.3.15 TListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.21.3.16 TListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.21.3.17 TListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
```

```
      int &Height)
{
  Graphics::TBitmap *bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Making these static keeps them around until the project terminates.
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.

}
```

See Also

DrawItem (see page 2326)

ItemHeight (see page 2346)

MeasureItem (see page 2333)

OnDrawItem (see page 2357)

Style (see page 2353)

29.1.21.3.18 TListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.21.3.19 TListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
```

```
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.21.3.20 TListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.21.3.21 TListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.21.3.22 TListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.21.3.23 TListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.21.3.24 TListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.21.3.25 TListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

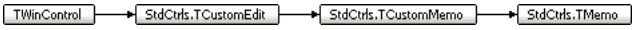
OnDragOver

OnDragDrop

29.1.22 TMemo Class

TMemo is a wrapper for a Windows multiline edit control.

Class Hierarchy



Pascal

```
TMemo = class(TCustomMemo);
```

C++

```
class TMemo : public TCustomMemo;
```

File

StdCtrls

Description

Use TMemo to put a standard Windows multiline edit control on a form. Multiline edit boxes allow the user to enter more than one line of text. They are appropriate for representing lengthy information.

TMemo implements the generic behavior introduced in TCustomMemo (see page 2361). TMemo publishes many of the properties inherited from TCustomMemo (see page 2361), but does not introduce any new behavior.

Note: Under Win 9x, there is a limit of 64k of text for this control.

See Also

TDBMemo

TEdit (see page 2380)

TRichEdit

Displaying and Editing Text in a Memo Control

Adding Scroll Bars at Runtime

Setting Text Alignment

Memo and Rich Edit Controls

Edit Controls

29.1.22.1 TMemo Members

The following tables list the members exposed by TMemo.

Protected Methods

	Name	Description
	Change (see page 2266)	Generates an OnChange (see page 2299) event.
V	CreateParams (see page 2269)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateParams (see page 2364)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2270)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 2365)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 2270)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2273)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2273)	Sets the MaxLength (see page 2286) property.
V	DoSetTextHint (see page 2273)	This is DoSetTextHint, a member of class TCustomEdit.
V	GetCaretPos (see page 2365)	Returns the value of the CaretPos (see page 2368) property.
V	GetSelLength (see page 2274)	Returns the SelLength (see page 2289) property.
V	GetSelStart (see page 2274)	Returns the SelStart (see page 2293) property.
V	GetSelText (see page 2275)	Returns the SelText (see page 2296) property.
V	KeyPress (see page 2366)	Ensures that return characters are not entered into the text when WantReturns (see page 2372) is false.
V	Loaded (see page 2366)	Initializes the memo control when it is first loaded into memory.
	SetAlignment (see page 2279)	Sets the Alignment (see page 2283) property.
V	SetAutoSize (see page 2279)	Sets the AutoSize (see page 2284) property.
V	SetCaretPos (see page 2366)	Sets the CaretPos (see page 2368) property.
	SetLines (see page 2367)	Sets the Lines (see page 2368) property.
	SetScrollBars (see page 2367)	Sets the ScrollBars (see page 2369) property.
V	SetSelLength (see page 2280)	Sets the SelLength (see page 2289) property.
V	SetSelStart (see page 2280)	Sets the SelStart (see page 2293) property.
	SetWordWrap (see page 2367)	Sets the WordWrap (see page 2372) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2284)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2284)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	NumbersOnly (see page 2287)	Allows only numbers to be typed into the text edit.

	PasswordChar (see page 2289)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Methods

	Name	Description
	Clear (see page 2267)	Deletes all text from the edit control.
	ClearSelection (see page 2267)	Removes the selected text from the edit control.
	ClearUndo (see page 2268)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2268)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2265)	Creates an instance of TCustomEdit (see page 2263).
	Create (see page 2363)	Creates and initializes an instance of TCustomMemo (see page 2361).
	CutToClipboard (see page 2271)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2272)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 2364)	Destroys an instance of TCustomMemo (see page 2361).
	GetControlsAlignment (see page 2274)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2275)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2276)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2278)	Selects all text in the edit control.
	SetSelText (see page 2281)	Sets the SelText (see page 2296) property.
	SetSelTextBuf (see page 2281)	Replaces the selected text with a null-terminated string.
	Undo (see page 2282)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	CanUndo (see page 2285)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 2368)	Indicates the position of the caret in the client area of the memo.
	Modified (see page 2287)	Indicates whether the user edited the text of the edit control.
	SelLength (see page 2289)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2293)	Specifies the position of the first selected character in the text.
	SelText (see page 2296)	Specifies the selected portion of the edit control's text.
	Text (see page 2297)	Contains a text string associated with the control.
	TextHint (see page 2299)	A hint or message to be displayed when the Text (see page 2297) property is empty.

Published Events

	Name	Description
	OnChange (see page 2690)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2690)	Occurs when the user clicks the control.
	OnContextPopup (see page 2692)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2693)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2693)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2696)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2699)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2700)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2701)	Occurs when a control receives the input focus.
	OnExit (see page 2703)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2706)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2710)	Occurs when a key is pressed.
	OnKeyUp (see page 2711)	Occurs when the user releases a key that has been pressed.

	OnMouseActivate (see page 2713)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2714)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2716)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2716)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2716)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2719)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2721)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2721)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2654)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2657)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2658)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2658)	Specifies which edges of the control are beveled.
	BevelInner (see page 2659)	Specifies the cut of the inner bevel.
	BevelKind (see page 2659)	Specifies the control's bevel style.
	BevelOuter (see page 2660)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2660)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2660)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2661)	Determines the case of the text within the edit control.
	Color (see page 2661)	Specifies the background color of the control.
	Constraints (see page 2662)	Specifies the size constraints for the control.
	Ctl3D (see page 2663)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2663)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2665)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2665)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2666)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2667)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2668)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2669)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2669)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2670)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 2670)	Contains the individual lines of text in the memo control.
	MaxLength (see page 2671)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 2672)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2672)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2673)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2673)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2673)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2674)	Determines where a control looks for its font information.
	ParentShowHint (see page 2674)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2675)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2680)	Determines whether the user can change the text of the edit control.

	ScrollBars (see page 2680)	Determines whether the memo control has scroll bars.
	ShowHint (see page 2682)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2684)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2684)	Determines if the user can tab to a control.
	Visible (see page 2685)	Determines whether the component appears onscreen.
	WantReturns (see page 2686)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2687)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2687)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

29.1.22.2 TMemo Properties

The properties of the TMemo class are listed here.

Published Properties

	Name	Description
	Align (see page 2654)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2657)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2658)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2658)	Specifies which edges of the control are beveled.
	BevelInner (see page 2659)	Specifies the cut of the inner bevel.
	BevelKind (see page 2659)	Specifies the control's bevel style.
	BevelOuter (see page 2660)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2660)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2660)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2661)	Determines the case of the text within the edit control.
	Color (see page 2661)	Specifies the background color of the control.
	Constraints (see page 2662)	Specifies the size constraints for the control.
	Ctl3D (see page 2663)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2663)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2665)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2665)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2666)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2667)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2668)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2669)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2669)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2670)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 2670)	Contains the individual lines of text in the memo control.
	MaxLength (see page 2671)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 2672)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2672)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2673)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2673)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2673)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2674)	Determines where a control looks for its font information.

	ParentShowHint (see page 2674)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2675)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2680)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 2680)	Determines whether the memo control has scroll bars.
	ShowHint (see page 2682)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2684)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2684)	Determines if the user can tab to a control.
	Visible (see page 2685)	Determines whether the component appears onscreen.
	WantReturns (see page 2686)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2687)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2687)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

29.1.22.2.1 TMemo.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
```

```
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
```

```
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.22.2.2 TMemo.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.

taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2279)

Text

Displaying and Editing Fields in an Edit Box

29.1.22.2.3 TMemo.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.22.2.4 TMemo.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

29.1.22.2.5 TMemo.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

29.1.22.2.6 TMemo.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

29.1.22.2.7 TMemo.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

29.1.22.2.8 TMemo.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.22.2.9 TMemo.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

29.1.22.2.10 TMemo.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

29.1.22.2.11 TMemo.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
```

```
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.22.2.12 TMemo.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.22.2.13 TMemo.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.22.2.14 TMemo.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
```

```
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.22.2.15 TMemo.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.22.2.16 TMemo.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.22.2.17 TMemo.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.22.2.18 TMemo.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.22.2.19 TMemo.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.22.2.20 TMemo.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2289)

SelStart (see page 2293)

SelText (see page 2296)

SetSelTextBuf (see page 2281)

29.1.22.2.21 TMemo.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

29.1.22.2.22 TMemo.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

29.1.22.2.23 TMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with

the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings

SelText (see page 2296)

SetLines (see page 2367)

Text

WordWrap (see page 2372)

29.1.22.2.24 TMemo.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

29.1.22.2.25 TMemo.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

29.1.22.2.26 TMemo.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.22.2.27 TMemo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.22.2.28 TMemo.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.22.2.29 TMemo.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.22.2.30 TMemo.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.22.2.31 TMemo.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.22.2.32 TMemo.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
```

```
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
```

```
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.22.2.33 TMemo.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2268)

GetSelTextBuf (see page 2275)

29.1.22.2.34 TMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
```

```
    // move text cursor to top of editend;
    RichEdit1.SelStart := 0;
  end;

  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    RichEdit1.HideScrollbars := CheckBox1.Checked;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  const
    ScrollBarA: array[0..3] of TScrollStyle = (
      ssBoth,
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

SetScrollBars (see page 2367)

WordWrap (see page 2372)

Displaying and Editing Text in a Memo Control

29.1.22.2.35 TMemo.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
```

```
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.22.2.36 TMemo.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.22.2.37 TMemo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.22.2.38 TMemo.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
```

```
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.22.2.39 TMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text

WantTabs (see page 2372)

WordWrap (see page 2372)

29.1.22.2.40 TMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text

WantReturns (see page 2372)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

29.1.22.2.41 TMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
```

```
    // move text cursor to top of editend;
    RichEdit1.SelStart := 0;
  end;

  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    RichEdit1.HideScrollbars := CheckBox1.Checked;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  const
    ScrollBarA: array[0..3] of TScrollStyle = (
      ssBoth,
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

Lines (see page 2368)

ScrollBars (see page 2369)

Text

WantReturns (see page 2372)

Displaying and Editing Text in a Memo Control

29.1.22.3 TMemo Events

The events of the TMemo class are listed here.

Published Events

	Name	Description
	OnChange (see page 2690)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2690)	Occurs when the user clicks the control.

	OnContextPopup (see page 2692)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2693)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2693)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2696)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2699)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2700)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2701)	Occurs when a control receives the input focus.
	OnExit (see page 2703)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2706)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2710)	Occurs when a key is pressed.
	OnKeyUp (see page 2711)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2713)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2714)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2716)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2716)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2716)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2719)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2721)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2721)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.22.3.1 TMemo.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2266)

Text

29.1.22.3.2 TMemo.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.22.3.3 TMemo.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.22.3.4 TMemo.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.22.3.5 TMemo.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.22.3.6 TMemo.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor

property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
```

```
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.22.3.7 TMemo.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.22.3.8 TMemo.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.22.3.9 TMemo.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
```

```
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.22.3.10 TMemo.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;
```

```
    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

29.1.22.3.11 TMemo.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)
```

```
{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
```

```
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
```

```
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.22.3.12 TMemo.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
```

```
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.22.3.13 TMemo.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.22.3.14 TMemo.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.22.3.15 TMemo.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.22.3.16 TMemo.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.22.3.17 TMemo.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.22.3.18 TMemo.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.22.3.19 TMemo.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.22.3.20 TMemo.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.22.3.21 TMemo.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.23 TPushButtonActionLink Class

Class Hierarchy

Pascal

```
TPushButtonActionLink = class(TButtonActionLink);
```

C++

```
class TPushButtonActionLink : public TButtonActionLink;
```

File

StdCtrls

Description

This is class StdCtrls.TPushButtonActionLink.

29.1.23.1 TPushButtonActionLink Members

The following tables list the members exposed by TPushButtonActionLink.

Protected Fields

	Name	Description
	FClient (see page 2031)	This is FClient, a member of class TButtonActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2032)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 2032)	Indicates whether the checked state for the action and the client are linked.
	IsImageIndexLinked (see page 2723)	Indicates whether the image index for the action and the client are linked.
	SetChecked (see page 2032)	Sets the checked state of a button control if the client is linked to the action for this property.
	SetImageIndex (see page 2724)	Introduces an interface for setting the image index for a client.

29.1.23.2 TPushButtonActionLink Methods

The methods of the TPushButtonActionLink class are listed here.

Protected Methods

	Name	Description
	IsImageIndexLinked (see page 2723)	Indicates whether the image index for the action and the client are linked.
	SetImageIndex (see page 2724)	Introduces an interface for setting the image index for a client.

29.1.23.2.1 TPushButtonActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex

ImageIndex

SetImageIndex

Action

29.1.23.2.2 TPushButtonActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex

ImageIndex

Action

IsImageIndexLinked

29.1.24 TRadioButton Class

TRadioButton is a wrapper for a system radio button.

Class Hierarchy

Pascal

```
TRadioButton = class(TButtonControl);
```

C++

```
class TRadioButton : public TButtonControl;
```

File

StdCtrls

Description

Use TRadioButton to add a radio button to a form. Radio buttons present a set of mutually exclusive options to the user- that is, only one radio button in a set can be selected at a time. When the user selects a radio button, the previously selected radio button becomes unselected. Radio buttons are frequently grouped in a radio group box (TGroupBox (see page 2451)). Add the group box to the form first, then get the radio buttons from the Component palette and put them into the group box. Or you can add a TRadioGroup, a group that already has radio buttons in it, so that you do not have to get the buttons from the Tool

Palette.

By default, all radio buttons that are directly contained in the same windowed control container, such as a TRadioGroup or TPanel, are grouped. For example, two radio buttons on a form can be checked at the same time only if they are contained in separate containers, such as two different group boxes.

See Also

TCheckBox (see page 2040)

TPanel

TRadioGroup

Buttons and Similar Controls

Group Boxes and Radio Groups

Radio Buttons

29.1.24.1 TRadioButton Members

The following tables list the members exposed by TRadioButton.

Protected Methods

	Name	Description
	ActionChange (see page 2034)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2727)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2728)	Creates the underlying window.
	GetActionLinkClass (see page 2035)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2728)	Returns whether the button control is checked.
	SetChecked (see page 2729)	Sets the Checked property.
	WndProc (see page 2036)	Handles Windows messages.

Protected Properties

	Name	Description
	ClicksDisabled (see page 2039)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Create (see page 2726)	Creates an instance of TRadioButton (see page 2724).
	GetControlsAlignment (see page 2728)	Returns the alignment of any text in the control.

Published Events

	Name	Description
	OnClick (see page 2758)	Occurs when the user clicks the control.
	OnContextPopup (see page 2760)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2761)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2761)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2764)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2767)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2768)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2769)	Occurs when a control receives the input focus.
	OnExit (see page 2771)	Occurs when the input focus shifts away from one control to another.

	OnKeyDown (see page 2774)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2777)	Occurs when a key is pressed.
	OnKeyUp (see page 2779)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2780)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2781)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2784)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2784)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2784)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2786)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2789)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2789)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2730)	Designates the action associated with the control.
	Align (see page 2730)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2734)	Determines the position of the caption relative to the radio button.
	Anchors (see page 2734)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2735)	Specifies the bi-directional mode for the control.
	Caption (see page 2735)	Specifies a text string that identifies the control to the user.
	Checked (see page 2738)	Determines whether the option represented by the radio button is selected.
	Color (see page 2738)	Specifies the background color of the control.
	Constraints (see page 2739)	Specifies the size constraints for the control.
	Ctl3D (see page 2740)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2740)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2742)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2742)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2743)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2744)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2745)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2746)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2746)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2747)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2747)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2747)	Determines where a control looks for its font information.
	ParentShowHint (see page 2748)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2748)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2753)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2755)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2755)	Determines if the user can tab to a control.
	Visible (see page 2756)	Determines whether the component appears onscreen.
	WordWrap (see page 2757)	Specifies whether the button text wraps to fit the width of the control.

29.1.24.2 TRadioButton.Create Constructor

Creates an instance of TRadioButton (see page 2724).

Pascal

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

C++

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

Description

Call Create to instantiate a radio button at runtime. Radio buttons added to forms at design time are created automatically.

AOwner is the component, typically a form, that becomes the value of the Owner property. The Owner is responsible for freeing the radio button object.

See Also

Owner

29.1.24.3 TRadioButton Methods

The methods of the TRadioButton class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2727)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2728)	Creates the underlying window.
	GetChecked (see page 2728)	Returns whether the button control is checked.
	SetChecked (see page 2729)	Sets the Checked property.

Public Methods

	Name	Description
	GetControlsAlignment (see page 2728)	Returns the alignment of any text in the control.

29.1.24.3.1 TRadioButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

29.1.24.3.2 TRadioButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

29.1.24.3.3 TRadioButton.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked property.

See Also

Checked (see page 2037)

29.1.24.3.4 TRadioButton.GetControlsAlignment Method

Returns the alignment of any text in the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the orientation of text in controls that do not have an Alignment (see page 2734) property.

GetControlsAlignment overrides the inherited method, returning taRightJustify if UseRightToLeftAlignment returns false. If UseRightToLeftAlignment returns true, GetControlsAlignment returns the opposite of the Alignment (see page 2734) property.

29.1.24.3.5 TRadioButton.SetChecked Method

Sets the Checked property.

Pascal

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

C++

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

Description

SetChecked is the write implementer for the Checked property.

See Also

Checked (see page 2037)

29.1.24.4 TRadioButton Properties

The properties of the TRadioButton class are listed here.

Published Properties

	Name	Description
	Action (see page 2730)	Designates the action associated with the control.
	Align (see page 2730)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2734)	Determines the position of the caption relative to the radio button.
	Anchors (see page 2734)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2735)	Specifies the bi-directional mode for the control.
	Caption (see page 2735)	Specifies a text string that identifies the control to the user.
	Checked (see page 2738)	Determines whether the option represented by the radio button is selected.
	Color (see page 2738)	Specifies the background color of the control.
	Constraints (see page 2739)	Specifies the size constraints for the control.
	Ctl3D (see page 2740)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2740)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2742)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2742)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2743)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2744)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2745)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2746)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2746)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2747)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2747)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

	ParentFont (see page 2747)	Determines where a control looks for its font information.
	ParentShowHint (see page 2748)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2748)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2753)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2755)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2755)	Determines if the user can tab to a control.
	Visible (see page 2756)	Determines whether the component appears onscreen.
	WordWrap (see page 2757)	Specifies whether the button text wraps to fit the width of the control.

29.1.24.4.1 TRadioButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

29.1.24.4.2 TRadioButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
```

```
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
```

```
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.24.4.3 TRadioButton.Alignment Property

Determines the position of the caption relative to the radio button.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

Set Alignment to determine whether the caption of the radio button should appear to the left of the radio button (taLeftJustify) or to the right of the radio button (taRightJustify).

Note: Do not confuse the Alignment property with the alignment of the caption in the text area of the radio button. The caption always appears left aligned. Alignment only controls on which side of the button the text area appears.

See Also

Caption

29.1.24.4.4 TRadioButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.24.4.5 TRadioButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.24.4.6 TRadioButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
```

```
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

29.1.24.4.7 TRadioButton.Checked Property

Determines whether the option represented by the radio button is selected.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Read (see page 3727) Checked to determine whether the radio button is selected. Set Checked to true to select the radio button and deselect all other radio buttons in the same container. Set Checked to false to deselect the radio button, leaving no radio button in the group selected.

See Also

TRadioGroup

29.1.24.4.8 TRadioButton.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.24.4.9 TRadioButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.24.4.10 TRadioButton.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.24.4.11 TRadioButton.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.24.4.12 TRadioButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.24.4.13 TRadioButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.24.4.14 TRadioButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
```

```
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.24.4.15 TRadioButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.24.4.16 TRadioButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.24.4.17 TRadioButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.24.4.18 TRadioButton.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.24.4.19 TRadioButton.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.24.4.20 TRadioButton.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.24.4.21 TRadioButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.24.4.22 TRadioButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.24.4.23 TRadioButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.24.4.24 TRadioButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.24.4.25 TRadioButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.24.4.26 TRadioButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.24.4.27 TRadioButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.24.4.28 TRadioButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

29.1.24.5 TRadioButton Events

The events of the TRadioButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 2758)	Occurs when the user clicks the control.
	OnContextPopup (see page 2760)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2761)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2761)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2764)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2767)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2768)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2769)	Occurs when a control receives the input focus.
	OnExit (see page 2771)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2774)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2777)	Occurs when a key is pressed.
	OnKeyUp (see page 2779)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2780)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2781)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2784)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2784)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2784)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2786)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2789)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2789)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.24.5.1 TRadioButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.24.5.2 TRadioButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.24.5.3 TRadioButton.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.24.5.4 TRadioButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.24.5.5 TRadioButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
```

```
  label control to be dropped on a panel control and stops the
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  }
  type
    TForm1 = class(TForm)
      Panel1: TPanel;
      Label1: TLabel;
      procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
        State: TDragState; var Accept: Boolean);
      procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.24.5.6 TRadioButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.24.5.7 TRadioButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.24.5.8 TRadioButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.24.5.9 TRadioButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
```

```
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

29.1.24.5.10 TRadioButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
```

```
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.24.5.11 TRadioButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.24.5.12 TRadioButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.24.5.13 TRadioButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.24.5.14 TRadioButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.24.5.15 TRadioButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.24.5.16 TRadioButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.24.5.17 TRadioButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.24.5.18 TRadioButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.24.5.19 TRadioButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.24.5.20 TRadioButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

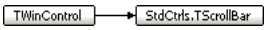
OnDragOver

OnDragDrop

29.1.25 TScrollBar Class

TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.

Class Hierarchy



Pascal

```
TScrollBar = class(TWinControl);
```

C++

```
class TScrollBar : public TWinControl;
```

File

StdCtrls

Description

Use TScrollBar to add a free-standing scroll bar to a form. Many controls have properties that add scroll bars which are an integral part of the control (TControlScrollBar objects). TScrollBar allows controls that do not have integrated scroll bars or groupings of controls to be scrolled when the user manipulates the TScrollBar object.

When the bi-directional mode of a scroll bar is changed, its coordinate system is reversed. Position (see page 2820) zero in left-to-right mode is the left side, and position zero in right-to-left mode is the right side.

See Also

TScrollingWinControl

TTrackBar

TUpDown

Scroll Bars

Specialized Input Controls

29.1.25.1 TScrollBar Members

The following tables list the members exposed by TScrollBar.

Protected Methods

	Name	Description
	Change (see page 2793)	This is Change, a member of class TScrollBar.
	CreateParams (see page 2793)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2793)	Creates the underlying window.
	Scroll (see page 2794)	This is Scroll, a member of class TScrollBar.

Public Methods

	Name	Description
	Create (see page 2792)	Creates and initializes an instance of TScrollBar (see page 2790).
	SetParams (see page 2794)	Sets the Position (see page 2820), Max (see page 2808), and Min (see page 2810) property values all at once.

Published Events

	Name	Description
	OnChange (see page 2829)	Occurs immediately after the value of Position (see page 2820) changes.
	OnContextPopup (see page 2829)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2830)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2833)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2836)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2836)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2837)	Occurs when a control receives the input focus.
	OnExit (see page 2840)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2842)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2846)	Occurs when a key is pressed.
	OnKeyUp (see page 2847)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 2849)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2849)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 2849)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 2851)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2852)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2795)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2799)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2799)	Specifies the bi-directional mode for the control.
	Constraints (see page 2800)	Specifies the size constraints for the control.
	Ctl3D (see page 2800)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2801)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2802)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2803)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2803)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2805)	Controls whether the control responds to mouse, keyboard, and timer events.

	Kind (see page 2805)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 2806)	Determines how much Position (see page 2820) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Max (see page 2808)	Specifies the maximum position represented by the scroll bar.
	Min (see page 2810)	Specifies the minimum position represented by the scroll bar.
	PageSize (see page 2812)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 2814)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 2814)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2814)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 2815)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2815)	Identifies the pop-up menu associated with the control.
	Position (see page 2820)	Indicates the current position of the scroll bar.
	ShowHint (see page 2822)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 2824)	Determines how much Position (see page 2820) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 2826)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2826)	Determines if the user can tab to a control.
	Visible (see page 2827)	Determines whether the component appears onscreen.

29.1.25.2 TScrollBar.Create Constructor

Creates and initializes an instance of TScrollBar (see page 2790).

Pascal

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

C++

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

Description

Call Create to create an instance of TScrollBar (see page 2790) at runtime. Scroll (see page 2794) bars placed on a form at design time are created automatically.

AOwner is the component that is responsible for freeing the scroll bar. This becomes the value of the Owner property.

See Also

Owner

29.1.25.3 TScrollBar Methods

The methods of the TScrollBar class are listed here.

Protected Methods

	Name	Description
	Change (see page 2793)	This is Change, a member of class TScrollBar.
V	CreateParams (see page 2793)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 2793)	Creates the underlying window.
	Scroll (see page 2794)	This is Scroll, a member of class TScrollBar.

Public Methods

	Name	Description
	SetParams (see page 2794)	Sets the Position (see page 2820), Max (see page 2808), and Min (see page 2810) property values all at once.

29.1.25.3.1 TScrollBar.Change Method

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

This is Change, a member of class TScrollBar.

29.1.25.3.2 TScrollBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

29.1.25.3.3 TScrollBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

29.1.25.3.4 TScrollBar.Scroll Method

Pascal

```
procedure Scroll(ScrollCode: TScrollCode; var ScrollPos: Integer); dynamic;
```

C++

```
__fastcall Scroll(TScrollCode ScrollCode, int ScrollPos);
```

Description

This is Scroll, a member of class TScrollBar.

29.1.25.3.5 TScrollBar.SetParams Method

Sets the Position (see page 2820), Max (see page 2808), and Min (see page 2810) property values all at once.

Pascal

```
procedure SetParams(APosition: Integer; AMin: Integer; AMax: Integer);
```

C++

```
__fastcall SetParams(int APosition, int AMin, int AMax);
```

Description

Use the SetParams method to set the Position (see page 2820), Min (see page 2810), and Max (see page 2808) property values all at once, preventing repaints that would occur if each property value were set separately.

See Also

Max (see page 2808)

Min (see page 2810)

Position (see page 2820)

29.1.25.4 TScrollBar Properties

The properties of the TScrollBar class are listed here.

Published Properties

	Name	Description
	Align (see page 2795)	Determines how the control aligns within its container (parent control).

	Anchors (see page 2799)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2799)	Specifies the bi-directional mode for the control.
	Constraints (see page 2800)	Specifies the size constraints for the control.
	Ctl3D (see page 2800)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2801)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2802)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2803)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2803)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2805)	Controls whether the control responds to mouse, keyboard, and timer events.
	Kind (see page 2805)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 2806)	Determines how much Position (see page 2820) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Max (see page 2808)	Specifies the maximum position represented by the scroll bar.
	Min (see page 2810)	Specifies the minimum position represented by the scroll bar.
	PageSize (see page 2812)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 2814)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 2814)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2814)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 2815)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2815)	Identifies the pop-up menu associated with the control.
	Position (see page 2820)	Indicates the current position of the scroll bar.
	ShowHint (see page 2822)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 2824)	Determines how much Position (see page 2820) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 2826)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2826)	Determines if the user can tab to a control.
	Visible (see page 2827)	Determines whether the component appears onscreen.

29.1.25.4.1 TScrollBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.25.4.2 TScrollBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.25.4.3 TScrollBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.25.4.4 TScrollBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.25.4.5 TScrollBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

29.1.25.4.6 TScrollBar.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.25.4.7 TScrollBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.25.4.8 TScrollBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.25.4.9 TScrollBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.25.4.10 TScrollBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.25.4.11 TScrollBar.Kind Property

Specifies whether the scroll bar is horizontal or vertical.

Pascal

```
property Kind: TScrollBarKind;
```

C++

```
__property TScrollBarKind Kind;
```

Description

Set Kind to indicate the orientation of the scroll bar. These are the possible values:

Value	Meaning
sbHorizontal	Scroll (see page 2794) bar is horizontal
sbVertical	Scroll (see page 2794) bar is vertical

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

29.1.25.4.12 TScrollBar.LargeChange Property

Determines how much Position (see page 2820) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.

Pascal

```
property LargeChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc LargeChange;
```

Description

Set LargeChange to specify how much to change the value of Position (see page 2820) when the user clicks on the side of the thumb tab (that is, on the thumb tab track, but not on the thumb tab itself). The Max (see page 2808) and Min (see page 2810) properties define the total range over which Position (see page 2820) can vary.

For example, if Max (see page 2808) is 30000 and Min (see page 2810) is 0, the scroll box can assume 30,000 positions. If the LargeChange property setting is 10000 and the Position (see page 2820) property is 0, the user can click the scroll bar track three times after the thumb tab before it moves all the way to the end of the scroll bar (30000 / 10000 = 3).

LargeChange should correspond to the portion of this range that the user sees in one page or screen, expressed in the logical units used by the Max (see page 2808), Min (see page 2810), and Position (see page 2820) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
```

```
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

Max (see page 2808)

Min (see page 2810)

Position (see page 2820)

SmallChange (see page 2824)

PageSize (see page 2812)

29.1.25.4.13 TScrollBar.Max Property

Specifies the maximum position represented by the scroll bar.

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Set Max to the maximum value the Position (see page 2820) property can take. The Max and Min (see page 2810) properties define the total range over which Position (see page 2820) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
```

```
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
```

```
    { set up the scrollbar to reflect the structure of the page control}
    with ScrollBar1 do
    begin
      Max := PageControl1.PageCount - 1;
      Min := 0;
      SmallChange := 1;
      { set LargeChange to a single row of tabs }
      LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
    end;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    with Sender as TPageControl do
      ScrollBar1.Position := ActivePage.PageIndex;
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
  end;
```

See Also

LargeChange (see page 2806)

Min (see page 2810)

Position (see page 2820)

SetParams (see page 2794)

SmallChange (see page 2824)

29.1.25.4.14 TScrollBar.Min Property

Specifies the minimum position represented by the scroll bar.

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Set Min to the minimum value the Position (see page 2820) property can take. The Min and Max (see page 2808) properties define the total range over which Position (see page 2820) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
```

```
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 2806)

Max (see page 2808)

Position (see page 2820)

SetParams (see page 2794)

SmallChange (see page 2824)

29.1.25.4.15 TScrollBar.PageSize Property

Specifies the size of the thumb tab.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

PageSize is the size of the thumb tab, measured in the same units as Position (see page 2820), Min (see page 2810), and Max (see page 2808) (not pixels).

C++ Examples:

```
/*
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
*/
void __fastcall SetThumbTab()
{
  // Get the minimum (default) size of the thumb tab
  int MinHeight = GetSystemMetrics(SM_CYVTHUMB);
  // Get the size of the scrollbar track
  int TrackHeight = Form1->ScrollBar1->ClientHeight - 2*GetSystemMetrics(SM_CYVSCROLL);
  // set PageSize to represent the visible portion of the scrolling range:
  Form1->ScrollBar1->PageSize = TrackHeight/(Form1->ScrollBar1->Max - Form1->ScrollBar1->Min +
1);
  // Don't let the thumb tab get too small
  if (Form1->ScrollBar1->PageSize < MinHeight)
  Form1->ScrollBar1->PageSize = MinHeight;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
```

```
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1EndDrag(TObject *Sender, TObject *Target,
      int X, int Y)
{
 SetThumbTab();
}
```

Delphi Examples:

```
{
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
}
procedure SetThumbTab(wincontrol: TWinControl);
var
  TrackHeight: Integer; { the size of the scroll bar track }
  MinHeight: Integer; { the default size of the thumb tab }
begin
  MinHeight := GetSystemMetrics(SM_CYVTHUMB); { save default size }
  with Form1.ScrollBar1 do
  begin
    TrackHeight := wincontrol.ClientHeight - 2 * GetSystemMetrics(SM_CYVSCROLL);
    PageSize := TrackHeight div (Max - Min + 1);
    if PageSize < MinHeight then PageSize := MinHeight;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  SetThumbTab(RichEdit1);
end;
procedure TForm1.RichEdit1Change(Sender: TObject);
begin
  SetThumbTab(RichEdit1);
end;
```

See Also

Min (see page 2810)

Max (see page 2808)

ClientHeight

LargeChange (see page 2806)

29.1.25.4.16 TScrollBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.25.4.17 TScrollBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

29.1.25.4.18 TScrollBar.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of

the parent's DoubleBuffered property.

29.1.25.4.19 TScrollBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.25.4.20 TScrollBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
```

```
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.25.4.21 TScrollBar.Position Property

Indicates the current position of the scroll bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Read (see page 3727) Position to determine the current position of the thumb tab. This value can be used to determine how to scroll any components controlled by the scroll bar. When the user scrolls the scroll bar, the value of Position changes. Set Position to programmatically move the thumb tab of the scroll bar.

The number of possible positions on the scroll bar is determined by the difference between the Max (see page 2808) property and the Min (see page 2810) property. If Position has the same value as Min (see page 2810), the thumb tab appears at the far left of a horizontal scroll bar or the top of a vertical scroll bar. If Position has the same value as Max (see page 2808), the thumb tab appears at the far right of a horizontal scroll bar or the bottom of a vertical scroll bar.

C++ Examples:

```
/*
```

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
```

```
    PageControl1.ActivePage := PageControl1.Pages[0];
    for i := 0 to PageControl1.PageCount - 1 do
      PageControl1.Pages[i].Brush.Color:= colorPalette[i];
    { set up the scrollbar to reflect the structure of the page control}
    with ScrollBar1 do
    begin
      Max := PageControl1.PageCount - 1;
      Min := 0;
      SmallChange := 1;
      { set LargeChange to a single row of tabs }
      LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
    end;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    with Sender as TPageControl do
      ScrollBar1.Position := ActivePage.PageIndex;
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
  end;
```

See Also

Max (see page 2808)

Min (see page 2810)

SetParams (see page 2794)

29.1.25.4.22 TScrollBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.25.4.23 TScrollBar.SmallChange Property

Determines how much Position (see page 2820) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.

Pascal

```
property SmallChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc SmallChange;
```

Description

Set SmallChange to specify how much to change the value of Position (see page 2820) when the user clicks on the arrow buttons. The Max (see page 2808) and Min (see page 2810) properties define the total range over which Position (see page 2820) can vary.

For example, if Max (see page 2808) is 100 and Min (see page 2810) is 0, the scroll box can assume 100 positions. If the SmallChange property setting is 5 and the Position (see page 2820) property is 0, the user can click the down or right arrow button 20 times before the thumb tab is moved all the way to the end of the scroll bar.

SmallChange should correspond to the portion of this range that the user sees as one line or one item, expressed in the logical units used by the Max (see page 2808), Min (see page 2810), and Position (see page 2820) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}
```

```
void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 2806)

Max (see page 2808)

Min (see page 2810)

Position (see page 2820)

29.1.25.4.24 TScrollBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.25.4.25 TScrollBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.25.4.26 TScrollBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.25.5 TScrollBar Events

The events of the TScrollBar class are listed here.

Published Events

	Name	Description
	OnChange (see page 2829)	Occurs immediately after the value of Position (see page 2820) changes.
	OnContextPopup (see page 2829)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2830)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2833)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2836)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2836)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2837)	Occurs when a control receives the input focus.
	OnExit (see page 2840)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2842)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2846)	Occurs when a key is pressed.
	OnKeyUp (see page 2847)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 2849)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2849)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 2849)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 2851)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2852)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.25.5.1 TScrollBar.OnChange Event

Occurs immediately after the value of Position (see page 2820) changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3789) an OnChange event handler to control another object or group of objects using TScrollBar (see page 2790). Adjust the objects controlled by the scroll bar to reflect the new value of Position (see page 2820).

OnChange occurs whether the value of Position (see page 2820) is changed by the user or programmatically. If the value of Position (see page 2820) is changed by the user, OnChange occurs immediately after the OnScroll (see page 2849) event.

See Also

OnScroll (see page 2849)

Position (see page 2820)

29.1.25.5.2 TScrollBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.25.5.3 TScrollBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.25.5.4 TScrollBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.25.5.5 TScrollBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.25.5.6 TScrollBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.25.5.7 TScrollBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

29.1.25.5.8 TScrollBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
```

```
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

29.1.25.5.9 TScrollBar.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
```

```
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
```

```
  Behavior is dependent on keyboard driver(USB, serial, etc).
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

29.1.25.5.10 TScrollBar.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

29.1.25.5.11 TScrollBar.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

29.1.25.5.12 TScrollBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.25.5.13 TScrollBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.25.5.14 TScrollBar.OnScroll Event

Occurs when the user scrolls the scroll bar with the mouse or keyboard.

Pascal

```
property OnScroll: TScrollEvent;
```

C++

```
__property TScrollEvent OnScroll;
```

Description

Write (see page 3789) an OnScroll to alter the way the scrollbar adjusts its position when the user scrolls the scroll bar.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 2806)

OnChange (see page 2829)

Position (see page 2820)

SmallChange (see page 2824)

29.1.25.5.15 TScrollBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.25.5.16 TScrollBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.1.26 TStaticText Class

TStaticText is a windowed control that displays text on a form.

Class Hierarchy

Pascal

```
TStaticText = class(TCustomStaticText);
```

C++

```
class TStaticText : public TCustomStaticText;
```

File

StdCtrls

Description

The TStaticText component functions like TLabel (see page 2522), except that it descends from TWinControl and therefore has a window handle. Use TStaticText instead of TLabel (see page 2522) when the component's accelerator key must belong to a windowed control—for example, on an ActiveX property page.

See Also

TCustomStaticText (see page 2375)

TLabel (see page 2522)

TWinControl

Labels

Display Controls

29.1.26.1 TStaticText Members

The following tables list the members exposed by TStaticText.

Protected Methods

	Name	Description
	CreateParams (see page 2376)	Initializes a window-creation parameter data structure.
	Loaded (see page 2377)	Initializes the control after it is loaded from a stream.
	Notification (see page 2377)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2378)	This is SetAutoSize, a member of class TCustomStaticText.

Public Methods

	Name	Description
	Create (see page 2376)	Creates and initializes a TCustomStaticText (see page 2375) instance.

Published Events

	Name	Description
	OnClick (see page 2885)	Occurs when the user clicks the control.
	OnContextPopup (see page 2887)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 2887)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2888)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2891)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2894)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2894)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2895)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2896)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2899)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2899)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2899)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2901)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2904)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2904)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2855)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2859)	Determines how text is aligned within the component.
	Anchors (see page 2859)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2860)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 2860)	Specifies which edges of the control are beveled.
	BevelInner (see page 2861)	Specifies the cut of the inner bevel.
	BevelKind (see page 2861)	Specifies the control's bevel style.
	BevelOuter (see page 2862)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2862)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2862)	Determines the appearance of the control's border.
	Caption (see page 2863)	Specifies a text string that identifies the control to the user.
	Color (see page 2865)	Specifies the background color of the control.
	Constraints (see page 2866)	Specifies the size constraints for the control.
	DoubleBuffered (see page 2867)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2868)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2869)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2869)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2871)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2871)	Links the component with another windowed control.
	Font (see page 2872)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2873)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2873)	Determines where a control looks for its color information.
	ParentDoubleBuffered (see page 2874)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2874)	Determines where a control looks for its font information.
	ParentShowHint (see page 2874)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2875)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2880)	Enables accelerator characters.
	ShowHint (see page 2880)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2882)	Indicates the position of the control in its parent's tab order.

	TabStop (see page 2882)	Determines if the user can tab to a control.
	Transparent (see page 2883)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 2883)	Determines whether the component appears onscreen.

29.1.26.2 TStaticText Properties

The properties of the TStaticText class are listed here.

Published Properties

	Name	Description
	Align (see page 2855)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2859)	Determines how text is aligned within the component.
	Anchors (see page 2859)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2860)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 2860)	Specifies which edges of the control are beveled.
	BevelInner (see page 2861)	Specifies the cut of the inner bevel.
	BevelKind (see page 2861)	Specifies the control's bevel style.
	BevelOuter (see page 2862)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2862)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2862)	Determines the appearance of the control's border.
	Caption (see page 2863)	Specifies a text string that identifies the control to the user.
	Color (see page 2865)	Specifies the background color of the control.
	Constraints (see page 2866)	Specifies the size constraints for the control.
	DoubleBuffered (see page 2867)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2868)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2869)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2869)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2871)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2871)	Links the component with another windowed control.
	Font (see page 2872)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2873)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2873)	Determines where a control looks for its color information.
	ParentDoubleBuffered (see page 2874)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2874)	Determines where a control looks for its font information.
	ParentShowHint (see page 2874)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2875)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2880)	Enables accelerator characters.
	ShowHint (see page 2880)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2882)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2882)	Determines if the user can tab to a control.
	Transparent (see page 2883)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 2883)	Determines whether the component appears onscreen.

29.1.26.2.1 TStaticText.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
```

```
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
```

```
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

29.1.26.2.2 TStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font

Caption

29.1.26.2.3 TStaticText.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

29.1.26.2.4 TStaticText.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints

CanAutoSize

29.1.26.2.5 TStaticText.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

29.1.26.2.6 TStaticText.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

29.1.26.2.7 TStaticText.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

29.1.26.2.8 TStaticText.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

29.1.26.2.9 TStaticText.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

29.1.26.2.10 TStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

29.1.26.2.11 TStaticText.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
```

```
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

29.1.26.2.12 TStaticText.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

29.1.26.2.13 TStaticText.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

29.1.26.2.14 TStaticText.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

29.1.26.2.15 TStaticText.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

29.1.26.2.16 TStaticText.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

29.1.26.2.17 TStaticText.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

29.1.26.2.18 TStaticText.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

29.1.26.2.19 TStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption

FocusControl (see page 2313)

TWinControl

29.1.26.2.20 TStaticText.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

29.1.26.2.21 TStaticText.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

29.1.26.2.22 TStaticText.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

29.1.26.2.23 TStaticText.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

29.1.26.2.24 TStaticText.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

29.1.26.2.25 TStaticText.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

29.1.26.2.26 TStaticText.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

29.1.26.2.27 TStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption

29.1.26.2.28 TStaticText.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

29.1.26.2.29 TStaticText.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

29.1.26.2.30 TStaticText.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

29.1.26.2.31 TStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

29.1.26.2.32 TStaticText.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

29.1.26.3 TStaticText Events

The events of the TStaticText class are listed here.

Published Events

	Name	Description
	OnClick (see page 2885)	Occurs when the user clicks the control.

	OnContextPopup (see page 2887)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2887)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2888)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2891)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2894)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2894)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2895)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2896)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2899)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2899)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2899)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2901)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2904)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2904)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

29.1.26.3.1 TStaticText.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

29.1.26.3.2 TStaticText.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

29.1.26.3.3 TStaticText.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

29.1.26.3.4 TStaticText.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

29.1.26.3.5 TStaticText.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}
```

```
void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
```

```
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

29.1.26.3.6 TStaticText.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

29.1.26.3.7 TStaticText.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

29.1.26.3.8 TStaticText.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse

button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.26.3.9 TStaticText.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area

of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

29.1.26.3.10 TStaticText.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

29.1.26.3.11 TStaticText.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

29.1.26.3.12 TStaticText.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

29.1.26.3.13 TStaticText.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

29.1.26.3.14 TStaticText.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

29.1.26.3.15 TStaticText.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

29.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCheckBoxState (see page 2906)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 2907)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 1959) unit.
	TEditCharCase (see page 2907)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 2908)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TImageAlignment (see page 2909)	Alignment of image on TButton (see page 1961).
	TListBoxStyle (see page 2910)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 2911)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 2912)	Identifies a set of scrollbars by their orientation.
	TStaticBorderStyle (see page 2913)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 2914)	TTextLayout indicates where text appears within the client area of a control.

Records

	Name	Description
	TSelection (see page 2913)	TSelection definese the start and position of a selection.

29.2.1 StdCtrls.TCheckBoxState Enumeration

TCheckBoxState indicates the state of a check box control.

Pascal

```
TCheckBoxState = (
  cbUnchecked,
  cbChecked,
  cbGrayed
);
```

C++

```
enum TCheckBoxState {
  cbUnchecked,
  cbChecked,
  cbGrayed
};
```

File

StdCtrls

Description

Following are the possible values of TCheckBoxState:

Value	Meaning
cbUnchecked	The check box has no check mark, indicating that the user has not selected the option.
cbChecked	The check box has a check mark in it, indicating that the user has selected the option.

cbGrayed	The check box has a check mark in it, but it is grayed.

29.2.2 StdCtrls.TComboBoxStyle Enumeration

TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 1959) unit.

Pascal

```
TComboBoxStyle = (
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
);
```

C++

```
enum TComboBoxStyle {
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
};
```

File

StdCtrls

Description

These are the values in the TComboBoxStyle type:

Value	Meaning
csDropDown	Creates a drop-down list with an edit box for manually entered text. All items are strings of the same height.
csSimple	Creates an edit box with a fixed list (list box) underneath. The length of the list is determined by the Height of the combo box.
csDropDownList	Creates a drop-down list with no edit box; the user cannot enter text manually. All items are strings of the same height.
csOwnerDrawFixed	Creates an owner-draw drop-down list with no edit box. Each item in the list is the height specified by the ItemHeight property.
csOwnerDrawVariable	Creates an owner-draw drop-down list with no edit box. List items can have varying heights.

29.2.3 StdCtrls.TEditCharCase Enumeration

TEditCharCase indicates how a control forces the case of characters, if at all.

Pascal

```
TEditCharCase = (
  ecNormal,
  ecUpperCase,
```

```
    ecLowerCase
  );
```

C++

```
  enum TEditCharCase {
    ecNormal,
    ecUpperCase,
    ecLowerCase
  };
```

File

StdCtrls

Description

TEditCharCase specifies what case is given to text typed into a control. It can have one of the following values:

Value	Meaning
ecLowerCase	The text is converted to lowercase.
ecNormal	The text appears in mixed case. It is not forced into any case.
ecUpperCase	The text is converted to uppercase.

29.2.4 StdCtrls.TEllipsisPosition Enumeration

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
  TEllipsisPosition = (
    epNone,
    epPathEllipsis,
    epEndEllipsis,
    epWordEllipsis
  );
```

C++

```
  enum TEllipsisPosition {
    epNone,
    epPathEllipsis,
    epEndEllipsis,
    epWordEllipsis
  };
```

File

StdCtrls

Description

TEllipsisPosition specifies how an ellipsis is placed in text that does not fit in its allocated rectangle. The constants here correspond to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

The following table lists values for TEllipsisPosition:

Value	Meaning
epNone	An ellipsis is not inserted.

epPathEllipsis	Replaces text in the middle with an ellipsis so that the resulting text fits in the rectangle specified. If the text contains backslash (\) characters, treat the text as a path and attempt to preserve as much text as possible after the last backslash. Corresponds to a value of DT_PATH_ELLIPSIS in the **dwDTFormat** parameter.
epEndEllipsis	Replaces text at the end with an ellipsis so that the resulting text fits in the rectangle specified. Any word not fitting in the rectangle is truncated without adding an ellipsis. Corresponds to a value of DT_END_ELLIPSIS in the **dwDTFormat** parameter.
epWordEllipsis	Truncates any word that doesn't fit in the rectangle and adds ellipses. Corresponds to a value of DT_WORD_ELLIPSIS in the **dwDTFormat** parameter.

29.2.5 StdCtrls.TImageAlignment Enumeration

Alignment of image on TButton (see page 1961).

Pascal

```
TImageAlignment = (
  iaLeft,
  iaRight,
  iaTop,
  iaBottom,
  iaCenter
);
```

C++

```
enum TImageAlignment {
  iaLeft,
  iaRight,
  iaTop,
  iaBottom,
  iaCenter
};
```

File

StdCtrls

Description

TImageAlignment indicates the alignment of an image on a TButton (see page 1961), which is the value of its ImageAlignment property. Following are the possible values of TImageAlignment:

Value	Meaning
iaLeft	Left aligned.
iaRight	Right aligned.
iaTop	Top aligned.
iaBottom	Bottom aligned.
iaCenter	Center aligned.

See Also

TButton (see page 1961)

ImageAlignment (see page 1983)

29.2.6 StdCtrls.TListBoxStyle Enumeration

TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.

Pascal

```
TListBoxStyle = (
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
);
```

C++

```
enum TListBoxStyle {
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
};
```

File

StdCtrls

Description

The following table lists the values for the TListBoxStyle type:

Value	Meaning
lbStandard	All items are strings, with each item the same height.
lbOwnerDrawFixed	The list box is owner-drawn, but each item in the list box is the height specified by the ItemHeight property. Each time an item is displayed in an lbOwnerDrawFixed list box, the OnDrawItem event occurs. The event handler for OnDrawItem draws the specified item. The ItemHeight property determines the height of each of the items.
lbOwnerDrawVariable	The list box is owner-drawn, and items in the list box can be of varying heights. Each time an item is displayed in an lbOwnerDrawVariable list box, two events occur. The first is the OnMeasureItem event. The code for the OnMeasureItem handler can set the height of each item. Then the OnDrawItem event occurs. The code for the OnDrawItem handler draws each item in the list box using the size specified by the OnMeasureItem handler.
lbVirtual	The list box is virtual, but all items are strings with the same height. You must indicate the number of items in the list box by setting the Count property. You must supply the items in the list box using an OnData event handler. If the items have associated objects, you must supply them using an OnDataObject event handler. In addition, you must supply an OnDataFind event handler to provide the mapping from the strings in the list box to their indexes.
lbVirtualOwnerDraw	The list box is virtual and owner-drawn. All items are the same height. As with lbOwnerDrawFixed, the item height is specified by the ItemHeight property, and you must draw items using an OnDrawItem event handler. As with lbVirtual, you must set the Count property and supply OnData and OnDataFind event handlers to provide the mapping between items and indexes. If items are associated with objects, you must also supply an OnDataObject event handler.

29.2.7 StdCtrls.TScrollCode Enumeration

TScrollCode indicates what action the user performed on a scroll bar.

Pascal

```
TScrollCode = (
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
);
```

C++

```
enum TScrollCode {
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
};
```

File

StdCtrls

Description

TScrollCode describes the type of scrolling action requested. It can be any of the following values:

Value	Meaning
scLineUp	User clicked the top or left scroll arrow or pressed the Up or Left arrow key.
scLineDown	User clicked the bottom or right scroll arrow or pressed the Down or Right arrow key.
scPageUp	User clicked the area to the left of the thumb tab or pressed the PgUp key.
scPageDown	User clicked the area to the right of the thumb tab or pressed the PgDn key.
scPosition	User positioned the thumb tab and released it.
scTrack	User is moving the thumb tab.
scTop	User moved the thumb tab to the top or far left on the scroll bar.
scBottom	User moved the thumb tab to the bottom or far right on the scroll bar.
scEndScroll	User finished moving the thumb tab on the scroll bar.

29.2.8 StdCtrls.TScrollStyle Enumeration

Identifies a set of scrollbars by their orientation.

Pascal

```
TScrollStyle = (
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
);
```

C++

```
enum TScrollStyle {
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
};
```

File

StdCtrls

Description

TScrollStyle can have one of the following values:

Value	Meaning
ssNone	No scrollbars are indicated.
ssHorizontal	Only a horizontal scrollbar.
ssVertical	Only a vertical scrollbar.
ssBoth	Both a horizontal and a vertical scrollbar.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

29.2.9 StdCtrls.TSelection Record

TSelection definese the start and position of a selection.

Pascal

```
TSelection = record
  EndPos: Integer;
  StartPos: Integer;
end;
```

C++

```
struct TSelection {
  int EndPos;
  int StartPos;
};
```

File

StdCtrls

Description

TSelection is used internally in Delphi to hold the start and end position of a selection.

29.2.10 StdCtrls.TStaticBorderStyle Enumeration

TStaticBorderStyle determines the appearance of the border on a static text control.

Pascal

```
TStaticBorderStyle = (
  sbsNone,
  sbsSingle,
  sbsSunken
);
```

C++

```
enum TStaticBorderStyle {
  sbsNone,
  sbsSingle,
  sbsSunken
};
```

File

StdCtrls

Description

These are the values in the TStaticBorderStyle type:

Value	Meaning
sbsNone	No visible border.
sbsSingle	Single-line border.
sbsSunken	Raised border.

29.2.11 StdCtrls.TTextLayout Enumeration

TTextLayout indicates where text appears within the client area of a control.

Pascal

```
TTextLayout = (
  tlTop,
  tlCenter,
  tlBottom
);
```

C++

```
enum TTextLayout {
  tlTop,
  tlCenter,
  tlBottom
};
```

File

StdCtrls

Description

The following table describes the possible values:

Value	Meaning
tlTop	The text appears at the top of the its control.
tlCenter	The text is vertically centered in the control.
tlBottom	The text appears along the bottom of the control.

29.3 Types

The following table lists types in this documentation.

Types

Name	Description
TButtonActionLinkClass (see page 2915)	TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2030).
TCustomComboBoxStringsClass (see page 2915)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2257).
TDrawItemEvent (see page 2916)	TScrollEvent (see page 2920) is the type of an OnScroll event handler.
TFNDrawText (see page 2916)	This is type StdCtrls.TFNDrawText.
TLBFindDataEvent (see page 2917)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 2917)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 2918)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 2918)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 2918)	Indicates state information that can influence how an item is drawn.
TPushButtonActionLinkClass (see page 2919)	This is type StdCtrls.TPushButtonActionLinkClass.
TScrollEvent (see page 2920)	This is type StdCtrls.TScrollEvent.

29.3.1 StdCtrls.TButtonActionLinkClass Type

TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2030).

Pascal

```
TButtonActionLinkClass = class of TButtonActionLink;
```

C++

```
TButtonActionLink TButtonActionLinkClass;
```

File

StdCtrls

Description

TButtonActionLinkClass is the metaclass for TButtonActionLink (see page 2030). Its value is the class reference for TButtonActionLink (see page 2030) or for one of its descendants.

See Also

TButtonActionLink (see page 2030)

29.3.2 StdCtrls.TCustomComboBoxStringsClass Type

TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2257).

Pascal

```
TCustomComboBoxStringsClass = class of TCustomComboBoxStrings;
```

C++

```
TCustomComboBoxStrings TCustomComboBoxStringsClass;
```

File

StdCtrls

Description

TCustomComboBoxStringsClass is the type for a class reference of TCustomComboBoxStrings (see page 2257) or any of its descendants. It is a pointer to the runtime type information (RTTI) for the class type.

29.3.3 StdCtrls.TDrawItemEvent Type

TScrollEvent (see page 2920) is the type of an OnScroll event handler.

Pascal

```
TDrawItemEvent = procedure (Control: TWinControl; Index: Integer; Rect: TRect; State:
TOwnerDrawState) of object;
```

C++

```
(Control: TWinControl; Index: Integer; Rect: TRect; State: TOwnerDrawState) (
TDrawItemEvent)();
```

File

StdCtrls

Description

TScrollEvent (see page 2920) defines the type of method procedure that you must define in order to implement the OnScroll event of a visual component such as TScroll.

The OnScroll event occurs when the user scrolls a scroll bar with the mouse or keyboard.

Sender is a reference to the component invoking the OnScroll event.

ScrollCode defines the user scroll action that was performed.

ScrollPos provides the new scroll position.

See Also

OnScroll

TScrollCode (see page 2911)

29.3.4 StdCtrls.TFNDrawText Type

Pascal

```
TFNDrawText = procedure (DC: HDC; const Text: UnicodeString; var TextRect: TRect; TextFlags:
Cardinal) of object;
```

C++

```
(DC: HDC; const Text: UnicodeString; var TextRect: TRect; TextFlags: Cardinal) (
TFNDrawText)();
```

File

StdCtrls

Description

This is type StdCtrls.TFNDrawText.

29.3.5 StdCtrls.TLBFindDataEvent Type

TLBFindDataEvent is the type of the list box OnDataFind event handler.

Pascal

```
TLBFindDataEvent = function (Control: TWinControl; FindString: string): Integer of object;
```

C++

```
(Control: TWinControl; FindString: string): Integer ( TLBFindDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item index is needed.

FindString is the text of the item whose index is needed.

The event handler returns the index of the item identified by FindString. This return value must be in the range from 0 to one less than the value of the Count property.

29.3.6 StdCtrls.TLBGetDataEvent Type

TLBGetDataEvent is the type of the list box OnData event handler.

Pascal

```
TLBGetDataEvent = procedure (Control: TWinControl; Index: Integer; var Data: string) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var Data: string) ( TLBGetDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item is needed.

Index is the index of the desired item. Index must be less than the value of the Count property.

Data returns the string that is the value of the specified item.

29.3.7 StdCtrls.TLBGetDataObjectEvent Type

TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.

Pascal

```
TLBGetDataObjectEvent = procedure (Control: TWinControl; Index: Integer; var DataObject:
TObject) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var DataObject: TObject) ( TLBGetDataObjectEvent)();
```

File

StdCtrls

Description

Control is the virtual list box that contains the item associated with the object.

Index is the index of the desired item. Index must be less than the value of the Count property.

DataObject returns the object associated with the specified item.

29.3.8 StdCtrls.TMeasureItemEvent Type

TMeasureItemEvent is the type for the OnMeasureItem event handler.

Pascal

```
TMeasureItemEvent = procedure (Control: TWinControl; Index: Integer; var Height: Integer) of
object;
```

C++

```
(Control: TWinControl; Index: Integer; var Height: Integer) ( TMeasureItemEvent)();
```

File

StdCtrls

Description

TMeasureItemEvent is the type for event handlers on custom-drawn controls that allow different items in a list to have different heights.

Control is a reference to the control that contains the item.

Index is the index of the item in the Items property.

Height is the default height of the item, in pixels. The event handler can reset Height to the appropriate value.

29.3.9 StdCtrls.TOwnerDrawState Type

Indicates state information that can influence how an item is drawn.

Pascal

```
TOwnerDrawState = Windows.TOwnerDrawState;
```

C++

```
Windows.TOwnerDrawState TOwnerDrawState;
```

File

StdCtrls

Description

TOwnerDrawState is used by event handlers in owner-draw controls to indicate the state of an item about to be drawn. It is a set that includes zero or more of the following:

Value	Meaning
odSelected	The item is selected.
odGrayed	The item should appear grayed.
odDisabled	The item is disabled.
odChecked	The item should appear checked. (This does not apply to all objects)
odFocused	Keyboard input is directed to the item.
odDefault	The item is the default item (This does not apply to all objects)
odHotLight	The item is hot-lighted because under the mouse (only if the operating system supports hot-lighting)
odInactive	The item and its associated menu are inactive
odNoAccel	Any underline of an accelerator character should be skipped when drawing the control
odNoFocusRect	Do not display visual cues that indicate focus
odReserved1	Not currently used.
odReserved2	Not currently used.
odComboBoxEdit	The drawing takes place in the edit box of a combo box control.

29.3.10 StdCtrls.TPushButtonActionLinkClass Type

Pascal

```
TPushButtonActionLinkClass = class of TPushButtonActionLink;
```

C++

```
TPushButtonActionLink TPushButtonActionLinkClass;
```

File

StdCtrls

Description

This is type StdCtrls.TPushButtonActionLinkClass.

29.3.11 StdCtrls.TScrollEvent Type

Pascal

```
TScrollEvent = procedure (Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) of object;
```

C++

```
(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) ( TScrollEvent)();
```

File

StdCtrls

Description

This is type StdCtrls.TScrollEvent.

30 StdStyleActnCtrls Namespace

This is namespace StdStyleActnCtrls.

Topics

Name	Description
Classes (see page 2922)	The following table lists classes in this documentation.
Variables (see page 2925)	The following table lists variables in this documentation.

Classes

	Name	Description
	TStandardStyleActionBars (see page 2922)	This is class StdStyleActnCtrls.TStandardStyleActionBars.

Variables

Name	Description
StandardStyle (see page 2925)	This is variable StdStyleActnCtrls.StandardStyle.

30.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardStyleActionBars (see page 2922)	This is class StdStyleActnCtrls.TStandardStyleActionBars.

30.1.1 TStandardStyleActionBars Class

Class Hierarchy

Pascal

```
TStandardStyleActionBars = class(TActionBarStyleEx);
```

C++

```
class TStandardStyleActionBars : public TActionBarStyleEx;
```

File

StdStyleActnCtrls

Description

This is class StdStyleActnCtrls.TStandardStyleActionBars.

30.1.1.1 TStandardStyleActionBars Members

The following tables list the members exposed by TStandardStyleActionBars.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 2923)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 2923)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 2923)	Returns the control class type on an action bar item.
	GetPopupClass (see page 2924)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 2924)	This is GetScrollBtnClass, a member of class TStandardStyleActionBars.
	GetStyleName (see page 2924)	Returns the style associated with action manager.

30.1.1.2 TStandardStyleActionBars Methods

The methods of the TStandardStyleActionBars class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 2923)	Returns the base class for ActionBand menus for customizing action bands.

	GetColorMapClass (see page 2923)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 2923)	Returns the control class type on an action bar item.
	GetPopupClass (see page 2924)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 2924)	This is GetScrollBtnClass, a member of class TStandardStyleActionBars.
	GetStyleName (see page 2924)	Returns the style associated with action manager.

30.1.1.2.1 TStandardStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

30.1.1.2.2 TStandardStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

30.1.1.2.3 TStandardStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

30.1.1.2.4 TStandardStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

30.1.1.2.5 TStandardStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TStandardStyleActionBars.

30.1.1.2.6 TStandardStyleActionBars.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetStyleName();
```

Description

Returns the value of the Style property.

30.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
StandardStyle (see page 2925)	This is variable StdStyleActnCtrls.StandardStyle.

30.2.1 StdStyleActnCtrls.StandardStyle Variable

Pascal

```
StandardStyle: TStandardStyleActionBars;
```

C++

```
TStandardStyleActionBars StandardStyle;
```

File

StdStyleActnCtrls

Description

This is variable StdStyleActnCtrls.StandardStyle.

31 StdVCL Namespace

This is namespace StdVCL.

Topics

Name	Description
Interfaces (see page 2928)	The following table lists interfaces in this documentation.
Constants (see page 2944)	The following table lists constants in this documentation.

Constants

Name	Description
IID_IDataBroker (see page 2944)	This is constant StdVCL.IID_IDataBroker.
IID_IProvider (see page 2944)	This is constant StdVCL.IID_IProvider.
IID_IStrings (see page 2945)	This is constant StdVCL.IID_IStrings.
LIBID_StdVCL (see page 2945)	This is constant StdVCL.LIBID_StdVCL.

Interfaces

	Name	Description
	IDataBroker (see page 2928)	This is class StdVCL.IDataBroker.
	IDataBrokerDisp (see page 2929)	This is class StdVCL.IDataBrokerDisp.
	IProvider (see page 2930)	This is class StdVCL.IProvider.
	IProviderDisp (see page 2934)	This is class StdVCL.IProviderDisp.
	IStrings (see page 2937)	This is class StdVCL.IStrings.
	IStringsDisp (see page 2940)	This is class StdVCL.IStringsDisp.

31.1 Interfaces

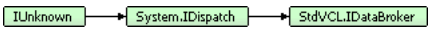
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDataBroker (see page 2928)	This is class StdVCL.IDataBroker.
	IDataBrokerDisp (see page 2929)	This is class StdVCL.IDataBrokerDisp.
	IProvider (see page 2930)	This is class StdVCL.IProvider.
	IProviderDisp (see page 2934)	This is class StdVCL.IProviderDisp.
	IStrings (see page 2937)	This is class StdVCL.IStrings.
	IStringsDisp (see page 2940)	This is class StdVCL.IStringsDisp.

31.1.1 IDataBroker Interface

Class Hierarchy



Pascal

```
IDataBroker = interface(IDispatch);
```

C++

```
__interface IDataBroker : public IDispatch;
```

File

StdVCL

Description

This is class StdVCL.IDataBroker.

31.1.1.1 IDataBroker Members

The following tables list the members exposed by IDataBroker.

Public Methods

	Name	Description
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetProviderNames (see page 2929)	This is GetProviderNames, a member of class IDataBroker.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.

31.1.1.2 IDataBroker Methods

The methods of the IDataBroker class are listed here.

Public Methods

	Name	Description
	GetProviderNames (see page 2929)	This is GetProviderNames, a member of class IDataBroker.

31.1.1.2.1 IDataBroker.GetProviderNames Method

Pascal

```
function GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProviderNames();
```

Description

This is GetProviderNames, a member of class IDataBroker.

31.1.2 IDataBrokerDisp Interface

Class Hierarchy

StdVCL.IDataBrokerDisp

Pascal

```
IDataBrokerDisp = dispinterface;
```

C++

```
class IDataBrokerDisp;
```

File

StdVCL

Description

This is class StdVCL.IDataBrokerDisp.

31.1.2.1 IDataBrokerDisp Members

The following tables list the members exposed by IDataBrokerDisp.

Public Methods

	Name	Description
	GetProviderNames (see page 2930)	This is GetProviderNames, a member of class IDataBrokerDisp.

31.1.2.2 IDataBrokerDisp Methods

The methods of the IDataBrokerDisp class are listed here.

Public Methods

	Name	Description
	GetProviderNames (see page 2930)	This is GetProviderNames, a member of class IDataBrokerDisp.

31.1.2.2.1 IDataBrokerDisp.GetProviderNames Method

Pascal

```
function GetProviderNames: OleVariant;
```

C++

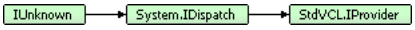
```
__fastcall OleVariant GetProviderNames();
```

Description

This is GetProviderNames, a member of class IDataBrokerDisp.

31.1.3 IProvider Interface

Class Hierarchy



Pascal

```
IProvider = interface(IDispatch);
```

C++

```
__interface IProvider : public IDispatch;
```

File

StdVCL

Description

This is class StdVCL.IProvider.

31.1.3.1 IProvider Members

The following tables list the members exposed by IProvider.

Public Methods

	Name	Description
	ApplyUpdates (see page 2931)	This is ApplyUpdates, a member of class IProvider.
	DataRequest (see page 2931)	This is DataRequest, a member of class IProvider.
	Get_Constraints (see page 2931)	This is Get_Constraints, a member of class IProvider.
	Get_Data (see page 2932)	This is Get_Data, a member of class IProvider.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetMetaData (see page 2932)	This is GetMetaData, a member of class IProvider.
	GetRecords (see page 2932)	This is GetRecords, a member of class IProvider.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Reset (see page 2932)	This is Reset, a member of class IProvider.
	Set_Constraints (see page 2933)	This is Set_Constraints, a member of class IProvider.
	SetParams (see page 2933)	This is SetParams, a member of class IProvider.

Public Properties

	Name	Description
	Constraints (see page 2933)	This is Constraints, a member of class IProvider.
R	Data (see page 2933)	This is Data, a member of class IProvider.

31.1.3.2 IProvider Methods

The methods of the IProvider class are listed here.

Public Methods

	Name	Description
	ApplyUpdates (see page 2931)	This is ApplyUpdates, a member of class IProvider.
	DataRequest (see page 2931)	This is DataRequest, a member of class IProvider.
	Get_Constraints (see page 2931)	This is Get_Constraints, a member of class IProvider.
	Get_Data (see page 2932)	This is Get_Data, a member of class IProvider.
	GetMetaData (see page 2932)	This is GetMetaData, a member of class IProvider.
	GetRecords (see page 2932)	This is GetRecords, a member of class IProvider.
	Reset (see page 2932)	This is Reset, a member of class IProvider.
	Set_Constraints (see page 2933)	This is Set_Constraints, a member of class IProvider.
	SetParams (see page 2933)	This is SetParams, a member of class IProvider.

31.1.3.2.1 IProvider.ApplyUpdates Method

Pascal

```
function ApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant; safecall;
```

C++

```
__fastcall OleVariant ApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is ApplyUpdates, a member of class IProvider.

31.1.3.2.2 IProvider.DataRequest Method

Pascal

```
function DataRequest(Input: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant DataRequest(OleVariant Input);
```

Description

This is DataRequest, a member of class IProvider.

31.1.3.2.3 IProvider.Get_Constraints Method

Pascal

```
function Get_Constraints: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Constraints();
```

Description

This is Get_Constraints, a member of class IProvider.

31.1.3.2.4 IProvider.Get_Data Method

Pascal

```
function Get_Data: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Data();
```

Description

This is Get_Data, a member of class IProvider.

31.1.3.2.5 IProvider.GetMetaData Method

Pascal

```
function GetMetaData: OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetMetaData();
```

Description

This is GetMetaData, a member of class IProvider.

31.1.3.2.6 IProvider.GetRecords Method

Pascal

```
function GetRecords(Count: Integer; out RecsOut: Integer): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetRecords(int Count, int RecsOut);
```

Description

This is GetRecords, a member of class IProvider.

31.1.3.2.7 IProvider.Reset Method

Pascal

```
procedure Reset(MetaData: WordBool); safecall;
```

C++

```
__fastcall Reset(WordBool MetaData);
```

Description

This is Reset, a member of class IProvider.

31.1.3.2.8 IProvider.Set_Constraints Method

Pascal

```
procedure Set_Constraints(Value: WordBool); safecall;
```

C++

```
__fastcall Set_Constraints(WordBool Value);
```

Description

This is Set_Constraints, a member of class IProvider.

31.1.3.2.9 IProvider.SetParams Method

Pascal

```
procedure SetParams(Values: OleVariant); safecall;
```

C++

```
__fastcall SetParams(OleVariant Values);
```

Description

This is SetParams, a member of class IProvider.

31.1.3.3 IProvider Properties

The properties of the IProvider class are listed here.

Public Properties

	Name	Description
	Constraints (see page 2933)	This is Constraints, a member of class IProvider.
R	Data (see page 2933)	This is Data, a member of class IProvider.

31.1.3.3.1 IProvider.Constraints Property

Pascal

```
property Constraints: WordBool;
```

C++

```
__property WordBool Constraints;
```

Description

This is Constraints, a member of class IProvider.

31.1.3.3.2 IProvider.Data Property

Pascal

```
property Data: OleVariant;
```

C++

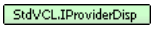
```
__property OleVariant Data;
```

Description

This is Data, a member of class IProvider.

31.1.4 IProviderDisp Interface

Class Hierarchy



Pascal

```
IProviderDisp = dispinterface;
```

C++

```
class IProviderDisp;
```

File

StdVCL

Description

This is class StdVCL.IProviderDisp.

31.1.4.1 IProviderDisp Members

The following tables list the members exposed by IProviderDisp.

Public Methods

	Name	Description
	ApplyUpdates (see page 2935)	This is ApplyUpdates, a member of class IProviderDisp.
	DataRequest (see page 2935)	This is DataRequest, a member of class IProviderDisp.
	GetMetaData (see page 2935)	This is GetMetaData, a member of class IProviderDisp.
	GetRecords (see page 2935)	This is GetRecords, a member of class IProviderDisp.
	Reset (see page 2936)	This is Reset, a member of class IProviderDisp.
	SetParams (see page 2936)	This is SetParams, a member of class IProviderDisp.

Public Properties

	Name	Description
	Constraints (see page 2936)	This is Constraints, a member of class IProviderDisp.
	Data (see page 2936)	This is Data, a member of class IProviderDisp.

31.1.4.2 IProviderDisp Methods

The methods of the IProviderDisp class are listed here.

Public Methods

	Name	Description
	ApplyUpdates (see page 2935)	This is ApplyUpdates, a member of class IProviderDisp.
	DataRequest (see page 2935)	This is DataRequest, a member of class IProviderDisp.
	GetMetaData (see page 2935)	This is GetMetaData, a member of class IProviderDisp.
	GetRecords (see page 2935)	This is GetRecords, a member of class IProviderDisp.

	Reset (see page 2936)	This is Reset, a member of class IProviderDisp.
	SetParams (see page 2936)	This is SetParams, a member of class IProviderDisp.

31.1.4.2.1 IProviderDisp.ApplyUpdates Method

Pascal

```
function ApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant;
```

C++

```
__fastcall OleVariant ApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is ApplyUpdates, a member of class IProviderDisp.

31.1.4.2.2 IProviderDisp.DataRequest Method

Pascal

```
function DataRequest(Input: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant DataRequest(OleVariant Input);
```

Description

This is DataRequest, a member of class IProviderDisp.

31.1.4.2.3 IProviderDisp.GetMetaData Method

Pascal

```
function GetMetaData: OleVariant;
```

C++

```
__fastcall OleVariant GetMetaData();
```

Description

This is GetMetaData, a member of class IProviderDisp.

31.1.4.2.4 IProviderDisp.GetRecords Method

Pascal

```
function GetRecords(Count: Integer; out RecsOut: Integer): OleVariant;
```

C++

```
__fastcall OleVariant GetRecords(int Count, int RecsOut);
```

Description

This is GetRecords, a member of class IProviderDisp.

31.1.4.2.5 IProviderDisp.Reset Method

Pascal

```
procedure Reset(MetaData: WordBool);
```

C++

```
__fastcall Reset(WordBool MetaData);
```

Description

This is Reset, a member of class IProviderDisp.

31.1.4.2.6 IProviderDisp.SetParams Method

Pascal

```
procedure SetParams(Values: OleVariant);
```

C++

```
__fastcall SetParams(OleVariant Values);
```

Description

This is SetParams, a member of class IProviderDisp.

31.1.4.3 IProviderDisp Properties

The properties of the IProviderDisp class are listed here.

Public Properties

	Name	Description
	Constraints (see page 2936)	This is Constraints, a member of class IProviderDisp.
	Data (see page 2936)	This is Data, a member of class IProviderDisp.

31.1.4.3.1 IProviderDisp.Constraints Property

Pascal

```
property Constraints: WordBool;
```

C++

```
__property WordBool Constraints;
```

Description

This is Constraints, a member of class IProviderDisp.

31.1.4.3.2 IProviderDisp.Data Property

Pascal

```
property Data: OleVariant;
```

C++

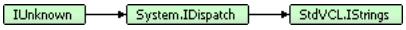
```
__property OleVariant Data;
```

Description

This is Data, a member of class IProviderDisp.

31.1.5 IStrings Interface

Class Hierarchy



Pascal

```
IStrings = interface(IDispatch);
```

C++

```
__interface IStrings : public IDispatch;
```

File

StdVCL

Description

This is class StdVCL.IStrings.

31.1.5.1 IStrings Members

The following tables list the members exposed by IStrings.

Public Methods

	Name	Description
	_NewEnum (see page 2938)	This is _NewEnum, a member of class IStrings.
	Add (see page 2938)	This is Add, a member of class IStrings.
	Clear (see page 2938)	This is Clear, a member of class IStrings.
	Count (see page 2938)	This is Count, a member of class IStrings.
	Get_ControlDefault (see page 2939)	This is Get_ControlDefault, a member of class IStrings.
	Get_Item (see page 2939)	This is Get_Item, a member of class IStrings.
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.
	Remove (see page 2939)	This is Remove, a member of class IStrings.
	Set_ControlDefault (see page 2939)	This is Set_ControlDefault, a member of class IStrings.
	Set_Item (see page 2940)	This is Set_Item, a member of class IStrings.

Public Properties

	Name	Description
	ControlDefault (see page 2940)	This is ControlDefault, a member of class IStrings.
	Item (see page 2940)	This is Item, a member of class IStrings.

31.1.5.2 IStrings Methods

The methods of the IStrings class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 2938)	This is _NewEnum, a member of class IStrings.
	Add (see page 2938)	This is Add, a member of class IStrings.
	Clear (see page 2938)	This is Clear, a member of class IStrings.
	Count (see page 2938)	This is Count, a member of class IStrings.
	Get_ControlDefault (see page 2939)	This is Get_ControlDefault, a member of class IStrings.
	Get_Item (see page 2939)	This is Get_Item, a member of class IStrings.
	Remove (see page 2939)	This is Remove, a member of class IStrings.
	Set_ControlDefault (see page 2939)	This is Set_ControlDefault, a member of class IStrings.
	Set_Item (see page 2940)	This is Set_Item, a member of class IStrings.

31.1.5.2.1 IStrings._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IStrings.

31.1.5.2.2 IStrings.Add Method

Pascal

```
function Add(Item: OleVariant): Integer; safecall;
```

C++

```
__fastcall int Add(OleVariant Item);
```

Description

This is Add, a member of class IStrings.

31.1.5.2.3 IStrings.Clear Method

Pascal

```
procedure Clear; safecall;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class IStrings.

31.1.5.2.4 IStrings.Count Method

Pascal

```
function Count: Integer; safecall;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class IStrings.

31.1.5.2.5 IStrings.Get_ControlDefault Method

Pascal

```
function Get_ControlDefault(Index: Integer): OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_ControlDefault(int Index);
```

Description

This is Get_ControlDefault, a member of class IStrings.

31.1.5.2.6 IStrings.Get_Item Method

Pascal

```
function Get_Item(Index: Integer): OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Item(int Index);
```

Description

This is Get_Item, a member of class IStrings.

31.1.5.2.7 IStrings.Remove Method

Pascal

```
procedure Remove(Index: Integer); safecall;
```

C++

```
__fastcall Remove(int Index);
```

Description

This is Remove, a member of class IStrings.

31.1.5.2.8 IStrings.Set_ControlDefault Method

Pascal

```
procedure Set_ControlDefault(Index: Integer; Value: OleVariant); safecall;
```

C++

```
__fastcall Set_ControlDefault(int Index, OleVariant Value);
```

Description

This is Set_ControlDefault, a member of class IStrings.

31.1.5.2.9 IStrings.Set_Item Method

Pascal

```
procedure Set_Item(Index: Integer; Value: OleVariant); safecall;
```

C++

```
__fastcall Set_Item(int Index, OleVariant Value);
```

Description

This is Set_Item, a member of class IStrings.

31.1.5.3 IStrings Properties

The properties of the IStrings class are listed here.

Public Properties

	Name	Description
	ControlDefault (see page 2940)	This is ControlDefault, a member of class IStrings.
	Item (see page 2940)	This is Item, a member of class IStrings.

31.1.5.3.1 IStrings.ControlDefault Property

Pascal

```
property ControlDefault [Index: Integer]: OleVariant;
```

C++

```
__property OleVariant ControlDefault[int Index];
```

Description

This is ControlDefault, a member of class IStrings.

31.1.5.3.2 IStrings.Item Property

Pascal

```
property Item [Index: Integer]: OleVariant;
```

C++

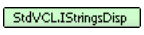
```
__property OleVariant Item[int Index];
```

Description

This is Item, a member of class IStrings.

31.1.6 IStringsDisp Interface

Class Hierarchy



Pascal

```
IStringsDisp = dispinterface;
```

C++

```
class IStringsDisp;
```

File

StdVCL

Description

This is class StdVCL.IStringsDisp.

31.1.6.1 IStringsDisp Members

The following tables list the members exposed by IStringsDisp.

Public Methods

	Name	Description
	_NewEnum (see page 2941)	This is _NewEnum, a member of class IStringsDisp.
	Add (see page 2942)	This is Add, a member of class IStringsDisp.
	Clear (see page 2942)	This is Clear, a member of class IStringsDisp.
	Count (see page 2942)	This is Count, a member of class IStringsDisp.
	Remove (see page 2942)	This is Remove, a member of class IStringsDisp.

Public Properties

	Name	Description
	ControlDefault (see page 2943)	This is ControlDefault, a member of class IStringsDisp.
	Item (see page 2943)	This is Item, a member of class IStringsDisp.

31.1.6.2 IStringsDisp Methods

The methods of the IStringsDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 2941)	This is _NewEnum, a member of class IStringsDisp.
	Add (see page 2942)	This is Add, a member of class IStringsDisp.
	Clear (see page 2942)	This is Clear, a member of class IStringsDisp.
	Count (see page 2942)	This is Count, a member of class IStringsDisp.
	Remove (see page 2942)	This is Remove, a member of class IStringsDisp.

31.1.6.2.1 IStringsDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IStringsDisp.

31.1.6.2.2 IStringsDisp.Add Method

Pascal

```
function Add(Item: OleVariant): Integer;
```

C++

```
__fastcall int Add(OleVariant Item);
```

Description

This is Add, a member of class IStringsDisp.

31.1.6.2.3 IStringsDisp.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class IStringsDisp.

31.1.6.2.4 IStringsDisp.Count Method

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class IStringsDisp.

31.1.6.2.5 IStringsDisp.Remove Method

Pascal

```
procedure Remove(Index: Integer);
```

C++

```
__fastcall Remove(int Index);
```

Description

This is Remove, a member of class IStringsDisp.

31.1.6.3 IStringsDisp Properties

The properties of the IStringsDisp class are listed here.

Public Properties

	Name	Description
	ControlDefault (see page 2943)	This is ControlDefault, a member of class IStringsDisp.
	Item (see page 2943)	This is Item, a member of class IStringsDisp.

31.1.6.3.1 IStringsDisp.ControlDefault Property

Pascal

```
property ControlDefault [Index: Integer]: OleVariant;
```

C++

```
__property OleVariant ControlDefault[int Index];
```

Description

This is ControlDefault, a member of class IStringsDisp.

31.1.6.3.2 IStringsDisp.Item Property

Pascal

```
property Item [Index: Integer]: OleVariant;
```

C++

```
__property OleVariant Item[int Index];
```

Description

This is Item, a member of class IStringsDisp.

31.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IID_IDataBroker (see page 2944)	This is constant StdVCL.IID_IDataBroker.
IID_IProvider (see page 2944)	This is constant StdVCL.IID_IProvider.
IID_IStrings (see page 2945)	This is constant StdVCL.IID_IStrings.
LIBID_StdVCL (see page 2945)	This is constant StdVCL.LIBID_StdVCL.

31.2.1 StdVCL.IID_IDataBroker Constant

Pascal

```
IID_IDataBroker: TGUID = '{6539BF65-6FE7-11D0-9E8C-00A02457621F}';
```

C++

```
TGUID IID_IDataBroker = '{6539BF65-6FE7-11D0-9E8C-00A02457621F}';
```

File

StdVCL

Description

This is constant StdVCL.IID_IDataBroker.

31.2.2 StdVCL.IID_IProvider Constant

Pascal

```
IID_IProvider: TGUID = '{6E644935-51F7-11D0-8D41-00A0248E4B9A}';
```

C++

```
TGUID IID_IProvider = '{6E644935-51F7-11D0-8D41-00A0248E4B9A}';
```

File

StdVCL

Description

This is constant StdVCL.IID_IProvider.

31.2.3 StdVCL.IID_IStrings Constant

Pascal

```
IID_IStrings: TGUID = '{EE05DFE2-5549-11D0-9EA9-0020AF3D82DA}';
```

C++

```
TGUID IID_IStrings = '{EE05DFE2-5549-11D0-9EA9-0020AF3D82DA}';
```

File

StdVCL

Description

This is constant StdVCL.IID_IStrings.

31.2.4 StdVCL.LIBID_StdVCL Constant

Pascal

```
LIBID_StdVCL: TGUID = '{EE05DFE0-5549-11D0-9EA9-0020AF3D82DA}';
```

C++

```
TGUID LIBID_StdVCL = '{EE05DFE0-5549-11D0-9EA9-0020AF3D82DA}';
```

File

StdVCL

Description

This is constant StdVCL.LIBID_StdVCL.

32 StrHlpr Namespace

This is namespace StrHlpr.

Topics

Name	Description
Functions (see page 2949)	The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiAppend (see page 2950)	This is function StrHlpr.AnsiAppend.
	AnsiAssign (see page 2950)	This is function StrHlpr.AnsiAssign.
	AnsiCat (see page 2950)	This is function StrHlpr.AnsiCat.
	AnsiCopy (see page 2951)	This is function StrHlpr.AnsiCopy.
	AnsiDelete (see page 2951)	This is function StrHlpr.AnsiDelete.
	AnsiEqual (see page 2951)	This is function StrHlpr.AnsiEqual.
	AnsiFree (see page 2952)	This is function StrHlpr.AnsiFree.
	AnsiFromPChar (see page 2952)	This is function StrHlpr.AnsiFromPChar.
	AnsiFromPWideChar (see page 2952)	This is function StrHlpr.AnsiFromPWideChar.
	AnsiFromUnicode (see page 2953)	This is function StrHlpr.AnsiFromUnicode.
	AnsiFromWide (see page 2953)	This is function StrHlpr.AnsiFromWide.
	AnsiInsert (see page 2953)	This is function StrHlpr.AnsiInsert.
	AnsiPos (see page 2954)	This is function StrHlpr.AnsiPos.
	AnsiSetLength (see page 2954)	This is function StrHlpr.AnsiSetLength.
	UnicodeAppend (see page 2954)	This is function StrHlpr.UnicodeAppend.
	UnicodeAssign (see page 2955)	This is function StrHlpr.UnicodeAssign.
	UnicodeCat (see page 2955)	This is function StrHlpr.UnicodeCat.
	UnicodeCopy (see page 2955)	This is function StrHlpr.UnicodeCopy.
	UnicodeDelete (see page 2956)	This is function StrHlpr.UnicodeDelete.
	UnicodeEqual (see page 2956)	This is function StrHlpr.UnicodeEqual.
	UnicodeFree (see page 2956)	This is function StrHlpr.UnicodeFree.
	UnicodeFromAnsi (see page 2957)	This is function StrHlpr.UnicodeFromAnsi.
	UnicodeFromPChar (see page 2957)	This is function StrHlpr.UnicodeFromPChar.
	UnicodeFromPWideChar (see page 2957)	This is function StrHlpr.UnicodeFromPWideChar.
	UnicodeFromWide (see page 2958)	This is function StrHlpr.UnicodeFromWide.
	UnicodeGreater (see page 2958)	This is function StrHlpr.UnicodeGreater.
	UnicodeInsert (see page 2958)	This is function StrHlpr.UnicodeInsert.

	UnicodeLess (see page 2959)	This is function StrHlpr.UnicodeLess.
	UnicodePos (see page 2959)	This is function StrHlpr.UnicodePos.
	UnicodeSetLength (see page 2959)	This is function StrHlpr.UnicodeSetLength.
	WideAppend (see page 2960)	This is function StrHlpr.WideAppend.
	WideAssign (see page 2960)	This is function StrHlpr.WideAssign.
	WideCat (see page 2960)	This is function StrHlpr.WideCat.
	WideCopy (see page 2961)	This is function StrHlpr.WideCopy.
	WideDelete (see page 2961)	This is function StrHlpr.WideDelete.
	WideEqual (see page 2961)	This is function StrHlpr.WideEqual.
	WideFree (see page 2962)	This is function StrHlpr.WideFree.
	WideFromAnsi (see page 2962)	This is function StrHlpr.WideFromAnsi.
	WideFromPChar (see page 2962)	This is function StrHlpr.WideFromPChar.
	WideFromUnicode (see page 2963)	This is function StrHlpr.WideFromUnicode.
	WideGreater (see page 2963)	This is function StrHlpr.WideGreater.
	WideInsert (see page 2963)	This is function StrHlpr.WideInsert.
	WideLength (see page 2964)	This is function StrHlpr.WideLength.
	WideLess (see page 2964)	This is function StrHlpr.WideLess.
	WidePos (see page 2964)	This is function StrHlpr.WidePos.
	WideSetLength (see page 2965)	This is function StrHlpr.WideSetLength.

32.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiAppend (see page 2950)	This is function StrHlpr.AnsiAppend.
	AnsiAssign (see page 2950)	This is function StrHlpr.AnsiAssign.
	AnsiCat (see page 2950)	This is function StrHlpr.AnsiCat.
	AnsiCopy (see page 2951)	This is function StrHlpr.AnsiCopy.
	AnsiDelete (see page 2951)	This is function StrHlpr.AnsiDelete.
	AnsiEqual (see page 2951)	This is function StrHlpr.AnsiEqual.
	AnsiFree (see page 2952)	This is function StrHlpr.AnsiFree.
	AnsiFromPChar (see page 2952)	This is function StrHlpr.AnsiFromPChar.
	AnsiFromPWideChar (see page 2952)	This is function StrHlpr.AnsiFromPWideChar.
	AnsiFromUnicode (see page 2953)	This is function StrHlpr.AnsiFromUnicode.
	AnsiFromWide (see page 2953)	This is function StrHlpr.AnsiFromWide.
	AnsiInsert (see page 2953)	This is function StrHlpr.AnsiInsert.
	AnsiPos (see page 2954)	This is function StrHlpr.AnsiPos.
	AnsiSetLength (see page 2954)	This is function StrHlpr.AnsiSetLength.
	UnicodeAppend (see page 2954)	This is function StrHlpr.UnicodeAppend.
	UnicodeAssign (see page 2955)	This is function StrHlpr.UnicodeAssign.
	UnicodeCat (see page 2955)	This is function StrHlpr.UnicodeCat.
	UnicodeCopy (see page 2955)	This is function StrHlpr.UnicodeCopy.
	UnicodeDelete (see page 2956)	This is function StrHlpr.UnicodeDelete.
	UnicodeEqual (see page 2956)	This is function StrHlpr.UnicodeEqual.
	UnicodeFree (see page 2956)	This is function StrHlpr.UnicodeFree.
	UnicodeFromAnsi (see page 2957)	This is function StrHlpr.UnicodeFromAnsi.
	UnicodeFromPChar (see page 2957)	This is function StrHlpr.UnicodeFromPChar.
	UnicodeFromPWideChar (see page 2957)	This is function StrHlpr.UnicodeFromPWideChar.
	UnicodeFromWide (see page 2958)	This is function StrHlpr.UnicodeFromWide.
	UnicodeGreater (see page 2958)	This is function StrHlpr.UnicodeGreater.
	UnicodeInsert (see page 2958)	This is function StrHlpr.UnicodeInsert.
	UnicodeLess (see page 2959)	This is function StrHlpr.UnicodeLess.
	UnicodePos (see page 2959)	This is function StrHlpr.UnicodePos.
	UnicodeSetLength (see page 2959)	This is function StrHlpr.UnicodeSetLength.
	WideAppend (see page 2960)	This is function StrHlpr.WideAppend.
	WideAssign (see page 2960)	This is function StrHlpr.WideAssign.
	WideCat (see page 2960)	This is function StrHlpr.WideCat.
	WideCopy (see page 2961)	This is function StrHlpr.WideCopy.
	WideDelete (see page 2961)	This is function StrHlpr.WideDelete.
	WideEqual (see page 2961)	This is function StrHlpr.WideEqual.
	WideFree (see page 2962)	This is function StrHlpr.WideFree.
	WideFromAnsi (see page 2962)	This is function StrHlpr.WideFromAnsi.
	WideFromPChar (see page 2962)	This is function StrHlpr.WideFromPChar.
	WideFromUnicode (see page 2963)	This is function StrHlpr.WideFromUnicode.
	WideGreater (see page 2963)	This is function StrHlpr.WideGreater.
	WideInsert (see page 2963)	This is function StrHlpr.WideInsert.
	WideLength (see page 2964)	This is function StrHlpr.WideLength.
	WideLess (see page 2964)	This is function StrHlpr.WideLess.

	WidePos (see page 2964)	This is function StrHlpr.WidePos.
	WideSetLength (see page 2965)	This is function StrHlpr.WideSetLength.

32.1.1 StrHlpr.AnsiAppend Function

Pascal

```
procedure AnsiAppend(var dst: RawByteString; const src: RawByteString);
```

C++

```
AnsiAppend(RawByteString dst, const RawByteString src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiAppend.

32.1.2 StrHlpr.AnsiAssign Function

Pascal

```
procedure AnsiAssign(var dst: RawByteString; var src: RawByteString);
```

C++

```
AnsiAssign(RawByteString dst, RawByteString src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiAssign.

32.1.3 StrHlpr.AnsiCat Function

Pascal

```
function AnsiCat(const x: RawByteString; const y: RawByteString): RawByteString;
```

C++

```
RawByteString AnsiCat(const RawByteString x, const RawByteString y);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiCat.

32.1.4 StrHlpr.AnsiCopy Function

Pascal

```
function AnsiCopy(const src: RawByteString; index: Integer; count: Integer): RawByteString;
```

C++

```
RawByteString AnsiCopy(const RawByteString src, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiCopy.

32.1.5 StrHlpr.AnsiDelete Function

Pascal

```
procedure AnsiDelete(var dst: RawByteString; index: Integer; count: Integer);
```

C++

```
AnsiDelete(RawByteString dst, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiDelete.

32.1.6 StrHlpr.AnsiEqual Function

Pascal

```
function AnsiEqual(const x: RawByteString; const y: RawByteString): Boolean;
```

C++

```
Boolean AnsiEqual(const RawByteString x, const RawByteString y);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiEqual.

32.1.7 StrHlpr.AnsiFree Function

Pascal

```
procedure AnsiFree(var s: RawByteString);
```

C++

```
AnsiFree(RawByteString s);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiFree.

32.1.8 StrHlpr.AnsiFromPChar Function

Pascal

```
procedure AnsiFromPChar(var dst: RawByteString; src: PAnsiChar);
```

C++

```
AnsiFromPChar(RawByteString dst, PAnsiChar src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiFromPChar.

32.1.9 StrHlpr.AnsiFromPWideChar Function

Pascal

```
procedure AnsiFromPWideChar(var dst: RawByteString; src: PWideChar);
```

C++

```
AnsiFromPWideChar(RawByteString dst, PWideChar src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiFromPWideChar.

32.1.10 StrHlpr.AnsiFromUnicode Function

Pascal

```
procedure AnsiFromUnicode(var dst: RawByteString; const src: UnicodeString);
```

C++

```
AnsiFromUnicode(RawByteString dst, const UnicodeString src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiFromUnicode.

32.1.11 StrHlpr.AnsiFromWide Function

Pascal

```
procedure AnsiFromWide(var dst: RawByteString; const src: WideString);
```

C++

```
AnsiFromWide(RawByteString dst, const BSTR src);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiFromWide.

32.1.12 StrHlpr.AnsiInsert Function

Pascal

```
procedure AnsiInsert(var dst: RawByteString; const src: RawByteString; index: Integer);
```

C++

```
AnsiInsert(RawByteString dst, const RawByteString src, int index);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiInsert.

32.1.13 StrHlpr.AnsiPos Function

Pascal

```
function AnsiPos(const src: RawByteString; const sub: RawByteString): Integer;
```

C++

```
int AnsiPos(const RawByteString src, const RawByteString sub);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiPos.

32.1.14 StrHlpr.AnsiSetLength Function

Pascal

```
procedure AnsiSetLength(var dst: RawByteString; len: Integer);
```

C++

```
AnsiSetLength(RawByteString dst, int len);
```

File

StrHlpr

Description

This is function StrHlpr.AnsiSetLength.

32.1.15 StrHlpr.UnicodeAppend Function

Pascal

```
procedure UnicodeAppend(var dst: UnicodeString; const src: UnicodeString);
```

C++

```
UnicodeAppend(UnicodeString dst, const UnicodeString src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeAppend.

32.1.16 StrHlpr.UnicodeAssign Function

Pascal

```
procedure UnicodeAssign(var dst: UnicodeString; var src: UnicodeString);
```

C++

```
UnicodeAssign(UnicodeString dst, UnicodeString src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeAssign.

32.1.17 StrHlpr.UnicodeCat Function

Pascal

```
function UnicodeCat(const x: UnicodeString; const y: UnicodeString): UnicodeString;
```

C++

```
UnicodeString UnicodeCat(const UnicodeString x, const UnicodeString y);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeCat.

32.1.18 StrHlpr.UnicodeCopy Function

Pascal

```
function UnicodeCopy(const src: UnicodeString; index: Integer; count: Integer): UnicodeString;
```

C++

```
UnicodeString UnicodeCopy(const UnicodeString src, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeCopy.

32.1.19 StrHlpr.UnicodeDelete Function

Pascal

```
procedure UnicodeDelete(var dst: UnicodeString; index: Integer; count: Integer);
```

C++

```
UnicodeDelete(UnicodeString dst, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeDelete.

32.1.20 StrHlpr.UnicodeEqual Function

Pascal

```
function UnicodeEqual(const x: UnicodeString; const y: UnicodeString): Boolean;
```

C++

```
Boolean UnicodeEqual(const UnicodeString x, const UnicodeString y);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeEqual.

32.1.21 StrHlpr.UnicodeFree Function

Pascal

```
procedure UnicodeFree(var s: UnicodeString);
```

C++

```
UnicodeFree(UnicodeString s);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeFree.

32.1.22 StrHlpr.UnicodeFromAnsi Function

Pascal

```
procedure UnicodeFromAnsi(var dst: UnicodeString; const src: RawByteString);
```

C++

```
UnicodeFromAnsi(UnicodeString dst, const RawByteString src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeFromAnsi.

32.1.23 StrHlpr.UnicodeFromPChar Function

Pascal

```
procedure UnicodeFromPChar(var dst: UnicodeString; src: PAnsiChar);
```

C++

```
UnicodeFromPChar(UnicodeString dst, PAnsiChar src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeFromPChar.

32.1.24 StrHlpr.UnicodeFromPWideChar Function

Pascal

```
procedure UnicodeFromPWideChar(var dst: UnicodeString; src: PWideChar);
```

C++

```
UnicodeFromPWideChar(UnicodeString dst, PWideChar src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeFromPWideChar.

32.1.25 StrHlpr.UnicodeFromWide Function

Pascal

```
procedure UnicodeFromWide(var dst: UnicodeString; const src: WideString);
```

C++

```
UnicodeFromWide(UnicodeString dst, const BSTR src);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeFromWide.

32.1.26 StrHlpr.UnicodeGreater Function

Pascal

```
function UnicodeGreater(const x: UnicodeString; const y: UnicodeString): Boolean;
```

C++

```
Boolean UnicodeGreater(const UnicodeString x, const UnicodeString y);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeGreater.

32.1.27 StrHlpr.UnicodeInsert Function

Pascal

```
procedure UnicodeInsert(var dst: UnicodeString; const src: UnicodeString; index: Integer);
```

C++

```
UnicodeInsert(UnicodeString dst, const UnicodeString src, int index);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeInsert.

32.1.28 StrHlpr.UnicodeLess Function

Pascal

```
function UnicodeLess(const x: UnicodeString; const y: UnicodeString): Boolean;
```

C++

```
Boolean UnicodeLess(const UnicodeString x, const UnicodeString y);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeLess.

32.1.29 StrHlpr.UnicodePos Function

Pascal

```
function UnicodePos(const src: UnicodeString; const sub: UnicodeString): Integer;
```

C++

```
int UnicodePos(const UnicodeString src, const UnicodeString sub);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodePos.

32.1.30 StrHlpr.UnicodeSetLength Function

Pascal

```
procedure UnicodeSetLength(var dst: UnicodeString; len: Integer);
```

C++

```
UnicodeSetLength(UnicodeString dst, int len);
```

File

StrHlpr

Description

This is function StrHlpr.UnicodeSetLength.

32.1.31 StrHlpr.WideAppend Function

Pascal

```
procedure WideAppend(var dst: WideString; const src: WideString);
```

C++

```
WideAppend(BSTR dst, const BSTR src);
```

File

StrHlpr

Description

This is function StrHlpr.WideAppend.

32.1.32 StrHlpr.WideAssign Function

Pascal

```
procedure WideAssign(var dst: WideString; var src: WideString);
```

C++

```
WideAssign(BSTR dst, BSTR src);
```

File

StrHlpr

Description

This is function StrHlpr.WideAssign.

32.1.33 StrHlpr.WideCat Function

Pascal

```
function WideCat(const x: WideString; const y: WideString): WideString;
```

C++

```
BSTR WideCat(const BSTR x, const BSTR y);
```

File

StrHlpr

Description

This is function StrHlpr.WideCat.

32.1.34 StrHlpr.WideCopy Function

Pascal

```
function WideCopy(const src: WideString; index: Integer; count: Integer): WideString;
```

C++

```
BSTR WideCopy(const BSTR src, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.WideCopy.

32.1.35 StrHlpr.WideDelete Function

Pascal

```
procedure WideDelete(var dst: WideString; index: Integer; count: Integer);
```

C++

```
WideDelete(BSTR dst, int index, int count);
```

File

StrHlpr

Description

This is function StrHlpr.WideDelete.

32.1.36 StrHlpr.WideEqual Function

Pascal

```
function WideEqual(const x: WideString; const y: WideString): Boolean;
```

C++

```
Boolean WideEqual(const BSTR x, const BSTR y);
```

File

StrHlpr

Description

This is function StrHlpr.WideEqual.

32.1.37 StrHlpr.WideFree Function

Pascal

```
procedure WideFree(var s: WideString);
```

C++

```
WideFree(BSTR s);
```

File

StrHlpr

Description

This is function StrHlpr.WideFree.

32.1.38 StrHlpr.WideFromAnsi Function

Pascal

```
procedure WideFromAnsi(var dst: WideString; const src: RawByteString);
```

C++

```
WideFromAnsi(BSTR dst, const RawByteString src);
```

File

StrHlpr

Description

This is function StrHlpr.WideFromAnsi.

32.1.39 StrHlpr.WideFromPChar Function

Pascal

```
procedure WideFromPChar(var dst: WideString; src: PAnsiChar);
```

C++

```
WideFromPChar(BSTR dst, PAnsiChar src);
```

File

StrHlpr

Description

This is function StrHlpr.WideFromPChar.

32.1.40 StrHlpr.WideFromUnicode Function

Pascal

```
procedure WideFromUnicode(var dst: WideString; const src: UnicodeString);
```

C++

```
WideFromUnicode(BSTR dst, const UnicodeString src);
```

File

StrHlpr

Description

This is function StrHlpr.WideFromUnicode.

32.1.41 StrHlpr.WideGreater Function

Pascal

```
function WideGreater(const x: WideString; const y: WideString): Boolean;
```

C++

```
Boolean WideGreater(const BSTR x, const BSTR y);
```

File

StrHlpr

Description

This is function StrHlpr.WideGreater.

32.1.42 StrHlpr.WideInsert Function

Pascal

```
procedure WideInsert(var dst: WideString; const src: WideString; index: Integer);
```

C++

```
WideInsert(BSTR dst, const BSTR src, int index);
```

File

StrHlpr

Description

This is function StrHlpr.WideInsert.

32.1.43 StrHlpr.WideLength Function

Pascal

```
function WideLength(const src: WideString): Integer;
```

C++

```
int WideLength(const BSTR src);
```

File

StrHlpr

Description

This is function StrHlpr.WideLength.

32.1.44 StrHlpr.WideLess Function

Pascal

```
function WideLess(const x: WideString; const y: WideString): Boolean;
```

C++

```
Boolean WideLess(const BSTR x, const BSTR y);
```

File

StrHlpr

Description

This is function StrHlpr.WideLess.

32.1.45 StrHlpr.WidePos Function

Pascal

```
function WidePos(const src: WideString; const sub: WideString): Integer;
```

C++

```
int WidePos(const BSTR src, const BSTR sub);
```

File

StrHlpr

Description

This is function StrHlpr.WidePos.

32.1.46 StrHlpr.WideSetLength Function

Pascal

```
procedure WideSetLength(var dst: WideString; len: Integer);
```

C++

```
WideSetLength(BSTR dst, int len);
```

File

StrHlpr

Description

This is function StrHlpr.WideSetLength.

33 StrUtils Namespace

This is namespace StrUtils.

Topics

Name	Description
Functions (see page 2970)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3000)	The following table lists structs, records, enums in this documentation.
Types (see page 3001)	The following table lists types in this documentation.
Variables (see page 3003)	The following table lists variables in this documentation.
Constants (see page 3005)	The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 3005)	This is constant StrUtils.WordDelimiters.

Functions

	Name	Description
	AnsiContainsStr (see page 2971)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 2972)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 2972)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 2973)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 2973)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 2974)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 2974)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 2975)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 2975)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 2976)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 2977)	Replaces all occurrences of a substring with another string.
	AnsiReplaceText (see page 2977)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 2978)	Indicates whether two strings are similar.
	AnsiReverseString (see page 2978)	Returns reversed string.
	AnsiRightStr (see page 2979)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 2979)	Indicates whether one string is a (case-sensitive) prefix of another.
	AnsiStartsText (see page 2980)	Indicates whether one string is a (case-insensitive) prefix of another.

	ContainsStr (see page 2980)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 2981)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 2981)	Converts an integer representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 2982)	Converts a Word representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.
	DupeString (see page 2982)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 2983)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 2983)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 2984)	Conditionally returns one of two specified values.
	IndexStr (see page 2984)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 2985)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 2985)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 2986)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 2986)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 2987)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MatchText (see page 2987)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MidBStr (see page 2988)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 2988)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 2988)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 2989)	Returns the index value of a substring.
	RandomFrom (see page 2989)	Returns a randomly selected element from an array.
	ReplaceStr (see page 2990)	ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.
	ReplaceText (see page 2990)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 2991)	ResemblesText uses a Soundex (see page 2994) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 2994) method, and false otherwise.
	ReverseString (see page 2991)	Returns the reverse of a specified string.
	RightBStr (see page 2992)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 2992)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 2992)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 2993)	Locates a substring within a text buffer.
	SearchBuf (see page 2993)	Locates a substring within a text buffer.
	Soundex (see page 2994)	Converts a string into its Soundex representation.
	SoundexCompare (see page 2994)	Compares the Soundex (see page 2994) representations of two strings.
	SoundexInt (see page 2995)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 2996)	Indicates whether two strings are similar.
	SoundexSimilar (see page 2996)	Indicates whether two strings are similar.
	SoundexWord (see page 2997)	Converts a string into a Word that represents its phonetic value.

	StartsStr (see page 2998)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	StartsText (see page 2998)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 2999)	Inserts a substring into a specified position of a string, replacing the current characters.

Structs, Records, Enums

	Name	Description
	TStringSeachOption (see page 3000)	TStringSeachOption is an enumeration that describes the behavior of string searches.

Types

Name	Description
TCompareTextProc (see page 3001)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 3001)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 2995) function. The integer value can be from 1 to 8.
TSoundexLength (see page 3002)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 3002)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3000).

Variables

Name	Description
AnsiResemblesProc (see page 3003)	Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar.
ResemblesProc (see page 3003)	Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3003).

33.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiContainsStr (see page 2971)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 2972)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 2972)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 2973)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 2973)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 2974)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 2974)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 2975)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 2975)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 2976)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 2977)	Replaces all occurrences of a substring with another string.
	AnsiReplaceText (see page 2977)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 2978)	Indicates whether two strings are similar.
	AnsiReverseString (see page 2978)	Returns reversed string.
	AnsiRightStr (see page 2979)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 2979)	Indicates whether one string is a (case-sensitive) prefix of another.
	AnsiStartsText (see page 2980)	Indicates whether one string is a (case-insensitive) prefix of another.
	ContainsStr (see page 2980)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 2981)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 2981)	Converts an integer representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 2982)	Converts a Word representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.
	DupeString (see page 2982)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 2983)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 2983)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 2984)	Conditionally returns one of two specified values.
	IndexStr (see page 2984)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 2985)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 2985)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 2986)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 2986)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 2987)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MatchText (see page 2987)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

	MidBStr (see page 2988)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 2988)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 2988)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 2989)	Returns the index value of a substring.
	RandomFrom (see page 2989)	Returns a randomly selected element from an array.
	ReplaceStr (see page 2990)	ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.
	ReplaceText (see page 2990)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 2991)	ResemblesText uses a Soundex (see page 2994) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 2994) method, and false otherwise.
	ReverseString (see page 2991)	Returns the reverse of a specified string.
	RightBStr (see page 2992)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 2992)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 2992)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 2993)	Locates a substring within a text buffer.
	SearchBuf (see page 2993)	Locates a substring within a text buffer.
	Soundex (see page 2994)	Converts a string into its Soundex representation.
	SoundexCompare (see page 2994)	Compares the Soundex (see page 2994) representations of two strings.
	SoundexInt (see page 2995)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 2996)	Indicates whether two strings are similar.
	SoundexSimilar (see page 2996)	Indicates whether two strings are similar.
	SoundexWord (see page 2997)	Converts a string into a Word that represents its phonetic value.
	StartsStr (see page 2998)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	StartsText (see page 2998)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 2999)	Inserts a substring into a specified position of a string, replacing the current characters.

33.1.1 StrUtils.AnsiContainsStr Function

Indicates whether one string is a (case-sensitive) substring of another.

Pascal

```
function AnsiContainsStr(const AText: string; const ASubText: string): Boolean; overload;
```

C++

```
Boolean AnsiContainsStr(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

AnsiContainsStr returns true if the string specified by ASubText appears as a substring of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiResemblesText (see page 2978)

AnsiStartsStr (see page 2979)

AnsiEndsStr (see page 2972)

AnsiCompareStr

Pos

StringReplace

AnsiContainsText (see page 2972)

33.1.2 StrUtils.AnsiContainsText Function

Indicates whether one string is a (case-insensitive) substring of another.

Pascal

```
function AnsiContainsText(const AText: string; const ASubText: string): Boolean; overload;
```

C++

```
Boolean AnsiContainsText(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

AnsiContainsText returns true if the string specified by ASubText appears as a substring of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiResemblesText (see page 2978)

AnsiStartsText (see page 2980)

AnsiEndsText (see page 2973)

AnsiCompareText

Pos

StringReplace

AnsiContainsStr (see page 2971)

33.1.3 StrUtils.AnsiEndsStr Function

Indicates whether one string is a (case-sensitive) suffix of another.

Pascal

```
function AnsiEndsStr(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiEndsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiEndsStr returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiStartsStr (see page 2979)

AnsiContainsStr (see page 2971)

Pos

RightStr (see page 2992)

AnsiEndsText (see page 2973)

33.1.4 StrUtils.AnsiEndsText Function

Indicates whether one string is a (case-insensitive) suffix of another.

Pascal

```
function AnsiEndsText(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiEndsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiEndsText returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiStartsText (see page 2980)

AnsiContainsText (see page 2972)

Pos

RightStr (see page 2992)

33.1.5 StrUtils.AnsiIndexStr Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexStr(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int AnsiIndexStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexStr returns the 0-offset index of the first string it finds in AValues that matches AText case-sensitively. If the string specified by AText does not have an exact match in AValues, AnsiIndexStr returns –1.

See Also

AnsiMatchStr (see page 2975)

RandomFrom (see page 2989)

AnsiIndexText (see page 2974)

33.1.6 StrUtils.AnsiIndexText Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexText(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int AnsiIndexText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexText returns the 0-offset index of the first string it finds in AValues that matches AText case-insensitively. If the string specified by AText does not have a (possibly case-insensitive) match in AValues, AnsiIndexText returns –1. Comparisons are based on the current system locale.

See Also

AnsiMatchText (see page 2975)

RandomFrom (see page 2989)

AnsiIndexStr (see page 2973)

33.1.7 StrUtils.AnsiLeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function AnsiLeftStr(const AText: string; const ACount: Integer): string; overload;
```

C++

```
UnicodeString AnsiLeftStr(const UnicodeString AText, const int ACount);
```

File

StrUtils

Description

AnsiLeftStr returns the leading characters of AText up to a length of ACount characters.

See Also

Copy

TrimLeft

AnsiStartsText (see page 2980)

AnsiStartsStr (see page 2979)

LeftStr (see page 2986)

LeftBStr (see page 2985)

33.1.8 StrUtils.AnsiMatchStr Function

Indicates whether an array of strings contains an exact match to a specified string.

Pascal

```
function AnsiMatchStr(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean AnsiMatchStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchStr to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-sensitive.

See Also

AnsiIndexStr (see page 2973)

AnsiSameStr

AnsiMatchText (see page 2975)

33.1.9 StrUtils.AnsiMatchText Function

Indicates whether an array of strings contains a case-insensitive match to a specified string.

Pascal

```
function AnsiMatchText(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean AnsiMatchText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchText to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-insensitive and use the current system locale.

AText is the string to locate.

AValues is the array of strings to search.

Note: AValues_Size is the index of the last element in AValues (one less than the number of elements).

AnsiMatchText returns true if AText (or a string that matches except for case) appears in AValues, false if it does not appear in the array.

See Also

AnsiIndexText (see page 2974)

AnsiSameText

ANsiMatchStr

33.1.10 StrUtils.AnsiMidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function AnsiMidStr(const AText: string; const AStart: Integer; const ACount: Integer):
string; overload;
```

C++

```
UnicodeString AnsiMidStr(const UnicodeString AText, const int AStart, const int ACount);
```

File

StrUtils

Description

AnsiMidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, AnsiMidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

See Also

Copy

LeftStr (see page 2986)

RightStr (see page 2992)

MidStr (see page 2988)

MidBStr (see page 2988)

33.1.11 StrUtils.AnsiReplaceStr Function

Replaces all occurrences of a substring with another string.

Pascal

```
function AnsiReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string; overload;
```

C++

```
UnicodeString AnsiReplaceStr(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

Call AnsiReplaceStr to scan the string specified by AText and replace all occurrences of the string specified by AFromText with the string specified by AToText. Occurrences of AFromText are identified by a case-sensitive comparison. The identification of substrings is not recursive: if the substitution of AToText results in a new match for AFromText, that match is not replaced.

See Also

AnsiContainsStr (see page 2971)

AnsiReplaceText (see page 2977)

ANsiMatchStr

StringReplace

SearchBuf (see page 2993)

StuffString (see page 2999)

33.1.12 StrUtils.AnsiReplaceText Function

Replaces all case-insensitive matches of a substring with another string.

Pascal

```
function AnsiReplaceText(const AText: string; const AFromText: string; const AToText: string):
string; overload;
```

C++

```
UnicodeString AnsiReplaceText(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

Call AnsiReplaceText to scan the string specified by AText and replace all occurrences of the string specified by AFromText with the string specified by AToText. Occurrences of AFromText are identified by a case-insensitive comparison that uses the current system locale. The identification of substrings is not recursive: if the substitution of AToText results in a new match for

AFromText, that match is not replaced.

See Also

AnsiContainsText (see page 2972)

AnsiReplaceStr (see page 2977)

StringReplace

SearchBuf (see page 2993)

StuffString (see page 2999)

33.1.13 StrUtils.AnsiResemblesText Function

Indicates whether two strings are similar.

Pascal

```
function AnsiResemblesText(const AText: string; const AOther: string): Boolean; overload;
```

C++

```
Boolean AnsiResemblesText(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

Description

AnsiResemblesText returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

The algorithm that controls whether AnsiResemblesText considers two strings similar is specified by the AnsiResemblesProc (see page 3003) variable. By default, AnsiResemblesProc (see page 3003) is set to SoundExProc, which converts both strings to a phonetic representation using a SoundEx algorithm and compares up to 4 characters of the resulting phonetic representation. By changing the function assigned to AnsiResemblesProc (see page 3003), you can change the comparison made by AnsiResemblesText.

See Also

AnsiResemblesProc (see page 3003)

SoundExProc

33.1.14 StrUtils.AnsiReverseString Function

Returns reversed string.

Pascal

```
function AnsiReverseString(const AText: string): string; overload;
```

C++

```
UnicodeString AnsiReverseString(const UnicodeString AText);
```

File

StrUtils

Description

AnsiReverseString returns AText with its characters reversed.

33.1.15 StrUtils.AnsiRightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function AnsiRightStr(const AText: string; const ACount: Integer): string; overload;
```

C++

```
UnicodeString AnsiRightStr(const UnicodeString AText, const int ACount);
```

File

StrUtils

Description

AnsiRightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, AnsiRightStr returns the string "grammer".

See Also

Copy

TrimRight

AnsiEndsText (see page 2973)

AnsiEndsStr (see page 2972)

RightStr (see page 2992)

RightBStr (see page 2992)

33.1.16 StrUtils.AnsiStartsStr Function

Indicates whether one string is a (case-sensitive) prefix of another.

Pascal

```
function AnsiStartsStr(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiStartsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiStartsStr returns true if the string specified by ASubText appears as the leading part of the string specified by AText.

Comparison is performed case-sensitively.

See Also

AnsiEndsStr (see page 2972)

AnsiContainsStr (see page 2971)

Pos

LeftStr (see page 2986)

AnsiStartsText (see page 2980)

33.1.17 StrUtils.AnsiStartsText Function

Indicates whether one string is a (case-insensitive) prefix of another.

Pascal

```
function AnsiStartsText(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiStartsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiStartsText returns true if the string specified by ASubText appears as the leading part of the string specified by AText. Comparison is performed case-insensitively and uses the current system locale.

See Also

AnsiEndsText (see page 2973)

AnsiContainsText (see page 2972)

Pos

LeftStr (see page 2986)

AnsiStartsStr (see page 2979)

33.1.18 StrUtils.ContainsStr Function

Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsStr(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsStr(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

For a case insensitive search, use the ContainsText (see page 2981) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsText (see page 2981)

33.1.19 StrUtils.ContainsText Function

Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsText(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsText(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

For a case sensitive search, use the ContainsStr (see page 2980) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsStr (see page 2980)

33.1.20 StrUtils.DecodeSoundexInt Function

Converts an integer representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexInt(AValue: Integer): string;
```

C++

```
UnicodeString DecodeSoundexInt(int AValue);
```

File

StrUtils

Description

Call DecodeSoundexInt to convert the integer representation of a string returned by the SoundexInt (see page 2995) function into the corresponding Soundex (see page 2994) string representation.

A Soundex (see page 2994) representation of a string is a phonetic encoding where each character of the Soundex (see page 2994) representation represents a family of similar phonemes.

See Also

SoundexInt (see page 2995)

DecodeSoundexWord (see page 2982)

Soundex (see page 2994)

33.1.21 StrUtils.DecodeSoundexWord Function

Converts a Word representation of a Soundex (see page 2994) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexWord(AValue: Word): string;
```

C++

```
UnicodeString DecodeSoundexWord(Word AValue);
```

File

StrUtils

Description

Call DecodeSoundexWord to convert the unsigned 16-bit representation of a string returned by the SoundexWord (see page 2997) function into the corresponding Soundex (see page 2994) string representation.

A Soundex (see page 2994) representation of a string is a phonetic encoding where each character of the Soundex (see page 2994) representation represents a family of similar phonemes.

See Also

SoundexWord (see page 2997)

DecodeSoundexInt (see page 2981)

Soundex (see page 2994)

33.1.22 StrUtils.DupeString Function

Returns the concatenation of a string with itself a specified number of repeats.

Pascal

```
function DupeString(const AText: string; ACount: Integer): string; overload;
```

C++

```
UnicodeString DupeString(const UnicodeString AText, int ACount);
```

File

StrUtils

Description

DupeString returns a string containing ACount repeats of the string specified by AText. For example,

S := DupeString('Ha', 5);

sets S to the string 'HaHaHaHaHa'.

See Also

StringOfChar

StringReplace

33.1.23 StrUtils.EndsStr Function

Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the EndsText (see page 2983) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsText (see page 2983)

33.1.24 StrUtils.EndsText Function

Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the EndsStr (see page 2983) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsStr (see page 2983)

33.1.25 StrUtils.IfThen Function

Conditionally returns one of two specified values.

Pascal

```
function IfThen(AValue: Boolean; const ATrue: string; AFalse: string = ''): string; overload;
```

C++

```
UnicodeString IfThen(Boolean AValue, const UnicodeString ATrue, UnicodeString AFalse = '');
```

File

StrUtils

Description

IfThen checks the expression passed as AValue and returns ATrue if it evaluates to true, or AFalse if it evaluates to false. In Delphi, if the AFalse parameter is omitted, IfThen returns 0 or an empty string when AValue evaluates to False.

33.1.26 StrUtils.IndexStr Function

IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexStr(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int IndexStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive comparison, use the IndexText (see page 2985) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexText (see page 2985)

33.1.27 StrUtils.IndexText Function

IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexText(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int IndexText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the IndexStr (see page 2984) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexStr (see page 2984)

33.1.28 StrUtils.LeftBStr Function

Returns the substring of a specified number of bytes that appears at the start of a string.

Pascal

```
function LeftBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString LeftBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

LeftBStr returns the leading bytes of AText up to a length of AByteCount bytes.

Note: LeftBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use LeftStr (see page 2986).

See Also

Copy

TrimLeft

AnsiStartsText (see page 2980)

AnsiStartsStr (see page 2979)

AnsiLeftStr (see page 2974)

LeftStr (see page 2986)

33.1.29 StrUtils.LeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function LeftStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function LeftStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString LeftStr(const AnsiString AText, const int ACount);
BSTR LeftStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

LeftStr returns the leading characters of AText up to a length of ACount characters.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, LeftStr may return more than ACount bytes. To return a specific number of bytes, use LeftBStr (see page 2985).

See Also

Copy

TrimLeft

AnsiStartsText (see page 2980)

AnsiStartsStr (see page 2979)

AnsiLeftStr (see page 2974)

LeftBStr (see page 2985)

33.1.30 StrUtils.MatchStr Function

MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchStr(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean MatchStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive match, use the MatchText (see page 2987) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchText (see page 2987)

33.1.31 StrUtils.MatchText Function

MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchText(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean MatchText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the MatchStr (see page 2987) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchStr (see page 2987)

33.1.32 StrUtils.MidBStr Function

Returns the substring of a specified number of bytes that appears at a specified position in a string.

Pascal

```
function MidBStr(const AText: AnsiString; const AByteStart: Integer; const AByteCount:
Integer): AnsiString;
```

C++

```
AnsiString MidBStr(const AnsiString AText, const int AByteStart, const int AByteCount);
```

File

StrUtils

Description

MidBStr returns a substring AByteCount bytes beginning at AByteStart bytes.

If AByteStart is larger than the length of AText in bytes, MidBStr returns an empty string.

If AByteCount specifies more bytes than are available, only the available bytes are returned.

Note: MidBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use MidStr (see page 2988).

See Also

Copy

LeftStr (see page 2986)

RightStr (see page 2992)

AnsiMidStr (see page 2976)

MidStr (see page 2988)

33.1.33 StrUtils.MidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function MidStr(const AText: AnsiString; const AStart: Integer; const ACount: Integer):
AnsiString; overload;
function MidStr(const AText: WideString; const AStart: Integer; const ACount: Integer):
WideString; overload;
```

C++

```
AnsiString MidStr(const AnsiString AText, const int AStart, const int ACount);
BSTR MidStr(const BSTR AText, const int AStart, const int ACount);
```

File

StrUtils

Description

MidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, MidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, MidStr may return more than ACount bytes. To return a specific number of bytes, use MidBStr (see page 2988).

See Also

Copy

LeftStr (see page 2986)

RightStr (see page 2992)

AnsiMidStr (see page 2976)

MidBStr (see page 2988)

33.1.34 StrUtils.PosEx Function

Returns the index value of a substring.

Pascal

```
function PosEx(const SubStr: string; const S: string; Offset: Integer = 1): Integer; overload;
```

C++

```
int PosEx(const UnicodeString SubStr, const UnicodeString S, int Offset = 1);
```

File

StrUtils

Description

PosEx returns the index of SubStr in S, beginning the search at Offset. If Offset is 1 (default), PosEx is equivalent to Pos (see page 3720).

PosEx returns 0 if SubStr is not found, if Offset is greater than the length of S, or if Offset is less than 1.

See Also

Pos

33.1.35 StrUtils.RandomFrom Function

Returns a randomly selected element from an array.

Pascal

```
function RandomFrom(const AValues: array of string): string; overload;
```

C++

```
UnicodeString RandomFrom(const array of string AValues);
```

File

StrUtils

Description

RandomFrom returns a randomly selected element from the array passed as the AValues argument. In C++, AValues_Size is the index of the last array element (one less than the number of elements.)

To initialize the random number generator, add a single call Randomize (see page 3726) or assign a value to the RandSeed (see page 3902) variable before making any calls to RandomFrom.

See Also

Randomize

RandSeed

33.1.36 StrUtils.ReplaceStr Function

ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.

Pascal

```
function ReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
UnicodeString ReplaceStr(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

For a case insensitive replacement, use the ReplaceText (see page 2990) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ReplaceText (see page 2990)

33.1.37 StrUtils.ReplaceText Function

ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.

Pascal

```
function ReplaceText(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
UnicodeString ReplaceText(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

For a case sensitive replacement, use the ReplaceStr (see page 2990) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ReplaceStr (see page 2990)

33.1.38 StrUtils.ResemblesText Function

ResemblesText uses a Soundex (see page 2994) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 2994) method, and false otherwise.

Pascal

```
function ResemblesText(const AText: string; const AOther: string): Boolean; overload;
```

C++

```
Boolean ResemblesText(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

See Also

Commonly Used Long String Routines

String Types

String Constants

AnsiResemblesText (see page 2978)

33.1.39 StrUtils.ReverseString Function

Returns the reverse of a specified string.

Pascal

```
function ReverseString(const AText: string): string; overload;
```

C++

```
UnicodeString ReverseString(const UnicodeString AText);
```

File

StrUtils

Description

ReverseString returns the string specified by AText with the characters in reverse order.

See Also

DupeString (see page 2982)

33.1.40 StrUtils.RightBStr Function

Returns the substring of a specified number of bytes that appears at the end of a string.

Pascal

```
function RightBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString RightBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

RightBStr returns the trailing bytes of AText up to a length of AByteCount bytes.

Note: RightBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use RightStr (see page 2992).

See Also

Copy

TrimRight

AnsiEndsText (see page 2973)

AnsiEndsStr (see page 2972)

AnsiRightStr (see page 2979)

33.1.41 StrUtils.RightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function RightStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function RightStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString RightStr(const AnsiString AText, const int ACount);
BSTR RightStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

RightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, RightStr returns the string "grammer".

Note: If AText is an AnsiString and the current locale uses multi-byte characters, RightStr may return more than ACount bytes. To return a specific number of bytes, use RightBStr (see page 2992).

See Also

Copy

TrimRight

AnsiEndsText (see page 2973)

AnsiEndsStr (see page 2972)

AnsiRightStr (see page 2979)

33.1.42 StrUtils.SearchBuf Function

Locates a substring within a text buffer.

Pascal

```
function SearchBuf(Buf: PAnsiChar; BufLen: Integer; SelStart: Integer; SelLength: Integer;
SearchString: AnsiString; Options: TStringSearchOptions = [soDown]): PAnsiChar; overload;
function SearchBuf(Buf: PWideChar; BufLen: Integer; SelStart: Integer; SelLength: Integer;
SearchString: UnicodeString; Options: TStringSearchOptions = [soDown]): PWideChar; overload;
```

C++

```
PAnsiChar SearchBuf(PAnsiChar Buf, int BufLen, int SelStart, int SelLength, AnsiString
SearchString, TStringSearchOptions Options = [soDown]);
PWideChar SearchBuf(PWideChar Buf, int BufLen, int SelStart, int SelLength, UnicodeString
SearchString, TStringSearchOptions Options = [soDown]);
```

File

StrUtils

Description

Call SearchBuf to search for a specified search string within a text buffer.

Buf is the text buffer to search.

BufLen is the length, in bytes, of Buf.

SelStart is the first character of the search when Options indicates a backward search (does not include soDown). The first character in Buf has position 0.

SelLength is the number of characters after SelStart that the search begins when Options indicates a forward search (includes soDown).

SearchString is the string to find in Buf.

Options determines whether the search runs forward (soDown) or backward from SelStart or SelStart+SelLength, whether the search is case sensitive (soMatchCase), and whether the matching string must be a whole word (soWholeWord).

If SearchBuf finds a match, it returns a pointer to the first character of the matching string in Buf. If it does not find a match, SearchBuf returns nil (Delphi) or NULL (C++).

See Also

TFindDialog

33.1.43 StrUtils.Soundex Function

Converts a string into its Soundex representation.

Pascal

```
function Soundex(const AText: string; ALength: TSoundexLength = 4): string;
```

C++

```
UnicodeString Soundex(const UnicodeString AText, TSoundexLength ALength = 4);
```

File

StrUtils

Description

Soundex converts the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It returns up to the first ALength Soundex codes in the resulting string representation.

See Also

SoundexWord (see page 2997)

SoundexInt (see page 2995)

SoundexSimilar (see page 2996)

SoundexCompare (see page 2994)

SoundexProc (see page 2996)

33.1.44 StrUtils.SoundexCompare Function

Compares the Soundex (see page 2994) representations of two strings.

Pascal

```
function SoundexCompare(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Integer;
```

C++

```
int SoundexCompare(const UnicodeString AText, const UnicodeString AOther, TSoundexLength
ALength = 4);
```

File

StrUtils

Description

SoundexCompare uses a Soundex (see page 2994) algorithm to convert the strings specified by AText and AOther into

phonetic representations where each character of the resulting string represents one of six families of similar phonemes. It then compares the first ALength characters of the resulting representations by calling AnsiCompareStr (see page 4114).

SoundexCompare returns 0 if the first ALength characters of the phonetic representations are identical, a value less then 0 if the representation of AText is less than the representation of AOther, and a value greater than 0 if the representation of AText is greater than the representation of AOther.

See Also

SoundexSimilar (see page 2996)

SoundexWord (see page 2997)

SoundexInt (see page 2995)

AnsiCompareStr

Soundex (see page 2994)

SoundexProc (see page 2996)

33.1.45 StrUtils.SoundexInt Function

Converts a string into an integer that represents its phonetic value.

Pascal

```
function SoundexInt(const AText: string; ALength: TSoundexIntLength = 4): Integer;
```

C++

```
int SoundexInt(const UnicodeString AText, TSoundexIntLength ALength = 4);
```

File

StrUtils

Description

SoundexInt uses a Soundex (see page 2994) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an integer that uniquely represents the first ALength Soundex (see page 2994) codes in that string representation.

Note: The TSoundexIntLength (see page 3001) type represents an integer between 1 and 8. This is because the size of the integer return value limits the number of Soundex (see page 2994) codes that can be used in the final representation.

See Also

SoundexSimilar (see page 2996)

SoundexWord (see page 2997)

DecodeSoundexInt (see page 2981)

SoundexCompare (see page 2994)

Soundex (see page 2994)

SoundexProc (see page 2996)

33.1.46 StrUtils.SoundexProc Function

Indicates whether two strings are similar.

Pascal

```
function SoundexProc(const AText: string; const AOther: string): Boolean;
```

C++

```
Boolean SoundexProc(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

Description

SoundexProc returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar. SoundexProc is the default value of the AnsiResemblesProc (see page 3003) variable, which provides the underlying implementation of AnsiResemblesText (see page 2978).

SoundexProc uses a Soundex (see page 2994) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to 4 digits in the resulting representation, returning true if they are all the same and false if they are not.

Note: SoundexProc is the same as the SoundexSimilar (see page 2996) function using an ALength value of 4.

See Also

AnsiResemblesProc (see page 3003)

SoundexSimilar (see page 2996)

SoundexCompare (see page 2994)

SoundexWord (see page 2997)

SOundexInt

Soundex (see page 2994)

AnsiResemblesText (see page 2978)

33.1.47 StrUtils.SoundexSimilar Function

Indicates whether two strings are similar.

Pascal

```
function SoundexSimilar(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Boolean;
```

C++

```
Boolean SoundexSimilar(const UnicodeString AText, const UnicodeString AOther, TSoundexLength
ALength = 4);
```

File

StrUtils

Description

SoundexSimilar returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

SoundexSimilar uses a Soundex (see page 2994) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to ALength digits in the resulting representation, returning true if they are all the same and false if they are not.

See Also

SOundexCompare

SoundexWord (see page 2997)

SoundexInt (see page 2995)

Soundex (see page 2994)

SoundexProc (see page 2996)

33.1.48 StrUtils.SoundexWord Function

Converts a string into a Word that represents its phonetic value.

Pascal

```
function SoundexWord(const AText: string): Word;
```

C++

```
Word SoundexWord(const UnicodeString AText);
```

File

StrUtils

Description

SoundexWord uses a Soundex (see page 2994) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an unsigned 16-bit value that uniquely represents the first 4 Soundex (see page 2994) codes in that string representation.

See Also

SoundexSimilar (see page 2996)

SoundexInt (see page 2995)

DecodeSoundexWord (see page 2982)

SoundexCompare (see page 2994)

Soundex (see page 2994)

SoundexProc (see page 2996)

33.1.49 StrUtils.StartsStr Function

Enter a short sentence, starting with a verb, that states the basic purpose of the property.

Pascal

```
function StartsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the StartsText (see page 2998) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsText (see page 2998)

33.1.50 StrUtils.StartsText Function

StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.

Pascal

```
function StartsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the StartsStr (see page 2998) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsStr (see page 2998)

33.1.51 StrUtils.StuffString Function

Inserts a substring into a specified position of a string, replacing the current characters.

Pascal

```
function StuffString(const AText: string; AStart: Cardinal; ALength: Cardinal; const ASubText:
string): string; overload;
```

C++

```
UnicodeString StuffString(const UnicodeString AText, unsigned AStart, unsigned ALength, const
UnicodeString ASubText);
```

File

StrUtils

Description

StuffString returns a new string that is constructed by replacing the characters of AText specified by AStart and ALength with the string specified by ASubText.

AText is the string to which ASubText is added. It is not modified by StuffString.

AStart is the index of the first character in AText to be replaced, where 1 identifies the first character, 2 identifies the second character, and so on.

ALength is the number of characters in AText that are replaced by ASubText. If ALength is 0, ASubText is inserted into AText without removing any characters. If AText is negative, the characters preceding the insertion point are duplicated after the occurrence of ASubText in the result string.

ASubText is the substring to insert into AText, replacing the characters specified by AStart and ALength.

See Also

AnsiReplaceStr (see page 2977)

AnsiReplaceText (see page 2977)

StringReplace

33.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStringSeachOption (see page 3000)	TStringSeachOption is an enumeration that describes the behavior of string searches.

33.2.1 StrUtils.TStringSeachOption Enumeration

TStringSeachOption is an enumeration that describes the behavior of string searches.

Pascal

```
TStringSeachOption = (
  soDown,
  soMatchCase,
  soWholeWord
);
```

C++

```
enum TStringSeachOption {
  soDown,
  soMatchCase,
  soWholeWord
};
```

File

StrUtils

Description

Note the unusual spelling. TStringSeachOptioncan have one of the following values:

Value	Meaning
soDown	A forward search is performed through the target text from the specified start point to the end of the search buffer. If soDown is not specified a reverse search is performed from the start point to the begining of the search buffer.
soMatchCase	Performs a case sensitive search.
soWholeWord	Match whole words, not pieces of words. Whole words are determined using standard identifier parser rules.

See Also

Commonly Used Long String Routines

String Types

String Constants

TStringSearchOptions (see page 3002)

33.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCompareTextProc (see page 3001)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 3001)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 2995) function. The integer value can be from 1 to 8.
TSoundexLength (see page 3002)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 3002)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3000).

33.3.1 StrUtils.TCompareTextProc Type

TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.

Pascal

```
TCompareTextProc = function (const AText, AOther: string): Boolean;
```

C++

```
(const AText, AOther: string): Boolean ( TCompareTextProc)();
```

File

StrUtils

See Also

TStringSeachOption (see page 3000)

33.3.2 StrUtils.TSoundexIntLength Type

TSoundexIntLength The maximum calculation length for the SoundexInt (see page 2995) function. The integer value can be from 1 to 8.

Pascal

```
TSoundexIntLength = 1..8;
```

C++

```
1..8 TSoundexIntLength;
```

File

StrUtils

Description

This value determines how many characters to use when generating the soundex string from the source string. There is a maximum of 8 characters due to the maximum capacity of the Integer type.

See Also

SoundexLength

33.3.3 StrUtils.TSoundexLength Type

TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.

Pascal

```
TSoundexLength = 1..MaxInt;
```

C++

```
1..MaxInt TSoundexLength;
```

File

StrUtils

See Also

SoundexIntLength

33.3.4 StrUtils.TStringSearchOptions Type

TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3000).

Pascal

```
TStringSearchOptions = set of TStringSeachOption;
```

C++

```
TStringSeachOption TStringSearchOptions;
```

File

StrUtils

Description

Note the unusual spelling of the enumeration.

See Also

TStringSeachOption (see page 3000)

33.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AnsiResemblesProc (see page 3003)	Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar.
ResemblesProc (see page 3003)	Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3003).

33.4.1 StrUtils.AnsiResemblesProc Variable

Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar.

Pascal

```
AnsiResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc AnsiResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 2978) function uses the function assigned to AnsiResemblesProc to determine when two strings should be considered similar. By default, AnsiResemblesProc is set to SoundExProc.

To make AnsiResemblesText (see page 2978) use a different algorithm, you can change AnsiResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundExSimilar than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 2978)

SoundExSimilar

33.4.2 StrUtils.ResemblesProc Variable

Controls the algorithm used by AnsiResemblesText (see page 2978) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3003).

Pascal

```
ResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc ResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 2978) function uses the function assigned to ResemblesProc to determine when two strings should be considered similar. By default, ResemblesProc is set to SoundExProc.

To make AnsiResemblesText (see page 2978) use a different algorithm, you can change ResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundExSimilar than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 2978)

AnsiResemblesProc (see page 3003)

SoundExSimilar

33.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 3005)	This is constant StrUtils.WordDelimiters.

33.5.1 StrUtils.WordDelimiters Constant

Pascal

```
WordDelimiters: set of AnsiChar = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

C++

```
set of AnsiChar WordDelimiters = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

File

StrUtils

Description

This is constant StrUtils.WordDelimiters.

34 SvcMgr Namespace

This is namespace SvcMgr.

Topics

Name	Description
Classes (see page 3009)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3048)	The following table lists structs, records, enums in this documentation.
Types (see page 3052)	The following table lists types in this documentation.
Variables (see page 3055)	The following table lists variables in this documentation.
Constants (see page 3056)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDependencies (see page 3009)	TDependencies is a container for TDependency (see page 3012) objects.
	TDependency (see page 3012)	TDependency object represents a dependent service or load ordering group used by TService (see page 3017) objects.
	TEventLogger (see page 3014)	This is class SvcMgr.TEventLogger.
	TService (see page 3017)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 3035)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 3041)	TServiceThread represents the thread associated with a specific service (TService (see page 3017) object).

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 3056)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

Structs, Records, Enums

	Name	Description
	TCurrentStatus (see page 3048)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 3049)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 3049)	TServiceType indicates the type of a service.
	TStartType (see page 3050)	TStartType indicates how a service starts up.

Types

Name	Description
TContinueEvent (see page 3052)	This is type SvcMgr.TContinueEvent.

TPauseEvent (see page 3052)	This is type SvcMgr.TPauseEvent.
TServiceController (see page 3053)	This is type SvcMgr.TServiceController.
TServiceEvent (see page 3053)	This is type SvcMgr.TServiceEvent.
TStartEvent (see page 3053)	This is type SvcMgr.TStartEvent.
TStopEvent (see page 3054)	This is type SvcMgr.TStopEvent.

Variables

Name	Description
Application (see page 3055)	This is variable SvcMgr.Application.

34.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDependencies (see page 3009)	TDependencies is a container for TDependency (see page 3012) objects.
	TDependency (see page 3012)	TDependency object represents a dependent service or load ordering group used by TService (see page 3017) objects.
	TEventLogger (see page 3014)	This is class SvcMgr.TEventLogger.
	TService (see page 3017)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 3035)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 3041)	TServiceThread represents the thread associated with a specific service (TService (see page 3017) object).

34.1.1 TDependencies Class

TDependencies is a container for TDependency (see page 3012) objects.

Class Hierarchy

Pascal

```
TDependencies = class(TCollection);
```

C++

```
class TDependencies : public TCollection;
```

File

SvcMgr

Description

TDependencies is a collection of TDependency (see page 3012) objects, each of which represents a dependent service or load ordering group for a service (TService (see page 3017) instance). TDependencies objects are the implementation of the Dependencies property of TService (see page 3017).

See Also

TDependency (see page 3012)

TService (see page 3017)

TService (see page 3017)

TService (see page 3017)

34.1.1.1 TDependencies Members

The following tables list the members exposed by TDependencies.

Protected Methods

	Name	Description
	GetOwner (see page 3010)	Returns the owner of an object.

Public Methods

	Name	Description
	Create (see page 3010)	Creates an instance of a TDependencies (see page 3009) object.

Public Properties

	Name	Description
	Items (see page 3011)	Lists the dependency objects in the dependencies collection.

34.1.1.2 TDependencies.Create Constructor

Creates an instance of a TDependencies (see page 3009) object.

Pascal

```
constructor Create(Owner: TPersistent);
```

C++

```
__fastcall TDependencies(TPersistent * Owner);
```

Description

Applications should not need to call the constructor directly. Instances of TService (see page 3017) create the TDependencies (see page 3009) object from their own constructors. The Owner parameter is the TService (see page 3017) instance that uses the TDependencies (see page 3009) object to implement its Dependencies property.

Create sets the TDependency (see page 3012) owner to the Owner parameter and calls the inherited Create.

See Also

Create

TService (see page 3017)

34.1.1.3 TDependencies Methods

The methods of the TDependencies class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 3010)	Returns the owner of an object.

34.1.1.3.1 TDependencies.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

34.1.1.4 TDependencies Properties

The properties of the TDependencies class are listed here.

Public Properties

	Name	Description
	Items (see page 3011)	Lists the dependency objects in the dependencies collection.

34.1.1.4.1 TDependencies.Items Property

Lists the dependency objects in the dependencies collection.

Pascal

```
property Items [Index: Integer]: TDependency;
```

C++

```
__property TDependency Items[int Index];
```

Description

The value of the Index parameter corresponds to the Index property of each TDependency (see page 3012) object in the collection. It represents the position of the dependency object in the dependencies collection.

Note: In Delphi, Items is the default property of TDependencies (see page 3009). This means that you can omit the property names. For example

Note: Service1.Dependencies.Items[i]

Note: can be written

Note: Service1.Dependencies[i]

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

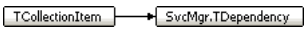
Index

operator_[]

34.1.2 TDependency Class

TDependency object represents a dependent service or load ordering group used by TService (see page 3017) objects.

Class Hierarchy



Pascal

```
TDependency = class(TCollectionItem);
```

C++

```
class TDependency : public TCollectionItem;
```

File

SvcMgr

Description

A TDependency object represents a new service or load ordering group that must start before the service represented by a TService (see page 3017) object. Collections of dependency objects are TDependencies (see page 3009) objects. TService (see page 3017) has Dependencies and LoadGroup properties that use dependency collections. Each dependency has a Name (see page 3014) property that identifies the service or load ordering group, and a boolean IsGroup (see page 3013) property to indicate whether the dependency represents a load ordering group.

See Also

TDependencies (see page 3009)

TService (see page 3017)

TService (see page 3017)

Service Name Properties

34.1.2.1 TDependency Members

The following tables list the members exposed by TDependency.

Protected Methods

	Name	Description
	GetDisplayName (see page 3013)	Returns the name of the collection item as it appears in the collection editor.

Published Properties

	Name	Description
	IsGroup (see page 3013)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 3017) object).
	Name (see page 3014)	Identifies the dependent service or load ordering group used by a TService (see page 3017) object.

34.1.2.2 TDependency Methods

The methods of the TDependency class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 3013)	Returns the name of the collection item as it appears in the collection editor.

34.1.2.2.1 TDependency.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

34.1.2.3 TDependency Properties

The properties of the TDependency class are listed here.

Published Properties

	Name	Description
	IsGroup (see page 3013)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 3017) object).
	Name (see page 3014)	Identifies the dependent service or load ordering group used by a TService (see page 3017) object.

34.1.2.3.1 TDependency.IsGroup Property

Indicates whether the dependency represents a load ordering group for a service (TService (see page 3017) object).

Pascal

```
property IsGroup: Boolean;
```

C++

```
__property Boolean IsGroup;
```

Description

IsGroup indicates whether the dependency represents a load ordering group.

Note: : Because services and service groups share the same name space, group names must be prefixed by the SC_GROUP_IDENTIFIER character to distinguish them from service names. If IsGroup is set to true, SC_GROUP_IDENTIFIER will automatically be prefixed to the Dependency name (Name (see page 3014) property). DisplayName is not modified.

See Also

Name (see page 3014)

LoadGroup (see page 3026)

34.1.2.3.2 TDependency.Name Property

Identifies the dependent service or load ordering group used by a TService (see page 3017) object.

Pascal

```
property Name: String;
```

C++

```
__property UnicodeString Name;
```

Description

Name is the name identifying the service or load ordering group. If Name is the name of a service, Name reflects the actual ServiceStartName of a TService (see page 3017) instance.

The property editor for the Dependencies property of TService (see page 3017) technically uses the dependency's DisplayName to identify the dependency service in its collection list. However, Name and DisplayName are nearly always the same.

Dependency objects that are load ordering groups are prefixed by SC_GROUP_IDENTIFIER. Setting the IsGroup (see page 3013) property handles prefixing the Name automatically.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

DisplayName

LoadGroup (see page 3026)

TService (see page 3017)

34.1.3 TEventLogger Class

Class Hierarchy

Pascal

```
TEventLogger = class(TObject);
```

C++

```
class TEventLogger : public TObject;
```

File

SvcMgr

Description

This is class SvcMgr.TEventLogger.

34.1.3.1 TEventLogger Members

The following tables list the members exposed by TEventLogger.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3015)	This is Create, a member of class TEventLogger.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3016)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	LogMessage (see page 3016)	This is LogMessage, a member of class TEventLogger.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

34.1.3.2 TEventLogger.Create Constructor

Pascal

```
constructor Create(Name: String);
```

C++

```
__fastcall TEventLogger(UnicodeString Name);
```

Description

This is Create, a member of class TEventLogger.

34.1.3.3 TEventLogger.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEventLogger();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

34.1.3.4 TEventLogger Methods

The methods of the TEventLogger class are listed here.

Public Methods

	Name	Description
	LogMessage (see page 3016)	This is LogMessage, a member of class TEventLogger.

34.1.3.4.1 TEventLogger.LogMessage Method

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Word = 0; ID: DWord = 0);
```

C++

```
__fastcall LogMessage(UnicodeString Message, DWord EventType = 1, Word Category = 0, DWord ID
= 0);
```

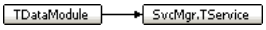
Description

This is LogMessage, a member of class TEventLogger.

34.1.4 TService Class

TService encapsulates a Windows NT service.

Class Hierarchy



Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TService = class(TDataModule);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TService : public TDataModule;
```

File

SvcMgr

Description

TService encapsulates a Windows NT service in an NT service application. A Win32 service is accessed via the Service Control Manager and can be started automatically at system boot, by a user through the Services control panel applet, or by a Win32-based application that uses the service functions. Services can execute even when no user is logged on to the system. Windows NT also supports a driver service that does not interact with the SCM.

See Also

TServiceApplication (see page 3035)

Service Applications

Service Name Properties

Service Threads

34.1.4.1 TService Members

The following tables list the members exposed by TService.

Protected Methods

	Name	Description
	Controller (see page 3020)	This is Controller, a member of class TService.
V	DoContinue (see page 3020)	This is DoContinue, a member of class TService.
V	DoCustomControl (see page 3020)	This is DoCustomControl, a member of class TService.
V	DoInterrogate (see page 3020)	This is DoInterrogate, a member of class TService.
V	DoPause (see page 3021)	This is DoPause, a member of class TService.
V	DoShutdown (see page 3021)	This is DoShutdown, a member of class TService.
V	DoStart (see page 3021)	This is DoStart, a member of class TService.
V	DoStop (see page 3021)	This is DoStop, a member of class TService.
	Main (see page 3023)	This is Main, a member of class TService.

Public Methods

	Name	Description
	CreateNew (see page 3019)	Instantiates and initializes a TService (see page 3017) object.
	Destroy (see page 3019)	Disposes of a TService (see page 3017) instance.
	GetServiceController (see page 3022)	Returns the service's registered handler function.
	LogMessage (see page 3022)	Sends an error message to the event log.
	ReportStatus (see page 3023)	Updates the Service Control Manager's status information for the calling service.

Public Properties

	Name	Description
	ErrCode (see page 3025)	Contains the service specific error code returned on starting or stopping the service.
	Param (see page 3027)	Lists the parameters to the service.
	ParamCount (see page 3027)	Indicates the number of startup parameters for the service.
	ServiceThread (see page 3028)	Identifies the dedicated thread for the service.
	Status (see page 3029)	Indicates the current state of the service.
	Terminated (see page 3030)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 3031)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Events

	Name	Description
	AfterInstall (see page 3031)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 3032)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 3032)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 3033)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 3033)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 3034)	Occurs when the thread associated with the service starts up.
	OnPause (see page 3034)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 3034)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 3035)	OnStartup occurs when the service first starts up, before the OnExecute (see page 3034) event.
	OnStop (see page 3035)	Occurs when the Service Control manager stops the service.

Published Properties

	Name	Description
	AllowPause (see page 3024)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 3024)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 3024)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 3025)	Identifies the service in the Windows Service Control Manager.
	ErrorSeverity (see page 3026)	Indicates the severity of service startup failure.
	Interactive (see page 3026)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 3026)	Names of the load ordering group that includes the service.
	Password (see page 3027)	Specifies the password, optionally used depending upon the ServiceStartName (see page 3028), for accessing the service.
	ServiceStartName (see page 3028)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 3029)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 3029)	Indicates when or how the service starts.
	TagID (see page 3030)	Specifies a unique tag for the service within its LoadGroup (see page 3026).
	WaitHint (see page 3030)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

34.1.4.2 TService.CreateNew Constructor

Instantiates and initializes a TService (see page 3017) object.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TService(TComponent * AOwner, int Dummy = 0);
```

Description

Call CreateNew to instantiate a service object at runtime. Services added at design time are created automatically. The AOwner parameter specifies the TServiceApplication (see page 3035) object that is responsible for ultimately freeing the service. The Dummy parameter is not used. Use CreateNew rather than the Create constructor to instantiate a new service object at runtime. CreateNew does not attempt to stream in an associated DFM file that indicates any components added to the service at design time.

See Also

Owner

Create (see page 3010)

TServiceApplication (see page 3035)

34.1.4.3 TService.Destroy Destructor

Disposes of a TService (see page 3017) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TService();
```

Description

Destroy frees the memory associated with the objects for the Params and Dependencies (see page 3024) properties that were created in the constructor, and for the EventLogger object created in the LogMessage (see page 3022) method. Then it calls the inherited Destroy.

See Also

System::TObject::Free (see page 3382)

LogMessage (see page 3022)

34.1.4.4 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	Controller (see page 3020)	This is Controller, a member of class TService.
	DoContinue (see page 3020)	This is DoContinue, a member of class TService.

	DoCustomControl (see page 3020)	This is DoCustomControl, a member of class TService.
	DoInterrogate (see page 3020)	This is DoInterrogate, a member of class TService.
	DoPause (see page 3021)	This is DoPause, a member of class TService.
	DoShutdown (see page 3021)	This is DoShutdown, a member of class TService.
	DoStart (see page 3021)	This is DoStart, a member of class TService.
	DoStop (see page 3021)	This is DoStop, a member of class TService.
	Main (see page 3023)	This is Main, a member of class TService.

Public Methods

	Name	Description
	GetServiceController (see page 3022)	Returns the service's registered handler function.
	LogMessage (see page 3022)	Sends an error message to the event log.
	ReportStatus (see page 3023)	Updates the Service Control Manager's status information for the calling service.

34.1.4.4.1 TService.Controller Method

Pascal

```
procedure Controller(CtrlCode: DWord);
```

C++

```
__fastcall Controller(DWord CtrlCode);
```

Description

This is Controller, a member of class TService.

34.1.4.4.2 TService.DoContinue Method

Pascal

```
function DoContinue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoContinue();
```

Description

This is DoContinue, a member of class TService.

34.1.4.4.3 TService.DoCustomControl Method

Pascal

```
function DoCustomControl(CtrlCode: DWord): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomControl(DWord CtrlCode);
```

Description

This is DoCustomControl, a member of class TService.

34.1.4.4.4 TService.DoInterrogate Method

Pascal

```
procedure DoInterrogate; virtual;
```

C++

```
virtual __fastcall DoInterrogate();
```

Description

This is DoInterrogate, a member of class TService.

34.1.4.4.5 TService.DoPause Method

Pascal

```
function DoPause: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoPause();
```

Description

This is DoPause, a member of class TService.

34.1.4.4.6 TService.DoShutdown Method

Pascal

```
procedure DoShutdown; virtual;
```

C++

```
virtual __fastcall DoShutdown();
```

Description

This is DoShutdown, a member of class TService.

34.1.4.4.7 TService.DoStart Method

Pascal

```
procedure DoStart; virtual;
```

C++

```
virtual __fastcall DoStart();
```

Description

This is DoStart, a member of class TService.

34.1.4.4.8 TService.DoStop Method

Pascal

```
function DoStop: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoStop();
```

Description

This is DoStop, a member of class TService.

34.1.4.4.9 TService.GetServiceController Method

Returns the service's registered handler function.

Pascal

```
function GetServiceController: TServiceController; virtual; abstract;
```

C++

```
virtual __fastcall TServiceController GetServiceController() = 0;
```

Description

GetServiceController returns the handler for the service. You do not need to call this function directly. The service object's main function calls it automatically.

Note: Each service in the service application has a main function for that service. When a request is made to start the service, the main thread of the service calls the RegisterServiceCtrlHandler function to register this control handler function. TService (see page 3017) automatically calls RegisterServiceCtrlHandler and returns this handler function. Because this function cannot be a class method (member function), the Service application wizard generates code that creates a non-member handler, (ServiceController) for your service object. This routine returns the service object's Handler (Controller (see page 3020)).

34.1.4.4.10 TService.LogMessage Method

Sends an error message to the event log.

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Integer = 0; ID: Integer
= 0);
```

C++

```
__fastcall LogMessage(UnicodeString Message, DWord EventType = 1, int Category = 0, int ID =
0);
```

Description

LogMessage sends an error message to the event log when an error or exception occurs. By default the EventType is EVENTLOG_ERROR_TYPE (usually indicating a loss of functionality or data) and the category and ID are zero. For example, if a service cannot be loaded as the system boots, it can log an error event, and the category and ID are zero.

The EventType specifies the type of event being logged. This parameter can be one of the following values:

Value	Meaning
EVENTLOG_WARNING_TYPE	Warning event
EVENTLOG_INFORMATION_TYPE	Information event
EVENTLOG_AUDIT_SUCCESS	Success Audit event
EVENTLOG_AUDIT_FAILURE	Failure Audit event

EVENTLOG_ERROR_TYPEError event

The Category parameter specifies the event category, which is source-specific information and can have any value.

The ID parameter specifies the event identifier, which is the message that goes with this event as an entry in the message file associated with the event source.

34.1.4.4.11 TService.Main Method

Pascal

```
procedure Main(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall Main(DWord Argc, PLPSTR Argv);
```

Description

This is Main, a member of class TService.

34.1.4.4.12 TService.ReportStatus Method

Updates the Service Control Manager's status information for the calling service.

Pascal

```
procedure ReportStatus;
```

C++

```
__fastcall ReportStatus();
```

Description

ReportStatus sends current status information about the calling service to the Service Control Manager.

See Also

LogMessage (see page 3022)

Status (see page 3029)

WaitHint (see page 3030)

34.1.4.5 TService Properties

The properties of the TService class are listed here.

Public Properties

	Name	Description
	ErrCode (see page 3025)	Contains the service specific error code returned on starting or stopping the service.
R	Param (see page 3027)	Lists the parameters to the service.
R	ParamCount (see page 3027)	Indicates the number of startup parameters for the service.
R	ServiceThread (see page 3028)	Identifies the dedicated thread for the service.
	Status (see page 3029)	Indicates the current state of the service.
R	Terminated (see page 3030)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 3031)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Properties

	Name	Description
	AllowPause (see page 3024)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 3024)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 3024)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 3025)	Identifies the service in the Windows Service Control Manager.

	ErrorSeverity (see page 3026)	Indicates the severity of service startup failure.
	Interactive (see page 3026)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 3026)	Names of the load ordering group that includes the service.
	Password (see page 3027)	Specifies the password, optionally used depending upon the ServiceStartName (see page 3028), for accessing the service.
	ServiceStartName (see page 3028)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 3029)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 3029)	Indicates when or how the service starts.
	TagID (see page 3030)	Specifies a unique tag for the service within its LoadGroup (see page 3026).
	WaitHint (see page 3030)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

34.1.4.5.1 TService.AllowPause Property

Indicates whether clients are allowed to pause the service.

Pascal

```
property AllowPause: Boolean;
```

C++

```
__property Boolean AllowPause;
```

Description

AllowPause indicates whether clients of the service can pause and resume the service by calling the Windows API ControlService function. When AllowPause is true, an OnPause (see page 3034) event occurs before the service pauses, and an OnContinue (see page 3033) event occurs when the service resumes.

See Also

OnPause (see page 3034)

OnContinue (see page 3033)

34.1.4.5.2 TService.AllowStop Property

Indicates whether clients are allowed to stop the service.

Pascal

```
property AllowStop: Boolean;
```

C++

```
__property Boolean AllowStop;
```

Description

AllowStop indicates whether clients of the service can stop it by calling the Windows API ControlService function. When AllowStop is true, an OnStop (see page 3035) event occurs before the service stops.

See Also

OnStop (see page 3035)

34.1.4.5.3 TService.Dependencies Property

Lists the services or load ordering groups on which the service depends.

Pascal

```
property Dependencies: TDependencies;
```

C++

```
__property TDependencies Dependencies;
```

Description

Dependencies is a container for TDependency (see page 3012) objects, which are dependent services or load ordering groups that must start before this service. If the container is empty, the service has no dependencies.

Dependency on a service means that this service can only run if the service it depends on is running. Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.

See Also

TDependency (see page 3012)

LoadGroup (see page 3026)

34.1.4.5.4 TService.DisplayName Property

Identifies the service in the Windows Service Control Manager.

Pascal

```
property DisplayName: String;
```

C++

```
__property UnicodeString DisplayName;
```

Description

DisplayName contains a string, with a maximum length of 256 characters, that is the name of the service as displayed in the Service Control Manager. User interface programs use DisplayName to identify the service. DisplayName is case-preserved in the Service Control Manager, however display name comparisons are always case-insensitive.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Name

ServiceStartName (see page 3028)

34.1.4.5.5 TService.ErrCode Property

Contains the service specific error code returned on starting or stopping the service.

Pascal

```
property ErrCode: DWord;
```

C++

```
__property DWord ErrCode;
```

Description

ErrCode specifies a service specific error code that the service returns when an error occurs while the service is starting or stopping. If ErrCode is not defined, the Win32ErrCode (see page 3031) property is used instead.

See Also

ReportStatus (see page 3023)

Win32ErrCode (see page 3031)

34.1.4.5.6 TService.ErrorSeverity Property

Indicates the severity of service startup failure.

Pascal

```
property ErrorSeverity: TErrorSeverity;
```

C++

```
__property TErrorSeverity ErrorSeverity;
```

Description

ErrorSeverity indicates the severity if this service fails to start during startup of the service application.

See Also

ErrCode (see page 3025)

ReportStatus (see page 3023)

Win32ErrCode (see page 3031)

34.1.4.5.7 TService.Interactive Property

Indicates whether the service is interactive with the Windows desktop.

Pascal

```
property Interactive: Boolean;
```

C++

```
__property Boolean Interactive;
```

Description

Interactive indicates whether the service can interact with the Windows desktop. Interactive only has an effect when ServiceType (see page 3029) is stWin32. For more information about creating interactive NT services, see the Microsoft Help.

See Also

ServiceType (see page 3029)

34.1.4.5.8 TService.LoadGroup Property

Names of the load ordering group that includes the service.

Pascal

```
property LoadGroup: String;
```

C++

```
__property UnicodeString LoadGroup;
```

Description

LoadGroup is the name of the load-ordering group that includes the service. This is used by other services that have

dependencies on the load ordering group (See Dependencies (see page 3024)). The registry lists load-ordering groups at HKEY_LOCAL_MACHINE\System (see page 3165)\CurrentControlSet\Control\ServiceGroupOrder. The order in which services are loaded depends on the order of the load groups in the Registry and the dependencies listed in service Dependencies (see page 3024) properties.

Because services and service groups share the same name space, the SC_GROUP_IDENTIFIER character must prefix group names to distinguish them from service names. This is automatically prefixed if the TDependency (see page 3012)'s IsGroup property is true.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Dependencies (see page 3024)

TagID (see page 3030)

34.1.4.5.9 TService.Param Property

Lists the parameters to the service.

Pascal

```
property Param [Index: Integer]: String;
```

C++

```
__property UnicodeString Param[int Index];
```

Description

Param is an indexed property that contains the parameters to the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

ParamCount (see page 3027)

34.1.4.5.10 TService.ParamCount Property

Indicates the number of startup parameters for the service.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters for the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

Param (see page 3027)

34.1.4.5.11 TService.Password Property

Specifies the password, optionally used depending upon the ServiceStartName (see page 3028), for accessing the service.

Pascal

```
property Password: String;
```

C++

```
__property UnicodeString Password;
```

Description

Password is the password for the account name specified by the ServiceStartName (see page 3028) property if the ServiceType (see page 3029) is stWin32.

If Password is nil (Delphi) or NULL (C++), or if it points to an empty string, the service has no password. If ServiceType (see page 3029) is stDevice or stFileSystem, this parameter is ignored.

See Also

ServiceStartName (see page 3028)

34.1.4.5.12 TService.ServiceStartName Property

ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.

Pascal

```
property ServiceStartName: String;
```

C++

```
__property UnicodeString ServiceStartName;
```

Description

ServiceStartName is the name used to start the service.

If ServiceType (see page 3029) is stWin32, ServiceStartName is the account name (in the form "DomainName\Username") which the service process is logged on as when it runs. If the account belongs to the built-in domain, ".\Username" can be specified. If the service application includes more than one service, leave ServiceStartName blank, and leave the Password (see page 3027) property blank.

If the service type is not stWin32, ServiceStartName is the driver object name (\FileSytems\Rdr or \Driver\Xns) which the input and output system uses to load the device driver. With device drivers or file systems drivers, you can leave ServiceStartName blank and the I/O system will create a default name based on the service name.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Password (see page 3027)

34.1.4.5.13 TService.ServiceThread Property

Identifies the dedicated thread for the service.

Pascal

```
property ServiceThread: TServiceThread;
```

C++

```
__property TServiceThread ServiceThread;
```

Description

ServiceThread is the instance of the TServiceThread (see page 3041) class that is the dedicated thread for the service. Each service in the service application has a dedicated thread that contains a loop in its Execute method for processing service requests. Often a service will spawn a new thread, derived from TThread, to free the service thread to continue processing service requests.

See Also

ProcessRequests (see page 3046)

TThread

34.1.4.5.14 TService.ServiceType Property

Identifies the type of the service as Win32, device driver, or file system driver.

Pascal

```
property ServiceType: TServiceType;
```

C++

```
__property TServiceType ServiceType;
```

Description

ServiceType indicates the type of service. ServiceType can be stWin32 (a Win32 service – the default), stDevice (a device driver), or stFileSystem (a file system driver).

See Also

Interactive (see page 3026)

34.1.4.5.15 TService.StartType Property

Indicates when or how the service starts.

Pascal

```
property StartType: TStartType;
```

C++

```
__property TStartType StartType;
```

Description

StartType specifies how and why the service starts

34.1.4.5.16 TService.Status Property

Indicates the current state of the service.

Pascal

```
property Status: TCurrentStatus;
```

C++

```
__property TCurrentStatus Status;
```

Description

Status indicates the current state of the service. The status information is displayed in the Service Control Manager.

See Also

ReportStatus (see page 3023)

34.1.4.5.17 TService.TagID Property

Specifies a unique tag for the service within its LoadGroup (see page 3026).

Pascal

```
property TagID: DWord;
```

C++

```
__property DWord TagID;
```

Description

TagID is the unique tag value for this service in its LoadGroup (see page 3026). If no tag is requested, this property is 0. Tags are only evaluated when the ServiceType (see page 3029) is not stWin32 and the StartType (see page 3029) is stBoot or stSystem.

See Also

Dependencies (see page 3024)

LoadGroup (see page 3026)

34.1.4.5.18 TService.Terminated Property

Indicates whether the service thread has terminated.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Terminated indicates whether the dedicated thread that the service is running in has terminated.

See Also

Terminated

Terminate

34.1.4.5.19 TService.WaitHint Property

Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

Pascal

```
property WaitHint: Integer;
```

C++

```
__property int WaitHint;
```

Description

WaitHint specifies the amount of time, in milliseconds, that can elapse between a pending start, stop, pause, or continue

operation and the service setting its status with a call to the Windows API. Normally this is handled automatically. If the service performs a lengthy operation of more than WaitHint time in one of its event handlers, you should call ReportStatus (see page 3023) method to update to the Service Control Manager.

See Also

ReportStatus (see page 3023)

34.1.4.5.20 TService.Win32ErrCode Property

Indicates the Win32 error code reported by the service if a service specific code is not defined.

Pascal

```
property Win32ErrCode: DWord;
```

C++

```
__property DWord Win32ErrCode;
```

Description

Win32ErrCode specifies a Win32 error code that the service uses to report an error that occurs when it is starting or stopping. If an error code specific to the service is defined, this property is automatically set to ERROR_SERVICE_SPECIFIC_ERROR and the service specific error is reported instead.

See Also

ErrCode (see page 3025)

ReportStatus (see page 3023)

34.1.4.6 TService Events

The events of the TService class are listed here.

Published Events

	Name	Description
	AfterInstall (see page 3031)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 3032)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 3032)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 3033)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 3033)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 3034)	Occurs when the thread associated with the service starts up.
	OnPause (see page 3034)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 3034)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 3035)	OnStartup occurs when the service first starts up, before the OnExecute (see page 3034) event.
	OnStop (see page 3035)	Occurs when the Service Control manager stops the service.

34.1.4.6.1 TService.AfterInstall Event

Occurs immediately after the service is registered with the Windows Service Control manager.

Pascal

```
property AfterInstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterInstall;
```

Description

Write (see page 3789) an AfterInstall event handler to take specific action immediately after the service is registered. Install services by running the service application with the INSTALL switch.

See Also

AfterUninstall (see page 3032)

OnStart (see page 3035)

OnExecute (see page 3034)

OnCreate

BeforeInstall (see page 3032)

34.1.4.6.2 TService.AfterUninstall Event

Occurs immediately after the service is removed from the Service Control manager's database.

Pascal

```
property AfterUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterUninstall;
```

Description

Write (see page 3789) an AfterUninstall event handler to take specific action immediately after the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterInstall (see page 3031)

OnStop (see page 3035)

BeforeUninstall (see page 3033)

Destroy

OnShutdown (see page 3034)

34.1.4.6.3 TService.BeforeInstall Event

Occurs before the service is first registered with the Windows Service Control manager.

Pascal

```
property BeforeInstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeInstall;
```

Description

Write (see page 3789) a BeforeInstall event handler to take specific action immediately before the service is first registered. Install services by running the service application with the INSTALL switch.

See Also

AfterInstall (see page 3031)

OnStart (see page 3035)

OnExecute (see page 3034)

OnCreate

BeforeUninstall (see page 3033)

34.1.4.6.4 TService.BeforeUninstall Event

Occurs immediately before the service is removed from the Service Control manager's database.

Pascal

```
property BeforeUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeUninstall;
```

Description

Write (see page 3789) a BeforeUninstall event handler to take specific action immediately before the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterUninstall (see page 3032)

OnStop (see page 3035)

BeforeInstall (see page 3032)

Destroy

OnShutdown (see page 3034)

34.1.4.6.5 TService.OnContinue Event

Occurs when the Service Control manager resumes the service after it has been paused.

Pascal

```
property OnContinue: TContinueEvent;
```

C++

```
__property TContinueEvent OnContinue;
```

Description

Write (see page 3789) an OnContinue event handler to take specific action when the Service Control manager resumes the service after it has been paused.

See Also

OnPause (see page 3034)

34.1.4.6.6 TService.OnExecute Event

Occurs when the thread associated with the service starts up.

Pascal

```
property OnExecute: TServiceEvent;
```

C++

```
__property TServiceEvent OnExecute;
```

Description

If you are not spawning a new thread to handle individual service requests in an OnStart (see page 3035) event handler, this is where you implement the service. When the OnExecute event handler finishes, the service thread terminates. Most OnExecute event handlers contain a loop that calls the service thread's ProcessRequests method so that other service requests are not locked out.

See Also

ProcessRequests (see page 3046)

OnStart (see page 3035)

34.1.4.6.7 TService.OnPause Event

Occurs when the Service Control manager pauses the service temporarily.

Pascal

```
property OnPause: TPauseEvent;
```

C++

```
__property TPauseEvent OnPause;
```

Description

Write (see page 3789) an OnPause event handler to take specific action when the service is temporarily paused.

See Also

OnContinue (see page 3033)

34.1.4.6.8 TService.OnShutdown Event

Occurs when the system running the service application is about to shut down.

Pascal

```
property OnShutdown: TServiceEvent;
```

C++

```
__property TServiceEvent OnShutdown;
```

Description

Write (see page 3789) an OnShutdown event handler to take specific action just before the service shuts down. The service application terminates immediately after OnShutdown.

See Also

OnStop (see page 3035)

OnStart (see page 3035)

OnPause (see page 3034)

34.1.4.6.9 TService.OnStart Event

OnStartup occurs when the service first starts up, before the OnExecute (see page 3034) event.

Pascal

```
property OnStart: TStartEvent;
```

C++

```
__property TStartEvent OnStart;
```

Description

This event should be used to initialize the service. For example, if each service request is handled in a separate thread (a good idea if handling the request takes much time) the thread for a request is spawned in an OnStart event handler.

See Also

OnStop (see page 3035)

OnContinue (see page 3033)

OnExecute (see page 3034)

34.1.4.6.10 TService.OnStop Event

Occurs when the Service Control manager stops the service.

Pascal

```
property OnStop: TStopEvent;
```

C++

```
__property TStopEvent OnStop;
```

Description

Write (see page 3789) an OnStop event handler to take specific action when the service stops.

See Also

OnShutdown (see page 3034)

OnStart (see page 3035)

OnPause (see page 3034)

34.1.5 TServiceApplication Class

TServiceApplication encapsulates a Windows NT service application.

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TServiceApplication = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TServiceApplication : public TComponent;
```

File

SvcMgr

Description

TServiceApplication encapsulates a Windows NT application type known as a service. TServiceApplication provides the fundamental behavior of this type of application. Each Service project automatically declares an Application (see page 3055) variable of type TServiceApplication as the instance of the application. The Application (see page 3055) variable is added to a project by selecting File | New (see page 3712) from the menu of the IDE main window and selecting Service Application (see page 3055) in the New (see page 3712) Items dialog.

TServiceApplication contains TService (see page 3017) objects, each of which encapsulates a Windows NT service. The service application object provides methods to create service objects and to install, register, dispatch, and uninstall services.

When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

Note: Do not add the Forms or Httpapp units to the uses list of the project (Delphi) or includ the unit header for these units in the project source (C++). Those units declare conflicting Application (see page 3055) variables of types TApplication and TWebApplication, respectively.

See Also

Application (see page 3055)

TService (see page 3017)

Service Applications

Service Name\ Properties

34.1.5.1 TServiceApplication Members

The following tables list the members exposed by TServiceApplication.

Protected Methods

	Name	Description
	DispatchServiceMain (see page 3038)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 3038)	This is DoHandleException, a member of class TServiceApplication.
	Hook (see page 3039)	This is Hook, a member of class TServiceApplication.
	RegisterServices (see page 3039)	This is RegisterServices, a member of class TServiceApplication.

Public Methods

	Name	Description
	Create (see page 3037)	Instantiates a TServiceApplication (see page 3035) object.
	CreateForm (see page 3038)	Creates the service object or objects for the service application.

	Destroy (see page 3037)	Disposes of an instance of a TServiceApplication (see page 3035) object.
	Initialize (see page 3039)	Sets the initial state of the application and hides any associated MainForm.
	Installing (see page 3039)	This is Installing, a member of class TServiceApplication.
	Run (see page 3040)	Installs, registers, and adds services to its component list, and creates a thread for the service application.

Public Properties

	Name	Description
	DelayInitialize (see page 3040)	This is DelayInitialize, a member of class TServiceApplication.
	ServiceCount (see page 3040)	Indicates the number of services in the service application.
	Title (see page 3041)	Indicates the name of the title application.

34.1.5.2 TServiceApplication.Create Constructor

Instantiates a TServiceApplication (see page 3035) object.

Pascal

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

C++

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

Description

Applications do not need to call the constructor directly. TServiceApplication (see page 3035) is instantiated by selecting File | New (see page 3712) from the menu of the IDE main window and selecting Service Application (see page 3055) in the New (see page 3712) Items dialog. An Application (see page 3055) variable of type TServiceApplication (see page 3035) is declared and initialized as the instance of the application.

34.1.5.3 TServiceApplication.Destroy Destructor

Disposes of an instance of a TServiceApplication (see page 3035) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServiceApplication();
```

Description

You do not need to call the destructor directly. TServiceApplication (see page 3035) is destroyed when your application closes.

The destructor frees the memory associated with sub-objects.

See Also

System::TObject::Free (see page 3382)

34.1.5.4 TServiceApplication Methods

The methods of the TServiceApplication class are listed here.

Protected Methods

	Name	Description
	DispatchServiceMain (see page 3038)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 3038)	This is DoHandleException, a member of class TServiceApplication.
	Hook (see page 3039)	This is Hook, a member of class TServiceApplication.
	RegisterServices (see page 3039)	This is RegisterServices, a member of class TServiceApplication.

Public Methods

	Name	Description
	CreateForm (see page 3038)	Creates the service object or objects for the service application.
	Initialize (see page 3039)	Sets the initial state of the application and hides any associated MainForm.
	Installing (see page 3039)	This is Installing, a member of class TServiceApplication.
	Run (see page 3040)	Installs, registers, and adds services to its component list, and creates a thread for the service application.

34.1.5.4.1 TServiceApplication.CreateForm Method

Creates the service object or objects for the service application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the service objects of the service application.

CreateForm instantiates a TService (see page 3017) object for each service contained in the service application. CreateForm is called automatically, for the first service, when the service application is first created. Subsequently, CreateForm is called for each new service added to the application. To add a new service, select File | New (see page 3712) from the menu of the IDE main window and choose Service in the New (see page 3712) Items dialog.

34.1.5.4.2 TServiceApplication.DispatchServiceMain Method

Pascal

```
procedure DispatchServiceMain(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall DispatchServiceMain(DWord Argc, PLPSTR Argv);
```

Description

This is DispatchServiceMain, a member of class TServiceApplication.

34.1.5.4.3 TServiceApplication.DoHandleException Method

Pascal

```
procedure DoHandleException(E: Exception); dynamic;
```

C++

```
__fastcall DoHandleException(Exception E);
```

Description

This is DoHandleException, a member of class TServiceApplication.

34.1.5.4.4 TServiceApplication.Hook Method

Pascal

```
function Hook(var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean Hook(TMessage Message);
```

Description

This is Hook, a member of class TServiceApplication.

34.1.5.4.5 TServiceApplication.Initialize Method

Sets the initial state of the application and hides any associated MainForm.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize sets the initial state of the application to hide any main form that may be associated with the application. Service applications are rarely interactive and do not typically have a form associated with them. If a form has been added to the application, Initialize hides it on start. Initialize then follows the standard (default) VCL application initialization process.

34.1.5.4.6 TServiceApplication.Installing Method

Pascal

```
function Installing: Boolean;
```

C++

```
__fastcall Boolean Installing();
```

Description

This is Installing, a member of class TServiceApplication.

34.1.5.4.7 TServiceApplication.RegisterServices Method

Pascal

```
procedure RegisterServices(Install: Boolean; Silent: Boolean);
```

C++

```
__fastcall RegisterServices(Boolean Install, Boolean Silent);
```

Description

This is RegisterServices, a member of class TServiceApplication.

34.1.5.4.8 TServiceApplication.Run Method

Installs, registers, and adds services to its component list, and creates a thread for the service application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

Run handles installing and uninstalling services, depending on the command line switches. When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

If not called with the /INSTALL or /UNINSTALL switch, Run creates a service application thread that responds to Windows messages from clients. Note that this thread is for the application and differs from the dedicated TServiceThread (see page 3041) instances associated with individual services in the application.

34.1.5.5 TServiceApplication Properties

The properties of the TServiceApplication class are listed here.

Public Properties

	Name	Description
	DelayInitialize (see page 3040)	This is DelayInitialize, a member of class TServiceApplication.
R	ServiceCount (see page 3040)	Indicates the number of services in the service application.
	Title (see page 3041)	Indicates the name of the title application.

34.1.5.5.1 TServiceApplication.DelayInitialize Property

Pascal

```
property DelayInitialize: Boolean;
```

C++

```
__property Boolean DelayInitialize;
```

Description

This is DelayInitialize, a member of class TServiceApplication.

34.1.5.5.2 TServiceApplication.ServiceCount Property

Indicates the number of services in the service application.

Pascal

```
property ServiceCount: Integer;
```

C++

```
__property int ServiceCount;
```

Description

ServiceCount is the number of services in the service application. Each service is encapsulated by a TService (see page 3017) instance and has its own dedicated service thread.

Note: If the Service application requires a user name and password, the application must implement only a single service.

See Also

TService (see page 3017)

34.1.5.5.3 TServiceApplication.Title Property

Indicates the name of the title application.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Title is the name of the service application. This text string appears below the icon of the application when it is minimized.

34.1.6 TServiceThread Class

TServiceThread represents the thread associated with a specific service (TService (see page 3017) object).

Class Hierarchy

Pascal

```
TServiceThread = class(TThread);
```

C++

```
class TServiceThread : public TThread;
```

File

SvcMgr

Description

TServiceThread is the thread associated with a service. Each service application has one or more service objects, each of which has its own dedicated thread. TServiceThread instances represent those threads.

Note: Service threads.

See Also

TService (see page 3017)

TService (see page 3017)

TServiceApplication (see page 3035)

TServiceThread

IDH

Service Threads

34.1.6.1 TServiceThread Members

The following tables list the members exposed by TServiceThread.

Protected Methods

	Name	Description
	Execute (see page 3043)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
	Create (see page 3042)	Instantiates and initializes a TServiceThread (see page 3041) object.
	ProcessRequests (see page 3046)	Allows clients to invoke the service associated with the service thread.

34.1.6.2 TServiceThread.Create Constructor

Instantiates and initializes a TServiceThread (see page 3041) object.

Pascal

```
constructor Create(Service: TService);
```

C++

```
__fastcall TServiceThread(TService Service);
```

Description

Applications do not need to call the constructor directly. Instances of TService (see page 3017) create the TServiceThread (see page 3041) object on startup. The Service parameter is the TService (see page 3017) instance for which the TServiceThread (see page 3041) is the dedicated thread.

The constructor sets the FreeOnTerminate property to true and then calls the base class constructor so that the thread object is created with its Suspended property set to false.

See Also

FreeOnTerminate

Suspended

34.1.6.3 TServiceThread Methods

The methods of the TServiceThread class are listed here.

Protected Methods

	Name	Description
	Execute (see page 3043)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
	ProcessRequests (see page 3046)	Allows clients to invoke the service associated with the service thread.

34.1.6.3.1 TServiceThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
```

```
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
```

```
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

34.1.6.3.2 TServiceThread.ProcessRequests Method

Allows clients to invoke the service associated with the service thread.

Pascal

```
procedure ProcessRequests(WaitForMessage: Boolean);
```

C++

```
__fastcall ProcessRequests(Boolean WaitForMessage);
```

Description

TServiceThread (see page 3041)'s ProcessRequests is called in the Execute (see page 3043) method after any associated service's OnExecute event handler has finished executing. For more information about optimizing service execution by spawning individual threads for each service request, see the Developer's Guide.

See Also

OnExecute (see page 3034)

34.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCurrentStatus (see page 3048)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 3049)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 3049)	TServiceType indicates the type of a service.
	TStartType (see page 3050)	TStartType indicates how a service starts up.

34.2.1 SvcMgr.TCurrentStatus Enumeration

TCurrentStatus indicates the state of a service.

Pascal

```
TCurrentStatus = (
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
);
```

C++

```
enum TCurrentStatus {
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
};
```

File

SvcMgr

Description

TCurrentStatus can have one of the following values:

Value	Meaning
csStopped	The service is not running.
csStartPending	The service is starting.
csStopPending	The service is stopping.
csRunning	The service is running.

csContinuePending	The service continue is pending.
csPausePending	The service pause is pending.
csPaused	The service is paused

34.2.2 SvcMgr.TErrorSeverity Enumeration

TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.

Pascal

```
TErrorSeverity = (
  esIgnore,
  esNormal,
  esSevere,
  esCritical
);
```

C++

```
enum TErrorSeverity {
  esIgnore,
  esNormal,
  esSevere,
  esCritical
};
```

File

SvcMgr

Description

TErrorSeverity can have one of the following values:

Value	What the startup program does
esIgnore	Logs the error and continues.
esNormal	Logs the error and displays a message, but continues.
esSevere	Logs the error but continues only if the last-known-good configuration is being started; otherwise the last-known-good configuration is started instead.
esCritical	Logs the error and starts up the last-known-good configuration instead. If this is the last-known-good configuration, the startup operation fails.

34.2.3 SvcMgr.TServiceType Enumeration

TServiceType indicates the type of a service.

Pascal

```
TServiceType = (
  stWin32,
  stDevice,
  stFileSystem
);
```

C++

```
enum TServiceType {
  stWin32,
  stDevice,
  stFileSystem
};
```

File

SvcMgr

Description

TServiceType can have one of the following values:

Value	Meaning
stWin32	a Win32 service.
stDevice	a device driver.
stFileSystem	a file system driver.

34.2.4 SvcMgr.TStartType Enumeration

TStartType indicates how a service starts up.

Pascal

```
TStartType = (
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
);
```

C++

```
enum TStartType {
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
};
```

File

SvcMgr

Description

The following table lists the possible values of TStartType:

StartType value	Service is started
stBoot(used only when ServiceType is not stWin32)	by the operating system loader
stSystem(used only when ServiceType is not stWin32)	after boot systems are initialized
stAuto	automatically during system startup

stManual	when a process calls the Windows StartService function
stDisabled	the service can't be started except by an administrator

34.3 Types

The following table lists types in this documentation.

Types

Name	Description
TContinueEvent (see page 3052)	This is type SvcMgr.TContinueEvent.
TPauseEvent (see page 3052)	This is type SvcMgr.TPauseEvent.
TServiceController (see page 3053)	This is type SvcMgr.TServiceController.
TServiceEvent (see page 3053)	This is type SvcMgr.TServiceEvent.
TStartEvent (see page 3053)	This is type SvcMgr.TStartEvent.
TStopEvent (see page 3054)	This is type SvcMgr.TStopEvent.

34.3.1 SvcMgr.TContinueEvent Type

Pascal

```
TContinueEvent = procedure (Sender: TService; var Continued: Boolean) of object;
```

C++

```
(Sender: TService; var Continued: Boolean) ( TContinueEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TContinueEvent.

34.3.2 SvcMgr.TPauseEvent Type

Pascal

```
TPauseEvent = procedure (Sender: TService; var Paused: Boolean) of object;
```

C++

```
(Sender: TService; var Paused: Boolean) ( TPauseEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TPauseEvent.

34.3.3 SvcMgr.TServiceController Type

Pascal

```
TServiceController = procedure (CtrlCode: DWord);
```

C++

```
(CtrlCode: DWord) ( TServiceController)();
```

File

SvcMgr

Description

This is type SvcMgr.TServiceController.

34.3.4 SvcMgr.TServiceEvent Type

Pascal

```
TServiceEvent = procedure (Sender: TService) of object;
```

C++

```
(Sender: TService) ( TServiceEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TServiceEvent.

34.3.5 SvcMgr.TStartEvent Type

Pascal

```
TStartEvent = procedure (Sender: TService; var Started: Boolean) of object;
```

C++

```
(Sender: TService; var Started: Boolean) ( TStartEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TStartEvent.

34.3.6 SvcMgr.TStopEvent Type

Pascal

```
TStopEvent = procedure (Sender: TService; var Stopped: Boolean) of object;
```

C++

```
(Sender: TService; var Stopped: Boolean) ( TStopEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TStopEvent.

34.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 3055)	This is variable SvcMgr.Application.

34.4.1 SvcMgr.Application Variable

Pascal

```
Application: TServiceApplication = nil;
```

C++

```
TServiceApplication Application = nil;
```

File

SvcMgr

Description

This is variable SvcMgr.Application.

34.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 3056)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

34.5.1 SvcMgr.CM_SERVICE_CONTROL_CODE Constant

Pascal

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

C++

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

File

SvcMgr

Description

This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

35 SyncObjs Namespace

This is namespace SyncObjs.

Topics

Name	Description
Classes (see page 3058)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3094)	The following table lists structs, records, enums in this documentation.
Types (see page 3095)	The following table lists types in this documentation.

Classes

	Name	Description
	TConditionVariableCS (see page 3058)	This is class SyncObjs.TConditionVariableCS.
	TConditionVariableHelper (see page 3062)	This is class SyncObjs.TConditionVariableHelper.
	TConditionVariableMutex (see page 3064)	This is class SyncObjs.TConditionVariableMutex.
	TCriticalSection (see page 3068)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TCriticalSectionHelper (see page 3072)	This is class SyncObjs.TCriticalSectionHelper.
	TEvent (see page 3074)	TEvent represents an external event.
	THandleObject (see page 3078)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 3083)	This is class SyncObjs.TMutex.
	TSemaphore (see page 3086)	This is class SyncObjs.TSemaphore.
	TSimpleEvent (see page 3090)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 3091)	TSynchroObject is the base class for all thread synchronization objects.

Structs, Records, Enums

	Name	Description
	TWaitResult (see page 3094)	TWaitResult indicates the reason an event object stopped waiting for its signal.

Types

Name	Description
THandleObjectArray (see page 3095)	This is type SyncObjs.THandleObjectArray.

35.1 Classes

The following table lists classes in this documentation.

Class Helpers

Name	Description
TConditionVariableHelper (see page 3062)	This is class SyncObjs.TConditionVariableHelper.
TCriticalSectionHelper (see page 3072)	This is class SyncObjs.TCriticalSectionHelper.

Classes

	Name	Description
	TConditionVariableCS (see page 3058)	This is class SyncObjs.TConditionVariableCS.
	TConditionVariableMutex (see page 3064)	This is class SyncObjs.TConditionVariableMutex.
	TCriticalSection (see page 3068)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TEvent (see page 3074)	TEvent represents an external event.
	THandleObject (see page 3078)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 3083)	This is class SyncObjs.TMutex.
	TSemaphore (see page 3086)	This is class SyncObjs.TSemaphore.
	TSimpleEvent (see page 3090)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 3091)	TSynchroObject is the base class for all thread synchronization objects.

35.1.1 TConditionVariableCS Class

Class Hierarchy

Pascal

```
TConditionVariableCS = class(TSynchroObject);
```

C++

```
class TConditionVariableCS : public TSynchroObject;
```

File

SyncObjs

Description

This is class SyncObjs.TConditionVariableCS.

35.1.1.1 TConditionVariableCS Members

The following tables list the members exposed by TConditionVariableCS.

Protected Fields

	Name	Description
	FConditionVariable (see page 3060)	This is FConditionVariable, a member of class TConditionVariableCS.

Public Methods

	Name	Description
	Acquire (see page 3060)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3059)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3061)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ReleaseAll (see page 3061)	This is ReleaseAll, a member of class TConditionVariableCS.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	WaitFor (see page 3061)	This is the overview for the WaitFor method overload.

35.1.1.2 TConditionVariableCS.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TConditionVariableCS();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

35.1.1.3 TConditionVariableCS Fields

The fields of the TConditionVariableCS class are listed here.

Protected Fields

	Name	Description
	FConditionVariable (see page 3060)	This is FConditionVariable, a member of class TConditionVariableCS.

35.1.1.3.1 TConditionVariableCS.FConditionVariable Field

Pascal

```
FConditionVariable: TRTLConditionVariable;
```

C++

```
TRTLConditionVariable FConditionVariable;
```

Description

This is FConditionVariable, a member of class TConditionVariableCS.

35.1.1.4 TConditionVariableCS Methods

The methods of the TConditionVariableCS class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3060)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3061)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ReleaseAll (see page 3061)	This is ReleaseAll, a member of class TConditionVariableCS.
	WaitFor (see page 3061)	This is the overview for the WaitFor method overload.

35.1.1.4.1 TConditionVariableCS.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3093)

35.1.1.4.2 TConditionVariableCS.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3092)

35.1.1.4.3 TConditionVariableCS.ReleaseAll Method

Pascal

```
procedure ReleaseAll;
```

C++

```
__fastcall ReleaseAll();
```

Description

This is ReleaseAll, a member of class TConditionVariableCS.

35.1.1.4.4 WaitFor Method

This is the overview for the WaitFor method overload.

Overload List

	Name	Description
	TConditionVariableCS.WaitFor (TCriticalSection, LongWord) (see page 3062)	This is WaitFor, a member of class TConditionVariableCS.
	TConditionVariableCS.WaitFor (TRTLCriticalSection, LongWord) (see page 3062)	This is WaitFor, a member of class TConditionVariableCS.

35.1.1.4.4.1 TConditionVariableCS.WaitFor Method (TCriticalSection, LongWord)

Pascal

```
function WaitFor(CriticalSection: TCriticalSection; TimeOut: LongWord = INFINITE):
TWaitResult; overload;
```

C++

```
__fastcall TWaitResult WaitFor(TCriticalSection CriticalSection, LongWord TimeOut = INFINITE);
```

Description

This is WaitFor, a member of class TConditionVariableCS.

35.1.1.4.4.2 TConditionVariableCS.WaitFor Method (TRTLCriticalSection, LongWord)

Pascal

```
function WaitFor(var CriticalSection: TRTLCriticalSection; TimeOut: LongWord = INFINITE):
TWaitResult; overload;
```

C++

```
__fastcall TWaitResult WaitFor(TRTLCriticalSection CriticalSection, LongWord TimeOut =
INFINITE);
```

Description

This is WaitFor, a member of class TConditionVariableCS.

35.1.2 TConditionVariableHelper Class Helper

Class Hierarchy

SyncObjs.TConditionVariableHelper

Pascal

```
TConditionVariableHelper = class helper for TRTLConditionVariable;
```

C++

```
class TConditionVariableHelper;
```

File

SyncObjs

Description

This is class SyncObjs.TConditionVariableHelper.

35.1.2.1 TConditionVariableHelper Members

The following tables list the members exposed by TConditionVariableHelper.

Public Methods

	Name	Description
	Create (see page 3063)	This is Create, a member of class TConditionVariableHelper.
	Free (see page 3063)	This is Free, a member of class TConditionVariableHelper.

	SleepCS (see page 3063)	This is SleepCS, a member of class TConditionVariableHelper.
	Wake (see page 3064)	This is Wake, a member of class TConditionVariableHelper.
	WakeAll (see page 3064)	This is WakeAll, a member of class TConditionVariableHelper.

35.1.2.2 TConditionVariableHelper Methods

The methods of the TConditionVariableHelper class are listed here.

Public Methods

	Name	Description
	Create (see page 3063)	This is Create, a member of class TConditionVariableHelper.
	Free (see page 3063)	This is Free, a member of class TConditionVariableHelper.
	SleepCS (see page 3063)	This is SleepCS, a member of class TConditionVariableHelper.
	Wake (see page 3064)	This is Wake, a member of class TConditionVariableHelper.
	WakeAll (see page 3064)	This is WakeAll, a member of class TConditionVariableHelper.

35.1.2.2.1 TConditionVariableHelper.Create Method

Pascal

```
class function Create: TRTLConditionVariable; static;
```

C++

```
static __fastcall TRTLConditionVariable Create();
```

Description

This is Create, a member of class TConditionVariableHelper.

35.1.2.2.2 TConditionVariableHelper.Free Method

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

This is Free, a member of class TConditionVariableHelper.

35.1.2.2.3 TConditionVariableHelper.SleepCS Method

Pascal

```
function SleepCS(var CriticalSection: TRTLCriticalSection; dwMilliseconds: DWORD): Boolean;
```

C++

```
__fastcall Boolean SleepCS(TRTLCriticalSection CriticalSection, DWORD dwMilliseconds);
```

Description

This is SleepCS, a member of class TConditionVariableHelper.

35.1.2.2.4 TConditionVariableHelper.Wake Method

Pascal

```
procedure Wake;
```

C++

```
__fastcall Wake();
```

Description

This is Wake, a member of class TConditionVariableHelper.

35.1.2.2.5 TConditionVariableHelper.WakeAll Method

Pascal

```
procedure WakeAll;
```

C++

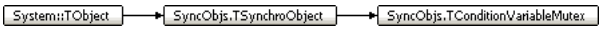
```
__fastcall WakeAll();
```

Description

This is WakeAll, a member of class TConditionVariableHelper.

35.1.3 TConditionVariableMutex Class

Class Hierarchy



Pascal

```
TConditionVariableMutex = class(TSynchroObject);
```

C++

```
class TConditionVariableMutex : public TSynchroObject;
```

File

SyncObjs

Description

This is class SyncObjs.TConditionVariableMutex.

35.1.3.1 TConditionVariableMutex Members

The following tables list the members exposed by TConditionVariableMutex.

Public Methods

	Name	Description
	Acquire (see page 3066)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.

	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3065)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3066)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3067)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ReleaseAll (see page 3067)	This is ReleaseAll, a member of class TConditionVariableMutex.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	WaitFor (see page 3067)	This is WaitFor, a member of class TConditionVariableMutex.

35.1.3.2 TConditionVariableMutex.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TConditionVariableMutex();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

35.1.3.3 TConditionVariableMutex.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConditionVariableMutex();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

35.1.3.4 TConditionVariableMutex Methods

The methods of the TConditionVariableMutex class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3066)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3067)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ReleaseAll (see page 3067)	This is ReleaseAll, a member of class TConditionVariableMutex.
	WaitFor (see page 3067)	This is WaitFor, a member of class TConditionVariableMutex.

35.1.3.4.1 TConditionVariableMutex.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3093)

35.1.3.4.2 TConditionVariableMutex.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3092)

35.1.3.4.3 TConditionVariableMutex.ReleaseAll Method

Pascal

```
procedure ReleaseAll;
```

C++

```
__fastcall ReleaseAll();
```

Description

This is ReleaseAll, a member of class TConditionVariableMutex.

35.1.3.4.4 TConditionVariableMutex.WaitFor Method

Pascal

```
function WaitFor(AExternalMutex: TMutex; TimeOut: LongWord = INFINITE): TWaitResult;
```

C++

```
__fastcall TWaitResult WaitFor(TMutex AExternalMutex, LongWord TimeOut = INFINITE);
```

Description

This is WaitFor, a member of class TConditionVariableMutex.

35.1.4 TCriticalSection Class

TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.

Class Hierarchy

Pascal

```
TCriticalSection = class(TSynchroObject);
```

C++

```
class TCriticalSection : public TSynchroObject;
```

File

SyncObjs

Description

Use TCriticalSection to safeguard operations that may be disrupted if another thread executes certain sections of code before the operation is completed. Critical sections work like gates that allow only a single thread to enter at a time. Because they block the execution of other threads, overusing critical sections can seriously interfere with performance.

Critical sections must be global in scope so that they are available to all threads. In every thread, any action that could interfere with the operations safeguarded by the critical section should only take place after calling the Acquire (see page 3070) or Enter (see page 3071) method. Threads that do not use the critical section for relevant operations can introduce bugs.

See Also

TThread

TCOMCriticalSection

Using Critical Sections

35.1.4.1 TCriticalSection Members

The following tables list the members exposed by TCriticalSection.

Protected Fields

	Name	Description
	FSection (see page 3070)	This is FSection, a member of class TCriticalSection.

Public Methods

	Name	Description
	Acquire (see page 3070)	Binds the critical section to the calling thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.

	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3069)	Creates an instance of TCriticalSection (see page 3068).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3069)	Destroys an instance of TCriticalSection (see page 3068).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Enter (see page 3071)	Blocks other threads when the calling thread enters a thread-sensitive section.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	Leave (see page 3071)	Allows other threads to use the critical section.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3072)	Allows other threads to use the critical section.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	TryEnter (see page 3072)	This is TryEnter, a member of class TCriticalSection.
	UnitName (see page 3392)	Returns a string containing the unit name.

35.1.4.2 TCriticalSection.Create Constructor

Creates an instance of TCriticalSection (see page 3068).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCriticalSection();
```

Description

Call Create to create and initialize an instance of TCriticalSection (see page 3068). A TCriticalSection (see page 3068) object can be created for a single object that contains thread-sensitive code, or as a global object to be shared by several threads.

See Also

System::TObject::Create (see page 3366)

35.1.4.3 TCriticalSection.Destroy Destructor

Destroys an instance of TCriticalSection (see page 3068).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCriticalSection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCriticalSection (see page 3068) object is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

35.1.4.4 TCriticalSection Fields

The fields of the TCriticalSection class are listed here.

Protected Fields

	Name	Description
	FSection (see page 3070)	This is FSection, a member of class TCriticalSection.

35.1.4.4.1 TCriticalSection.FSection Field

Pascal

```
FSection: TRTLCriticalSection;
```

C++

```
TRTLCriticalSection FSection;
```

Description

This is FSection, a member of class TCriticalSection.

35.1.4.5 TCriticalSection Methods

The methods of the TCriticalSection class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3070)	Binds the critical section to the calling thread.
	Enter (see page 3071)	Blocks other threads when the calling thread enters a thread-sensitive section.
	Leave (see page 3071)	Allows other threads to use the critical section.
	Release (see page 3072)	Allows other threads to use the critical section.
	TryEnter (see page 3072)	This is TryEnter, a member of class TCriticalSection.

35.1.4.5.1 TCriticalSection.Acquire Method

Binds the critical section to the calling thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire to block all other threads from acquiring this critical section until the Release (see page 3072) or Leave (see page 3071) method is called. Acquire does the same thing as the Enter (see page 3071) method.

See Also

Enter (see page 3071)

Leave (see page 3071)

Release (see page 3072)

Using Critical Sections

35.1.4.5.2 TCriticalSection.Enter Method

Blocks other threads when the calling thread enters a thread-sensitive section.

Pascal

```
procedure Enter;
```

C++

```
__fastcall Enter();
```

Description

Call Enter to block all other threads from entering code protected by this critical section until the Leave (see page 3071) or Release (see page 3072) method is called. Enter calls the Acquire (see page 3070) method to bind the critical section to the calling thread.

See Also

Acquire (see page 3070)

Leave (see page 3071)

Release (see page 3072)

35.1.4.5.3 TCriticalSection.Leave Method

Allows other threads to use the critical section.

Pascal

```
procedure Leave;
```

C++

```
__fastcall Leave();
```

Description

Call Leave at the end of a thread-sensitive section of code. Each call to Leave should be balanced by a call to Enter (see page 3071) or Acquire (see page 3070) at the beginning of the thread-sensitive section. Leave calls Release (see page 3072) to allow other threads to acquire the critical section.

See Also

Acquire (see page 3070)

Enter (see page 3071)

Release (see page 3072)

35.1.4.5.4 TCriticalSection.Release Method

Allows other threads to use the critical section.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release when the thread no longer needs to block other threads from executing code that is protected by this critical section. Each call to Release should be balance by an earlier call to Acquire (see page 3070) or Enter (see page 3071) that blocked other threads from executing. Release does the same thing as the Leave (see page 3071) method.

See Also

Acquire (see page 3070)

Enter (see page 3071)

Leave (see page 3071)

Using Critical Sections

35.1.4.5.5 TCriticalSection.TryEnter Method

Pascal

```
function TryEnter: Boolean;
```

C++

```
__fastcall Boolean TryEnter();
```

Description

This is TryEnter, a member of class TCriticalSection.

35.1.5 TCriticalSectionHelper Class Helper

Class Hierarchy

SyncObjs.TCriticalSectionHelper

Pascal

```
TCriticalSectionHelper = class helper for TRTLCriticalSection;
```

C++

```
class TCriticalSectionHelper;
```

File

SyncObjs

Description

This is class SyncObjs.TCriticalSectionHelper.

35.1.5.1 TCriticalSectionHelper Members

The following tables list the members exposed by TCriticalSectionHelper.

Public Methods

	Name	Description
	Destroy (see page 3073)	This is Destroy, a member of class TCriticalSectionHelper.
	Enter (see page 3073)	This is Enter, a member of class TCriticalSectionHelper.
	Free (see page 3074)	This is Free, a member of class TCriticalSectionHelper.
	Initialize (see page 3074)	This is Initialize, a member of class TCriticalSectionHelper.
	Leave (see page 3074)	This is Leave, a member of class TCriticalSectionHelper.
	TryEnter (see page 3074)	This is TryEnter, a member of class TCriticalSectionHelper.

35.1.5.2 TCriticalSectionHelper Methods

The methods of the TCriticalSectionHelper class are listed here.

Public Methods

	Name	Description
	Destroy (see page 3073)	This is Destroy, a member of class TCriticalSectionHelper.
	Enter (see page 3073)	This is Enter, a member of class TCriticalSectionHelper.
	Free (see page 3074)	This is Free, a member of class TCriticalSectionHelper.
	Initialize (see page 3074)	This is Initialize, a member of class TCriticalSectionHelper.
	Leave (see page 3074)	This is Leave, a member of class TCriticalSectionHelper.
	TryEnter (see page 3074)	This is TryEnter, a member of class TCriticalSectionHelper.

35.1.5.2.1 TCriticalSectionHelper.Destroy Method

Pascal

```
procedure Destroy;
```

C++

```
__fastcall Destroy();
```

Description

This is Destroy, a member of class TCriticalSectionHelper.

35.1.5.2.2 TCriticalSectionHelper.Enter Method

Pascal

```
procedure Enter;
```

C++

```
__fastcall Enter();
```

Description

This is Enter, a member of class TCriticalSectionHelper.

35.1.5.2.3 TCriticalSectionHelper.Free Method

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

This is Free, a member of class TCriticalSectionHelper.

35.1.5.2.4 TCriticalSectionHelper.Initialize Method

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

This is Initialize, a member of class TCriticalSectionHelper.

35.1.5.2.5 TCriticalSectionHelper.Leave Method

Pascal

```
procedure Leave;
```

C++

```
__fastcall Leave();
```

Description

This is Leave, a member of class TCriticalSectionHelper.

35.1.5.2.6 TCriticalSectionHelper.TryEnter Method

Pascal

```
function TryEnter: Boolean;
```

C++

```
__fastcall Boolean TryEnter();
```

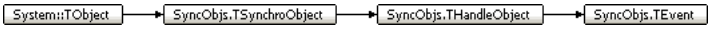
Description

This is TryEnter, a member of class TCriticalSectionHelper.

35.1.6 TEvent Class

TEvent represents an external event.

Class Hierarchy



Pascal

```
TEvent = class(THandleObject);
```

C++

```
class TEvent : public THandleObject;
```

File

SyncObjs

Description

Use TEvent to signal that an event has occurred or a state has been reached.

In a multi-threaded application, use TEvent to allow one thread to signal to other threads that an event has occurred.

The handle of a TEvent object can also be used to communicate with other processes, so that an application can coordinate the timing of events with other applications. For example, use the handle of a TEvent object to wait until another process is ready to transfer information.

In a single-threaded application, use TEvent to coordinate between sections of code that respond to asynchronous events such as system events.

See Also

TThread

Waiting for a Task to Be Completed

35.1.6.1 TEvent Members

The following tables list the members exposed by TEvent.

Protected Fields

	Name	Description
	FHandle (see page 3081)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3081)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3081)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3092)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3076)	This is the overview for the Create constructor overload.

	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3080)	Destroys an instance of THandleObject (see page 3078).
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3093)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ResetEvent (see page 3077)	Turns off the signal of the event object.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetEvent (see page 3078)	Turns on the signal of the event object.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	WaitFor (see page 3082)	This is WaitFor, a member of class THandleObject.
	WaitForMultiple (see page 3082)	This is WaitForMultiple, a member of class THandleObject.

Public Properties

	Name	Description
	Handle (see page 3082)	Specifies the handle of the Windows object used by THandleObject (see page 3078).
	LastError (see page 3083)	Indicates the Windows error code obtained from Handle (see page 3082).

35.1.6.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TEvent.Create (Boolean) (see page 3076)	Instantiates a TEvent (see page 3074) object to represent an event object.
	TEvent.Create (PSecurityAttributes, Boolean, Boolean, string, Boolean) (see page 3076)	Instantiates a TEvent (see page 3074) object to represent an event object.

35.1.6.2.1 TEvent.Create Constructor (Boolean)

Instantiates a TEvent (see page 3074) object to represent an event object.

Pascal

```
constructor Create(EventAttributes: PSecurityAttributes; ManualReset: Boolean; InitialState:
Boolean; const Name: string; UseCOMWait: Boolean = False); overload;
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TEvent(PSecurityAttributes EventAttributes, Boolean ManualReset, Boolean
InitialState, const UnicodeString Name, Boolean UseCOMWait = False);
__fastcall TEvent(Boolean UseCOMWait = False);
```

Description

Call Create to create a TEvent (see page 3074) object. Create can generate a new event object or can provide access to an existing named event object.

Set the EventAttributes parameter to specify the access rights and inheritance capabilities of the new event object. Calling Create with EventAttributes set to nil (Delphi) or NULL (C++) creates an event object with a default set of security attributes. If the TEvent (see page 3074) object is created to access an existing named event object, only the bInheritHandle field of EventAttributes is used, as the access rights were determined when the event object was created.

Set ManualReset to specify whether the signal of the TEvent (see page 3074) object can only be turned off by a call to the ResetEvent (see page 3077) method, or whether it is automatically reset when a single waiting thread is released. When ManualReset is true, the TEvent (see page 3074) signal stays set until the ResetEvent (see page 3077) method turns it off. When ManualReset is false, the signal is automatically turned off when only a single thread waits on the signal and that thread is released. If the TEvent (see page 3074) object is created to access an existing named event object, the ManualReset parameter is ignored.

Set InitialState to indicate whether the TEvent (see page 3074) object should be created with the signal set, or turned off. When InitialState is true, the TEvent (see page 3074) object is created with the signal set. If the TEvent (see page 3074) object is created to access an existing named event object, the InitialState parameter is ignored.

Set Name to provide a name for a new event object or to specify an existing named event object. If no other thread or process will need to access the event object to wait for its signal, Name can be left blank. Name can be a string of up to 260 characters, not including the backslash character (\). If Name is used to specify an existing event object, the value must match the name of the existing event in a case-sensitive comparison. If Name matches the name of an existing semaphore, mutex, or file-mapping object, the TEvent (see page 3074) object will be created with Handle (see page 3082) set to 0 and all method calls will fail.

See Also

Handle (see page 3082)

ResetEvent (see page 3077)

35.1.6.3 TEvent Methods

The methods of the TEvent class are listed here.

Public Methods

	Name	Description
	ResetEvent (see page 3077)	Turns off the signal of the event object.
	SetEvent (see page 3078)	Turns on the signal of the event object.

35.1.6.3.1 TEvent.ResetEvent Method

Turns off the signal of the event object.

Pascal

```
procedure ResetEvent;
```

C++

```
__fastcall ResetEvent();
```

Description

Call ResetEvent to signal that the event has not occurred or the desired state was not achieved. Other threads or processes that wait for the event will wait until the signal is turned on again by the SetEvent (see page 3078) method, the event object is destroyed, or the waiting process times out.

See Also

SetEvent (see page 3078)

WaitFor

Waiting for a Task to Be Completed

35.1.6.3.2 TEvent.SetEvent Method

Turns on the signal of the event object.

Pascal

```
procedure SetEvent;
```

C++

```
__fastcall SetEvent();
```

Description

Call SetEvent to signal that the event has occurred or the desired state was achieved. Other threads or processes that wait for the event will resume execution with a return value indicating that the signal was set.

See Also

ResetEvent (see page 3077)

WaitFor

Waiting for a Task to Be Completed

35.1.7 THandleObject Class

THandleObject is the base class for all thread synchronization objects that use a handle.

Class Hierarchy

Pascal

```
THandleObject = class(TSynchroObject);
```

C++

```
class THandleObject : public TSynchroObject;
```

File

SyncObjs

Description

Use THandleObject as a base class when defining an object that uses a handle when coordinating the execution of different threads in a multi-threaded application. Handled synchronization objects include event objects that allow threads to signal when critical events have occurred.

Do not create instances of THandleObject. THandleObject is a base class that introduces properties for managing the handle of a synchronization object but does not provide a mechanism for acquiring that handle. Descendants of THandleObject must acquire the appropriate type of handle.

See Also

TThread

TEvent (see page 3074)

35.1.7.1 THandleObject Members

The following tables list the members exposed by THandleObject.

Protected Fields

	Name	Description
	FHandle (see page 3081)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3081)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3081)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3092)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3080)	This is Create, a member of class THandleObject.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3080)	Destroys an instance of THandleObject (see page 3078).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3093)	Provides the interface for a method that releases the synchronization object for use by other threads.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
V	WaitFor (see page 3082)	This is WaitFor, a member of class THandleObject.
	WaitForMultiple (see page 3082)	This is WaitForMultiple, a member of class THandleObject.

Public Properties

	Name	Description
R	Handle (see page 3082)	Specifies the handle of the Windows object used by THandleObject (see page 3078).
R	LastError (see page 3083)	Indicates the Windows error code obtained from Handle (see page 3082).

35.1.7.2 THandleObject.Create Constructor

Pascal

```
constructor Create(UseCOMWait: Boolean = False);
```

C++

```
__fastcall THandleObject(Boolean UseCOMWait = False);
```

Description

This is Create, a member of class THandleObject.

35.1.7.3 THandleObject.Destroy Destructor

Destroys an instance of THandleObject (see page 3078).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THandleObject();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THandleObject (see page 3078) reference is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

Handle (see page 3082)

35.1.7.4 THandleObject Fields

The fields of the THandleObject class are listed here.

Protected Fields

	Name	Description
	FHandle (see page 3081)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3081)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3081)	This is FUseCOMWait, a member of class THandleObject.

35.1.7.4.1 THandleObject.FHandle Field

Pascal

```
FHandle: THandle;
```

C++

```
THandle FHandle;
```

Description

This is FHandle, a member of class THandleObject.

35.1.7.4.2 THandleObject.FLastError Field

Pascal

```
FLastError: Integer;
```

C++

```
int FLastError;
```

Description

This is FLastError, a member of class THandleObject.

35.1.7.4.3 THandleObject.FUseCOMWait Field

Pascal

```
FUseCOMWait: Boolean;
```

C++

```
Boolean FUseCOMWait;
```

Description

This is FUseCOMWait, a member of class THandleObject.

35.1.7.5 THandleObject Methods

The methods of the THandleObject class are listed here.

Public Methods

	Name	Description
	WaitFor (see page 3082)	This is WaitFor, a member of class THandleObject.

	WaitForMultiple (see page 3082)	This is WaitForMultiple, a member of class THandleObject.

35.1.7.5.1 THandleObject.WaitFor Method

Pascal

```
function WaitFor(Timeout: LongWord): TWaitResult; virtual;
```

C++

```
virtual __fastcall TWaitResult WaitFor(LongWord Timeout);
```

Description

This is WaitFor, a member of class THandleObject.

35.1.7.5.2 THandleObject.WaitForMultiple Method

Pascal

```
class function WaitForMultiple(const HandleObjs: THandleObjectArray; Timeout: LongWord; AAll:
Boolean; out SignaledObj: THandleObject; UseCOMWait: Boolean = False; Len: Integer = 0):
TWaitResult;
```

C++

```
__fastcall TWaitResult WaitForMultiple(const THandleObjectArray HandleObjs, LongWord Timeout,
Boolean AAll, THandleObject SignaledObj, Boolean UseCOMWait = False, int Len = 0);
```

Description

This is WaitForMultiple, a member of class THandleObject.

35.1.7.6 THandleObject Properties

The properties of the THandleObject class are listed here.

Public Properties

	Name	Description
R	Handle (see page 3082)	Specifies the handle of the Windows object used by THandleObject (see page 3078).
R	LastError (see page 3083)	Indicates the Windows error code obtained from Handle (see page 3082).

35.1.7.6.1 THandleObject.Handle Property

Specifies the handle of the Windows object used by THandleObject (see page 3078).

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Read (see page 3727) Handle to obtain the Windows handle for the synchronization object. Use Handle when making Windows API calls that require a handle to the synchronization object.

THandleObject (see page 3078) provides no mechanism for setting the Handle property. Descendants of THandleObject (see page 3078) set the Handle property to the handle for whatever Windows synchronization object they represent.

See Also

Destroy (see page 3080)

35.1.7.6.2 THandleObject.LastError Property

Indicates the Windows error code obtained from Handle (see page 3082).

Pascal

```
property LastError: Integer;
```

C++

```
__property int LastError;
```

Description

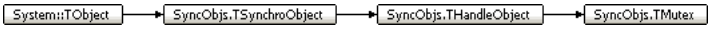
Read (see page 3727) LastError when an operation of the THandleObject (see page 3078) fails. LastError returns the Windows error code that represents the reason for the failure. If no operation failed, LastError is 0.

See Also

Handle (see page 3082)

35.1.8 TMutex Class

Class Hierarchy



Pascal

```
TMutex = class(THandleObject);
```

C++

```
class TMutex : public THandleObject;
```

File

SyncObjs

Description

This is class SyncObjs.TMutex.

35.1.8.1 TMutex Members

The following tables list the members exposed by TMutex.

Protected Fields

	Name	Description
	FHandle (see page 3081)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3081)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3081)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3086)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3084)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3080)	Destroys an instance of THandleObject (see page 3078).
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3086)	Provides the interface for a method that releases the synchronization object for use by other threads.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	WaitFor (see page 3082)	This is WaitFor, a member of class THandleObject.
	WaitForMultiple (see page 3082)	This is WaitForMultiple, a member of class THandleObject.

Public Properties

	Name	Description
	Handle (see page 3082)	Specifies the handle of the Windows object used by THandleObject (see page 3078).
	LastError (see page 3083)	Indicates the Windows error code obtained from Handle (see page 3082).

35.1.8.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TMutex.Create (Boolean) (see page 3085)	This is Create, a member of class TMutex.
	TMutex.Create (LongWord, Boolean, string, Boolean) (see page 3085)	This is Create, a member of class TMutex.
	TMutex.Create (PSecurityAttributes, Boolean, string, Boolean) (see page 3085)	This is Create, a member of class TMutex.

35.1.8.2.1 TMutex.Create Constructor (Boolean)

Pascal

```
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

35.1.8.2.2 TMutex.Create Constructor (LongWord, Boolean, string, Boolean)

Pascal

```
constructor Create(DesiredAccess: LongWord; InheritHandle: Boolean; const Name: string;
UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(LongWord DesiredAccess, Boolean InheritHandle, const UnicodeString Name,
Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

35.1.8.2.3 TMutex.Create Constructor (PSecurityAttributes, Boolean, string, Boolean)

Pascal

```
constructor Create(MutexAttributes: PSecurityAttributes; InitialOwner: Boolean; const Name:
string; UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(PSecurityAttributes MutexAttributes, Boolean InitialOwner, const
UnicodeString Name, Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

35.1.8.3 TMutex Methods

The methods of the TMutex class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3086)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3086)	Provides the interface for a method that releases the synchronization object for use by other threads.

35.1.8.3.1 TMutex.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3093)

35.1.8.3.2 TMutex.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

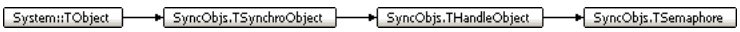
Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3092)

35.1.9 TSemaphore Class

Class Hierarchy



Pascal

```
TSemaphore = class(THandleObject);
```

C++

```
class TSemaphore : public THandleObject;
```

File

SyncObjs

Description

This is class SyncObjs.TSemaphore.

35.1.9.1 TSemaphore Members

The following tables list the members exposed by TSemaphore.

Protected Fields

	Name	Description
	FHandle (see page 3081)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3081)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3081)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3089)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3088)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3080)	Destroys an instance of THandleObject (see page 3078).
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3089)	This is the overview for the Release method overload.

	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	WaitFor (see page 3082)	This is WaitFor, a member of class THandleObject.
	WaitForMultiple (see page 3082)	This is WaitForMultiple, a member of class THandleObject.

Public Properties

	Name	Description
	Handle (see page 3082)	Specifies the handle of the Windows object used by THandleObject (see page 3078).
	LastError (see page 3083)	Indicates the Windows error code obtained from Handle (see page 3082).

35.1.9.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TSemaphore.Create (Boolean) (see page 3088)	This is Create, a member of class TSemaphore.
	TSemaphore.Create (LongWord, Boolean, string, Boolean) (see page 3088)	This is Create, a member of class TSemaphore.
	TSemaphore.Create (PSecurityAttributes, Integer, Integer, string, Boolean) (see page 3089)	This is Create, a member of class TSemaphore.

35.1.9.2.1 TSemaphore.Create Constructor (Boolean)

Pascal

```
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TSemaphore(Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TSemaphore.

35.1.9.2.2 TSemaphore.Create Constructor (LongWord, Boolean, string, Boolean)

Pascal

```
constructor Create(DesiredAccess: LongWord; InheritHandle: Boolean; const Name: string;
UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TSemaphore(LongWord DesiredAccess, Boolean InheritHandle, const UnicodeString Name,
Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TSemaphore.

35.1.9.2.3 TSemaphore.Create Constructor (PSecurityAttributes, Integer, Integer, string, Boolean)

Pascal

```
constructor Create(SemaphoreAttributes: PSecurityAttributes; AInitialCount: Integer;
AMaximumCount: Integer; const Name: string; UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TSemaphore(PSecurityAttributes SemaphoreAttributes, int AInitialCount, int
AMaximumCount, const UnicodeString Name, Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TSemaphore.

35.1.9.3 TSemaphore Methods

The methods of the TSemaphore class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3089)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3089)	This is the overview for the Release method overload.

35.1.9.3.1 TSemaphore.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3093)

35.1.9.3.2 Release Method

This is the overview for the Release method overload.

Overload List

	Name	Description
	TSemaphore.Release () (see page 3090)	Provides the interface for a method that releases the synchronization object for use by other threads.

	TSemaphore.Release (Integer) (see page 3090)	This is Release, a member of class TSemaphore.

35.1.9.3.2.1 TSemaphore.Release Method ()

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; override; overload;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3092)

35.1.9.3.2.2 TSemaphore.Release Method (Integer)

Pascal

```
function Release(AReleaseCount: Integer): Integer; overload; reintroduce;
```

C++

```
__fastcall int Release(int AReleaseCount);
```

Description

This is Release, a member of class TSemaphore.

35.1.10 TSimpleEvent Class

TSimpleEvent represents an unnamed manual event object.

Class Hierarchy

Pascal

```
TSimpleEvent = class(TEvent);
```

C++

```
class TSimpleEvent : public TEvent;
```

File

SyncObjs

Description

Use TSimpleEvent to coordinate between threads that all have access to the same simple event object. TSimpleEvent can also be used in a single-threaded application to coordinate sections of code that respond to different asynchronous events such as system messages or user actions.

Because TSimpleEvent represents an unnamed event object, a single TSimpleEvent object must be within the scope of all code that uses it. To coordinate with code that is out of scope, or with other applications, use a named TEvent (see page 3074) object.

See Also

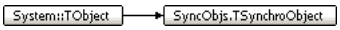
TThread

TEvent (see page 3074)

35.1.11 TSynchroObject Class

TSynchroObject is the base class for all thread synchronization objects.

Class Hierarchy



Pascal

```
TSynchroObject = class(TObject);
```

C++

```
class TSynchroObject : public TObject;
```

File

SyncObjs

Description

Use TSynchroObject as a base class when defining an object that coordinates the execution of different threads in a multi-threaded application. Synchronization objects include critical sections that temporarily block the execution of other threads which may interfere with the successful completion of a section of code, and event objects that allow threads to signal when critical events have occurred.

See Also

TCriticalSection (see page 3068)

TEvent (see page 3074)

TThread

35.1.11.1 TSynchroObject Members

The following tables list the members exposed by TSynchroObject.

Public Methods

	Name	Description
	Acquire (see page 3092)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3093)	Provides the interface for a method that releases the synchronization object for use by other threads.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

35.1.11.2 TSynchroObject Methods

The methods of the TSynchroObject class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3092)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3093)	Provides the interface for a method that releases the synchronization object for use by other threads.

35.1.11.2.1 TSynchroObject.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; virtual;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject (see page

3091) does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3093)

35.1.11.2.2 TSynchroObject.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; virtual;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject (see page 3091) does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3092)

35.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWaitResult (see page 3094)	TWaitResult indicates the reason an event object stopped waiting for its signal.

35.2.1 SyncObjs.TWaitResult Enumeration

TWaitResult indicates the reason an event object stopped waiting for its signal.

Pascal

```
TWaitResult = (
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError,
  wrIOCompletion
);
```

C++

```
enum TWaitResult {
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError,
  wrIOCompletion
};
```

File

SyncObjs

Description

TWaitResult includes the following values:

Value	Meaning
wrSignaled	The signal of the event object was set.
wrTimeOut	The time specified by the TimeOut parameter elapsed without the signal being set.
wrAbandoned	The event object was destroyed before the TimeOut period elapsed.
wrError	An error occurred while waiting. Check the LastError property for an error code giving more information.

35.3 Types

The following table lists types in this documentation.

Types

Name	Description
THandleObjectArray (see page 3095)	This is type SyncObjs.THandleObjectArray.

35.3.1 SyncObjs.THandleObjectArray Type

Pascal

```
THandleObjectArray = array of THandleObject;
```

C++

```
array of THandleObject THandleObjectArray;
```

File

SyncObjs

Description

This is type SyncObjs.THandleObjectArray.

36 SysConst Namespace

This is namespace SysConst.

Topics

Name	Description
Constants (see page 3101)	The following table lists constants in this documentation.

Constants

Name	Description
SAbstractError (see page 3104)	This is constant SysConst.SAbstractError.
SAccessDenied (see page 3104)	This is constant SysConst.SAccessDenied.
SAccessViolationArg3 (see page 3104)	This is constant SysConst.SAccessViolationArg3.
SAccessViolationNoArg (see page 3105)	This is constant SysConst.SAccessViolationNoArg.
SArgumentMissing (see page 3105)	This is constant SysConst.SArgumentMissing.
SAssertError (see page 3105)	This is constant SysConst.SAssertError.
SAssertionFailed (see page 3106)	This is constant SysConst.SAssertionFailed.
SByteIndexOutOfBounds (see page 3106)	This is constant SysConst.SByteIndexOutOfBounds.
SCannotCreateDir (see page 3106)	This is constant SysConst.SCannotCreateDir.
SCannotReadPackageInfo (see page 3107)	This is constant SysConst.SCannotReadPackageInfo.
SCharIndexOutOfBounds (see page 3107)	This is constant SysConst.SCharIndexOutOfBounds.
SCodesetConversionError (see page 3107)	This is constant SysConst.SCodesetConversionError.
SControlC (see page 3108)	This is constant SysConst.SControlC.
SDateEncodeError (see page 3108)	This is constant SysConst.SDateEncodeError.
SDiskFull (see page 3108)	This is constant SysConst.SDiskFull.
SDispatchError (see page 3109)	This is constant SysConst.SDispatchError.
SDivByZero (see page 3109)	This is constant SysConst.SDivByZero.
SDuplicatePackageUnit (see page 3109)	This is constant SysConst.SDuplicatePackageUnit.
SEndOfFile (see page 3110)	This is constant SysConst.SEndOfFile.
sErrorLoadingPackage (see page 3110)	This is constant SysConst.sErrorLoadingPackage.
SException (see page 3110)	This is constant SysConst.SException.
SExceptTitle (see page 3111)	This is constant SysConst.SExceptTitle.
SExternalException (see page 3111)	This is constant SysConst.SExternalException.
SFileNotFound (see page 3111)	This is constant SysConst.SFileNotFound.
SFormatTooLong (see page 3112)	This is constant SysConst.SFormatTooLong.
SInOutError (see page 3112)	This is constant SysConst.SInOutError.
SIntfCastError (see page 3112)	This is constant SysConst.SIntfCastError.

SIntOverflow (see page 3113)	This is constant SysConst.SIntOverflow.
SInvalidBoolean (see page 3113)	This is constant SysConst.SInvalidBoolean.
SInvalidCast (see page 3113)	This is constant SysConst.SInvalidCast.
SInvalidCharCount (see page 3114)	This is constant SysConst.SInvalidCharCount.
SInvalidCodePage (see page 3114)	This is constant SysConst.SInvalidCodePage.
SInvalidCurrency (see page 3114)	This is constant SysConst.SInvalidCurrency.
SInvalidDate (see page 3115)	This is constant SysConst.SInvalidDate.
SInvalidDateTime (see page 3115)	This is constant SysConst.SInvalidDateTime.
SInvalidDateTimeFloat (see page 3115)	This is constant SysConst.SInvalidDateTimeFloat.
SInvalidDestinationArray (see page 3116)	This is constant SysConst.SInvalidDestinationArray.
SInvalidDestinationIndex (see page 3116)	This is constant SysConst.SInvalidDestinationIndex.
SInvalidFilename (see page 3116)	This is constant SysConst.SInvalidFilename.
SInvalidFloat (see page 3117)	This is constant SysConst.SInvalidFloat.
SInvalidFormat (see page 3117)	This is constant SysConst.SInvalidFormat.
SInvalidGUID (see page 3117)	This is constant SysConst.SInvalidGUID.
SInvalidInput (see page 3118)	This is constant SysConst.SInvalidInput.
SInvalidInteger (see page 3118)	This is constant SysConst.SInvalidInteger.
SInvalidOp (see page 3118)	This is constant SysConst.SInvalidOp.
SInvalidPackageFile (see page 3119)	This is constant SysConst.SInvalidPackageFile.
SInvalidPackageHandle (see page 3119)	This is constant SysConst.SInvalidPackageHandle.
SInvalidPointer (see page 3119)	This is constant SysConst.SInvalidPointer.
SInvalidSourceArray (see page 3120)	This is constant SysConst.SInvalidSourceArray.
SInvalidTime (see page 3120)	This is constant SysConst.SInvalidTime.
SInvalidTimeStamp (see page 3120)	This is constant SysConst.SInvalidTimeStamp.
SInvalidVarCast (see page 3121)	This is constant SysConst.SInvalidVarCast.
SInvalidVarNullOp (see page 3121)	This is constant SysConst.SInvalidVarNullOp.
SInvalidVarOp (see page 3121)	This is constant SysConst.SInvalidVarOp.
SInvalidVarOpWithHResult (see page 3122)	This is constant SysConst.SInvalidVarOpWithHResult.
SInvalidVarOpWithHResultWithPrefix (see page 3122)	This is constant SysConst.SInvalidVarOpWithHResultWithPrefix.
SLongDayNameFri (see page 3122)	This is constant SysConst.SLongDayNameFri.
SLongDayNameMon (see page 3123)	This is constant SysConst.SLongDayNameMon.
SLongDayNameSat (see page 3123)	This is constant SysConst.SLongDayNameSat.
SLongDayNameSun (see page 3123)	This is constant SysConst.SLongDayNameSun.
SLongDayNameThu (see page 3124)	This is constant SysConst.SLongDayNameThu.
SLongDayNameTue (see page 3124)	This is constant SysConst.SLongDayNameTue.
SLongDayNameWed (see page 3124)	This is constant SysConst.SLongDayNameWed.
SLongMonthNameApr (see page 3125)	This is constant SysConst.SLongMonthNameApr.
SLongMonthNameAug (see page 3125)	This is constant SysConst.SLongMonthNameAug.
SLongMonthNameDec (see page 3125)	This is constant SysConst.SLongMonthNameDec.
SLongMonthNameFeb (see page 3126)	This is constant SysConst.SLongMonthNameFeb.
SLongMonthNameJan (see page 3126)	This is constant SysConst.SLongMonthNameJan.
SLongMonthNameJul (see page 3126)	This is constant SysConst.SLongMonthNameJul.
SLongMonthNameJun (see page 3127)	This is constant SysConst.SLongMonthNameJun.
SLongMonthNameMar (see page 3127)	This is constant SysConst.SLongMonthNameMar.
SLongMonthNameMay (see page 3127)	This is constant SysConst.SLongMonthNameMay.
SLongMonthNameNov (see page 3128)	This is constant SysConst.SLongMonthNameNov.
SLongMonthNameOct (see page 3128)	This is constant SysConst.SLongMonthNameOct.
SLongMonthNameSep (see page 3128)	This is constant SysConst.SLongMonthNameSep.
SModuleAccessViolation (see page 3129)	This is constant SysConst.SModuleAccessViolation.
SMonitorLockException (see page 3129)	This is constant SysConst.SMonitorLockException.
SNL (see page 3129)	This is constant SysConst.SNL.
SNoMonitorSupportException (see page 3130)	This is constant SysConst.SNoMonitorSupportException.
SOperationAborted (see page 3130)	This is constant SysConst.SOperationAborted.
SOSError (see page 3130)	This is constant SysConst.SOSError.

SOutOfMemory (see page 3131)	This is constant SysConst.SOutOfMemory.
SOverflow (see page 3131)	This is constant SysConst.SOverflow.
SPrivilege (see page 3131)	This is constant SysConst.SPrivilege.
SQuit (see page 3132)	This is constant SysConst.SQuit.
SRangeError (see page 3132)	This is constant SysConst.SRangeError.
SReadAccess (see page 3132)	This is constant SysConst.SReadAccess.
SResultTooLong (see page 3133)	This is constant SysConst.SResultTooLong.
SSafecallException (see page 3133)	This is constant SysConst.SSafecallException.
SShortDayNameFri (see page 3133)	This is constant SysConst.SShortDayNameFri.
SShortDayNameMon (see page 3134)	This is constant SysConst.SShortDayNameMon.
SShortDayNameSat (see page 3134)	This is constant SysConst.SShortDayNameSat.
SShortDayNameSun (see page 3134)	This is constant SysConst.SShortDayNameSun.
SShortDayNameThu (see page 3135)	This is constant SysConst.SShortDayNameThu.
SShortDayNameTue (see page 3135)	This is constant SysConst.SShortDayNameTue.
SShortDayNameWed (see page 3135)	This is constant SysConst.SShortDayNameWed.
SShortMonthNameApr (see page 3136)	This is constant SysConst.SShortMonthNameApr.
SShortMonthNameAug (see page 3136)	This is constant SysConst.SShortMonthNameAug.
SShortMonthNameDec (see page 3136)	This is constant SysConst.SShortMonthNameDec.
SShortMonthNameFeb (see page 3137)	This is constant SysConst.SShortMonthNameFeb.
SShortMonthNameJan (see page 3137)	This is constant SysConst.SShortMonthNameJan.
SShortMonthNameJul (see page 3137)	This is constant SysConst.SShortMonthNameJul.
SShortMonthNameJun (see page 3138)	This is constant SysConst.SShortMonthNameJun.
SShortMonthNameMar (see page 3138)	This is constant SysConst.SShortMonthNameMar.
SShortMonthNameMay (see page 3138)	This is constant SysConst.SShortMonthNameMay.
SShortMonthNameNov (see page 3139)	This is constant SysConst.SShortMonthNameNov.
SShortMonthNameOct (see page 3139)	This is constant SysConst.SShortMonthNameOct.
SShortMonthNameSep (see page 3139)	This is constant SysConst.SShortMonthNameSep.
SStackOverflow (see page 3140)	This is constant SysConst.SStackOverflow.
STimeEncodeError (see page 3140)	This is constant SysConst.STimeEncodeError.
STooManyOpenFiles (see page 3140)	This is constant SysConst.STooManyOpenFiles.
SUnderflow (see page 3141)	This is constant SysConst.SUnderflow.
SUnknown (see page 3141)	This is constant SysConst.SUnknown.
sUnknownFromType (see page 3141)	This is constant SysConst.sUnknownFromType.
sUnknownToType (see page 3142)	This is constant SysConst.sUnknownToType.
SUnkOSError (see page 3142)	This is constant SysConst.SUnkOSError.
SUnkWin32Error (see page 3142)	This is constant SysConst.SUnkWin32Error.
SVarArrayBounds (see page 3143)	This is constant SysConst.SVarArrayBounds.
SVarArrayCreate (see page 3143)	This is constant SysConst.SVarArrayCreate.
SVarArrayLocked (see page 3143)	This is constant SysConst.SVarArrayLocked.
SVarArrayWithHResult (see page 3144)	This is constant SysConst.SVarArrayWithHResult.
SVarBadType (see page 3144)	This is constant SysConst.SVarBadType.
SVarDataCastToRecursing (see page 3144)	This is constant SysConst.SVarDataCastToRecursing.
SVarDataClearRecursing (see page 3145)	This is constant SysConst.SVarDataClearRecursing.
SVarDataCopyNoIndRecursing (see page 3145)	This is constant SysConst.SVarDataCopyNoIndRecursing.
SVarDataCopyRecursing (see page 3145)	This is constant SysConst.SVarDataCopyRecursing.
SVarDataInitRecursing (see page 3146)	This is constant SysConst.SVarDataInitRecursing.
SVarInvalid (see page 3146)	This is constant SysConst.SVarInvalid.
SVarIsEmpty (see page 3146)	This is constant SysConst.SVarIsEmpty.
SVarNotArray (see page 3147)	This is constant SysConst.SVarNotArray.
SVarNotImplemented (see page 3147)	This is constant SysConst.SVarNotImplemented.
SVarOutOfMemory (see page 3147)	This is constant SysConst.SVarOutOfMemory.
SVarOverflow (see page 3148)	This is constant SysConst.SVarOverflow.
SVarTypeAlreadyUsed (see page 3148)	This is constant SysConst.SVarTypeAlreadyUsed.
SVarTypeAlreadyUsedWithPrefix (see page 3148)	This is constant SysConst.SVarTypeAlreadyUsedWithPrefix.

SVarTypeConvertOverflow (see page 3149)	This is constant SysConst.SVarTypeConvertOverflow.
SVarTypeCouldNotConvert (see page 3149)	This is constant SysConst.SVarTypeCouldNotConvert.
SVarTypeNotUsable (see page 3149)	This is constant SysConst.SVarTypeNotUsable.
SVarTypeNotUsableWithPrefix (see page 3150)	This is constant SysConst.SVarTypeNotUsableWithPrefix.
SVarTypeOutOfRange (see page 3150)	This is constant SysConst.SVarTypeOutOfRange.
SVarTypeOutOfRangeWithPrefix (see page 3150)	This is constant SysConst.SVarTypeOutOfRangeWithPrefix.
SVarTypeRangeCheck1 (see page 3151)	This is constant SysConst.SVarTypeRangeCheck1.
SVarTypeRangeCheck2 (see page 3151)	This is constant SysConst.SVarTypeRangeCheck2.
SVarTypeTooManyCustom (see page 3151)	This is constant SysConst.SVarTypeTooManyCustom.
SVarTypeUnknown (see page 3152)	This is constant SysConst.SVarTypeUnknown.
SVarUnexpected (see page 3152)	This is constant SysConst.SVarUnexpected.
SWin32Error (see page 3152)	This is constant SysConst.SWin32Error.
SWriteAccess (see page 3153)	This is constant SysConst.SWriteAccess.
SZeroDivide (see page 3153)	This is constant SysConst.SZeroDivide.

36.1 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SAbstractError (see page 3104)	This is constant SysConst.SAbstractError.
SAccessDenied (see page 3104)	This is constant SysConst.SAccessDenied.
SAccessViolationArg3 (see page 3104)	This is constant SysConst.SAccessViolationArg3.
SAccessViolationNoArg (see page 3105)	This is constant SysConst.SAccessViolationNoArg.
SArgumentMissing (see page 3105)	This is constant SysConst.SArgumentMissing.
SAssertError (see page 3105)	This is constant SysConst.SAssertError.
SAssertionFailed (see page 3106)	This is constant SysConst.SAssertionFailed.
SByteIndexOutOfBounds (see page 3106)	This is constant SysConst.SByteIndexOutOfBounds.
SCannotCreateDir (see page 3106)	This is constant SysConst.SCannotCreateDir.
SCannotReadPackageInfo (see page 3107)	This is constant SysConst.SCannotReadPackageInfo.
SCharIndexOutOfBounds (see page 3107)	This is constant SysConst.SCharIndexOutOfBounds.
SCodesetConversionError (see page 3107)	This is constant SysConst.SCodesetConversionError.
SControlC (see page 3108)	This is constant SysConst.SControlC.
SDateEncodeError (see page 3108)	This is constant SysConst.SDateEncodeError.
SDiskFull (see page 3108)	This is constant SysConst.SDiskFull.
SDispatchError (see page 3109)	This is constant SysConst.SDispatchError.
SDivByZero (see page 3109)	This is constant SysConst.SDivByZero.
SDuplicatePackageUnit (see page 3109)	This is constant SysConst.SDuplicatePackageUnit.
SEndOfFile (see page 3110)	This is constant SysConst.SEndOfFile.
sErrorLoadingPackage (see page 3110)	This is constant SysConst.sErrorLoadingPackage.
SException (see page 3110)	This is constant SysConst.SException.
SExceptTitle (see page 3111)	This is constant SysConst.SExceptTitle.
SExternalException (see page 3111)	This is constant SysConst.SExternalException.
SFileNotFound (see page 3111)	This is constant SysConst.SFileNotFound.
SFormatTooLong (see page 3112)	This is constant SysConst.SFormatTooLong.
SInOutError (see page 3112)	This is constant SysConst.SInOutError.
SIntfCastError (see page 3112)	This is constant SysConst.SIntfCastError.
SIntOverflow (see page 3113)	This is constant SysConst.SIntOverflow.
SInvalidBoolean (see page 3113)	This is constant SysConst.SInvalidBoolean.
SInvalidCast (see page 3113)	This is constant SysConst.SInvalidCast.
SInvalidCharCount (see page 3114)	This is constant SysConst.SInvalidCharCount.
SInvalidCodePage (see page 3114)	This is constant SysConst.SInvalidCodePage.
SInvalidCurrency (see page 3114)	This is constant SysConst.SInvalidCurrency.
SInvalidDate (see page 3115)	This is constant SysConst.SInvalidDate.
SInvalidDateTime (see page 3115)	This is constant SysConst.SInvalidDateTime.
SInvalidDateTimeFloat (see page 3115)	This is constant SysConst.SInvalidDateTimeFloat.
SInvalidDestinationArray (see page 3116)	This is constant SysConst.SInvalidDestinationArray.
SInvalidDestinationIndex (see page 3116)	This is constant SysConst.SInvalidDestinationIndex.
SInvalidFilename (see page 3116)	This is constant SysConst.SInvalidFilename.
SInvalidFloat (see page 3117)	This is constant SysConst.SInvalidFloat.
SInvalidFormat (see page 3117)	This is constant SysConst.SInvalidFormat.
SInvalidGUID (see page 3117)	This is constant SysConst.SInvalidGUID.
SInvalidInput (see page 3118)	This is constant SysConst.SInvalidInput.
SInvalidInteger (see page 3118)	This is constant SysConst.SInvalidInteger.

SInvalidOp (see page 3118)	This is constant SysConst.SInvalidOp.
SInvalidPackageFile (see page 3119)	This is constant SysConst.SInvalidPackageFile.
SInvalidPackageHandle (see page 3119)	This is constant SysConst.SInvalidPackageHandle.
SInvalidPointer (see page 3119)	This is constant SysConst.SInvalidPointer.
SInvalidSourceArray (see page 3120)	This is constant SysConst.SInvalidSourceArray.
SInvalidTime (see page 3120)	This is constant SysConst.SInvalidTime.
SInvalidTimeStamp (see page 3120)	This is constant SysConst.SInvalidTimeStamp.
SInvalidVarCast (see page 3121)	This is constant SysConst.SInvalidVarCast.
SInvalidVarNullOp (see page 3121)	This is constant SysConst.SInvalidVarNullOp.
SInvalidVarOp (see page 3121)	This is constant SysConst.SInvalidVarOp.
SInvalidVarOpWithHResult (see page 3122)	This is constant SysConst.SInvalidVarOpWithHResult.
SInvalidVarOpWithHResultWithPrefix (see page 3122)	This is constant SysConst.SInvalidVarOpWithHResultWithPrefix.
SLongDayNameFri (see page 3122)	This is constant SysConst.SLongDayNameFri.
SLongDayNameMon (see page 3123)	This is constant SysConst.SLongDayNameMon.
SLongDayNameSat (see page 3123)	This is constant SysConst.SLongDayNameSat.
SLongDayNameSun (see page 3123)	This is constant SysConst.SLongDayNameSun.
SLongDayNameThu (see page 3124)	This is constant SysConst.SLongDayNameThu.
SLongDayNameTue (see page 3124)	This is constant SysConst.SLongDayNameTue.
SLongDayNameWed (see page 3124)	This is constant SysConst.SLongDayNameWed.
SLongMonthNameApr (see page 3125)	This is constant SysConst.SLongMonthNameApr.
SLongMonthNameAug (see page 3125)	This is constant SysConst.SLongMonthNameAug.
SLongMonthNameDec (see page 3125)	This is constant SysConst.SLongMonthNameDec.
SLongMonthNameFeb (see page 3126)	This is constant SysConst.SLongMonthNameFeb.
SLongMonthNameJan (see page 3126)	This is constant SysConst.SLongMonthNameJan.
SLongMonthNameJul (see page 3126)	This is constant SysConst.SLongMonthNameJul.
SLongMonthNameJun (see page 3127)	This is constant SysConst.SLongMonthNameJun.
SLongMonthNameMar (see page 3127)	This is constant SysConst.SLongMonthNameMar.
SLongMonthNameMay (see page 3127)	This is constant SysConst.SLongMonthNameMay.
SLongMonthNameNov (see page 3128)	This is constant SysConst.SLongMonthNameNov.
SLongMonthNameOct (see page 3128)	This is constant SysConst.SLongMonthNameOct.
SLongMonthNameSep (see page 3128)	This is constant SysConst.SLongMonthNameSep.
SModuleAccessViolation (see page 3129)	This is constant SysConst.SModuleAccessViolation.
SMonitorLockException (see page 3129)	This is constant SysConst.SMonitorLockException.
SNL (see page 3129)	This is constant SysConst.SNL.
SNoMonitorSupportException (see page 3130)	This is constant SysConst.SNoMonitorSupportException.
SOperationAborted (see page 3130)	This is constant SysConst.SOperationAborted.
SOSError (see page 3130)	This is constant SysConst.SOSError.
SOutOfMemory (see page 3131)	This is constant SysConst.SOutOfMemory.
SOverflow (see page 3131)	This is constant SysConst.SOverflow.
SPrivilege (see page 3131)	This is constant SysConst.SPrivilege.
SQuit (see page 3132)	This is constant SysConst.SQuit.
SRangeError (see page 3132)	This is constant SysConst.SRangeError.
SReadAccess (see page 3132)	This is constant SysConst.SReadAccess.
SResultTooLong (see page 3133)	This is constant SysConst.SResultTooLong.
SSafecallException (see page 3133)	This is constant SysConst.SSafecallException.
SShortDayNameFri (see page 3133)	This is constant SysConst.SShortDayNameFri.
SShortDayNameMon (see page 3134)	This is constant SysConst.SShortDayNameMon.
SShortDayNameSat (see page 3134)	This is constant SysConst.SShortDayNameSat.
SShortDayNameSun (see page 3134)	This is constant SysConst.SShortDayNameSun.
SShortDayNameThu (see page 3135)	This is constant SysConst.SShortDayNameThu.
SShortDayNameTue (see page 3135)	This is constant SysConst.SShortDayNameTue.
SShortDayNameWed (see page 3135)	This is constant SysConst.SShortDayNameWed.
SShortMonthNameApr (see page 3136)	This is constant SysConst.SShortMonthNameApr.
SShortMonthNameAug (see page 3136)	This is constant SysConst.SShortMonthNameAug.

SShortMonthNameDec (see page 3136)	This is constant SysConst.SShortMonthNameDec.
SShortMonthNameFeb (see page 3137)	This is constant SysConst.SShortMonthNameFeb.
SShortMonthNameJan (see page 3137)	This is constant SysConst.SShortMonthNameJan.
SShortMonthNameJul (see page 3137)	This is constant SysConst.SShortMonthNameJul.
SShortMonthNameJun (see page 3138)	This is constant SysConst.SShortMonthNameJun.
SShortMonthNameMar (see page 3138)	This is constant SysConst.SShortMonthNameMar.
SShortMonthNameMay (see page 3138)	This is constant SysConst.SShortMonthNameMay.
SShortMonthNameNov (see page 3139)	This is constant SysConst.SShortMonthNameNov.
SShortMonthNameOct (see page 3139)	This is constant SysConst.SShortMonthNameOct.
SShortMonthNameSep (see page 3139)	This is constant SysConst.SShortMonthNameSep.
SStackOverflow (see page 3140)	This is constant SysConst.SStackOverflow.
STimeEncodeError (see page 3140)	This is constant SysConst.STimeEncodeError.
STooManyOpenFiles (see page 3140)	This is constant SysConst.STooManyOpenFiles.
SUnderflow (see page 3141)	This is constant SysConst.SUnderflow.
SUnknown (see page 3141)	This is constant SysConst.SUnknown.
sUnknownFromType (see page 3141)	This is constant SysConst.sUnknownFromType.
sUnknownToType (see page 3142)	This is constant SysConst.sUnknownToType.
SUnkOSError (see page 3142)	This is constant SysConst.SUnkOSError.
SUnkWin32Error (see page 3142)	This is constant SysConst.SUnkWin32Error.
SVarArrayBounds (see page 3143)	This is constant SysConst.SVarArrayBounds.
SVarArrayCreate (see page 3143)	This is constant SysConst.SVarArrayCreate.
SVarArrayLocked (see page 3143)	This is constant SysConst.SVarArrayLocked.
SVarArrayWithHResult (see page 3144)	This is constant SysConst.SVarArrayWithHResult.
SVarBadType (see page 3144)	This is constant SysConst.SVarBadType.
SVarDataCastToRecursing (see page 3144)	This is constant SysConst.SVarDataCastToRecursing.
SVarDataClearRecursing (see page 3145)	This is constant SysConst.SVarDataClearRecursing.
SVarDataCopyNoIndRecursing (see page 3145)	This is constant SysConst.SVarDataCopyNoIndRecursing.
SVarDataCopyRecursing (see page 3145)	This is constant SysConst.SVarDataCopyRecursing.
SVarDataInitRecursing (see page 3146)	This is constant SysConst.SVarDataInitRecursing.
SVarInvalid (see page 3146)	This is constant SysConst.SVarInvalid.
SVarIsEmpty (see page 3146)	This is constant SysConst.SVarIsEmpty.
SVarNotArray (see page 3147)	This is constant SysConst.SVarNotArray.
SVarNotImplemented (see page 3147)	This is constant SysConst.SVarNotImplemented.
SVarOutOfMemory (see page 3147)	This is constant SysConst.SVarOutOfMemory.
SVarOverflow (see page 3148)	This is constant SysConst.SVarOverflow.
SVarTypeAlreadyUsed (see page 3148)	This is constant SysConst.SVarTypeAlreadyUsed.
SVarTypeAlreadyUsedWithPrefix (see page 3148)	This is constant SysConst.SVarTypeAlreadyUsedWithPrefix.
SVarTypeConvertOverflow (see page 3149)	This is constant SysConst.SVarTypeConvertOverflow.
SVarTypeCouldNotConvert (see page 3149)	This is constant SysConst.SVarTypeCouldNotConvert.
SVarTypeNotUsable (see page 3149)	This is constant SysConst.SVarTypeNotUsable.
SVarTypeNotUsableWithPrefix (see page 3150)	This is constant SysConst.SVarTypeNotUsableWithPrefix.
SVarTypeOutOfRange (see page 3150)	This is constant SysConst.SVarTypeOutOfRange.
SVarTypeOutOfRangeWithPrefix (see page 3150)	This is constant SysConst.SVarTypeOutOfRangeWithPrefix.
SVarTypeRangeCheck1 (see page 3151)	This is constant SysConst.SVarTypeRangeCheck1.
SVarTypeRangeCheck2 (see page 3151)	This is constant SysConst.SVarTypeRangeCheck2.
SVarTypeTooManyCustom (see page 3151)	This is constant SysConst.SVarTypeTooManyCustom.
SVarTypeUnknown (see page 3152)	This is constant SysConst.SVarTypeUnknown.
SVarUnexpected (see page 3152)	This is constant SysConst.SVarUnexpected.
SWin32Error (see page 3152)	This is constant SysConst.SWin32Error.
SWriteAccess (see page 3153)	This is constant SysConst.SWriteAccess.
SZeroDivide (see page 3153)	This is constant SysConst.SZeroDivide.

36.1.1 SysConst.SAbstractError Constant

Pascal

```
SAbstractError = 'Abstract Error';
```

C++

```
SAbstractError = 'Abstract Error';
```

File

SysConst

Description

This is constant SysConst.SAbstractError.

36.1.2 SysConst.SAccessDenied Constant

Pascal

```
SAccessDenied = 'File access denied';
```

C++

```
SAccessDenied = 'File access denied';
```

File

SysConst

Description

This is constant SysConst.SAccessDenied.

36.1.3 SysConst.SAccessViolationArg3 Constant

Pascal

```
SAccessViolationArg3 = 'Access violation at address %p. %s of address %p';
```

C++

```
SAccessViolationArg3 = 'Access violation at address %p. %s of address %p';
```

File

SysConst

Description

This is constant SysConst.SAccessViolationArg3.

36.1.4 SysConst.SAccessViolationNoArg Constant

Pascal

```
SAccessViolationNoArg = 'Access violation';
```

C++

```
SAccessViolationNoArg = 'Access violation';
```

File

SysConst

Description

This is constant SysConst.SAccessViolationNoArg.

36.1.5 SysConst.SArgumentMissing Constant

Pascal

```
SArgumentMissing = 'No argument for format ''%s''';
```

C++

```
SArgumentMissing = 'No argument for format ''%s''';
```

File

SysConst

Description

This is constant SysConst.SArgumentMissing.

36.1.6 SysConst.SAssertError Constant

Pascal

```
SAssertError = '%s (%s, line %d)';
```

C++

```
SAssertError = '%s (%s, line %d)';
```

File

SysConst

Description

This is constant SysConst.SAssertError.

36.1.7 SysConst.SAssertionFailed Constant

Pascal

```
SAssertionFailed = 'Assertion failed';
```

C++

```
SAssertionFailed = 'Assertion failed';
```

File

SysConst

Description

This is constant SysConst.SAssertionFailed.

36.1.8 SysConst.SByteIndexOutOfBounds Constant

Pascal

```
SByteIndexOutOfBounds = 'Start index out of bounds (%d)';
```

C++

```
SByteIndexOutOfBounds = 'Start index out of bounds (%d)';
```

File

SysConst

Description

This is constant SysConst.SByteIndexOutOfBounds.

36.1.9 SysConst.SCannotCreateDir Constant

Pascal

```
SCannotCreateDir = 'Unable to create directory';
```

C++

```
SCannotCreateDir = 'Unable to create directory';
```

File

SysConst

Description

This is constant SysConst.SCannotCreateDir.

36.1.10 SysConst.SCannotReadPackageInfo Constant

Pascal

```
SCannotReadPackageInfo = 'Cannot access package information for package ''%s''';
```

C++

```
SCannotReadPackageInfo = 'Cannot access package information for package ''%s''';
```

File

SysConst

Description

This is constant SysConst.SCannotReadPackageInfo.

36.1.11 SysConst.SCharIndexOutOfBounds Constant

Pascal

```
SCharIndexOutOfBounds = 'Character index out of bounds (%d)';
```

C++

```
SCharIndexOutOfBounds = 'Character index out of bounds (%d)';
```

File

SysConst

Description

This is constant SysConst.SCharIndexOutOfBounds.

36.1.12 SysConst.SCodesetConversionError Constant

Pascal

```
SCodesetConversionError = 'Codeset conversion failure';
```

C++

```
SCodesetConversionError = 'Codeset conversion failure';
```

File

SysConst

Description

This is constant SysConst.SCodesetConversionError.

36.1.13 SysConst.SControlC Constant

Pascal

```
SControlC = 'Control-C hit';
```

C++

```
SControlC = 'Control-C hit';
```

File

SysConst

Description

This is constant SysConst.SControlC.

36.1.14 SysConst.SDateEncodeError Constant

Pascal

```
SDateEncodeError = 'Invalid argument to date encode';
```

C++

```
SDateEncodeError = 'Invalid argument to date encode';
```

File

SysConst

Description

This is constant SysConst.SDateEncodeError.

36.1.15 SysConst.SDiskFull Constant

Pascal

```
SDiskFull = 'Disk full';
```

C++

```
SDiskFull = 'Disk full';
```

File

SysConst

Description

This is constant SysConst.SDiskFull.

36.1.16 SysConst.SDispatchError Constant

Pascal

```
SDispatchError = 'Variant method calls not supported';
```

C++

```
SDispatchError = 'Variant method calls not supported';
```

File

SysConst

Description

This is constant SysConst.SDispatchError.

36.1.17 SysConst.SDivByZero Constant

Pascal

```
SDivByZero = 'Division by zero';
```

C++

```
SDivByZero = 'Division by zero';
```

File

SysConst

Description

This is constant SysConst.SDivByZero.

36.1.18 SysConst.SDuplicatePackageUnit Constant

Pascal

```
SDuplicatePackageUnit = 'Cannot load package ''%s.''  It contains unit ''%s,''' + 'which is
also contained in package ''%s''';
```

C++

```
SDuplicatePackageUnit = 'Cannot load package ''%s.''  It contains unit ''%s,''' + 'which is
also contained in package ''%s''';
```

File

SysConst

Description

This is constant SysConst.SDuplicatePackageUnit.

36.1.19 SysConst.SEndOfFile Constant

Pascal

```
SEndOfFile = 'Read beyond end of file';
```

C++

```
SEndOfFile = 'Read beyond end of file';
```

File

SysConst

Description

This is constant SysConst.SEndOfFile.

36.1.20 SysConst.sErrorLoadingPackage Constant

Pascal

```
sErrorLoadingPackage = 'Can''t load package %s.'+sLineBreak+'%s';
```

C++

```
sErrorLoadingPackage = 'Can''t load package %s.'+sLineBreak+'%s';
```

File

SysConst

Description

This is constant SysConst.sErrorLoadingPackage.

36.1.21 SysConst.SException Constant

Pascal

```
SException = 'Exception %s in module %s at %p.' + sLineBreak + '%s%s' + sLineBreak;
```

C++

```
SException = 'Exception %s in module %s at %p.' + sLineBreak + '%s%s' + sLineBreak;
```

File

SysConst

Description

This is constant SysConst.SException.

36.1.22 SysConst.SExceptTitle Constant

Pascal

```
SExceptTitle = 'Application Error';
```

C++

```
SExceptTitle = 'Application Error';
```

File

SysConst

Description

This is constant SysConst.SExceptTitle.

36.1.23 SysConst.SExternalException Constant

Pascal

```
SExternalException = 'External exception %x';
```

C++

```
SExternalException = 'External exception %x';
```

File

SysConst

Description

This is constant SysConst.SExternalException.

36.1.24 SysConst.SFileNotFound Constant

Pascal

```
SFileNotFound = 'File not found';
```

C++

```
SFileNotFound = 'File not found';
```

File

SysConst

Description

This is constant SysConst.SFileNotFound.

36.1.25 SysConst.SFormatTooLong Constant

Pascal

```
SFormatTooLong = 'Format string too long';
```

C++

```
SFormatTooLong = 'Format string too long';
```

File

SysConst

Description

This is constant SysConst.SFormatTooLong.

36.1.26 SysConst.SInOutError Constant

Pascal

```
SInOutError = 'I/O error %d';
```

C++

```
SInOutError = 'I/O error %d';
```

File

SysConst

Description

This is constant SysConst.SInOutError.

36.1.27 SysConst.SIntfCastError Constant

Pascal

```
SIntfCastError = 'Interface not supported';
```

C++

```
SIntfCastError = 'Interface not supported';
```

File

SysConst

Description

This is constant SysConst.SIntfCastError.

36.1.28 SysConst.SIntOverflow Constant

Pascal

```
SIntOverflow = 'Integer overflow';
```

C++

```
SIntOverflow = 'Integer overflow';
```

File

SysConst

Description

This is constant SysConst.SIntOverflow.

36.1.29 SysConst.SInvalidBoolean Constant

Pascal

```
SInvalidBoolean = '''%s'' is not a valid boolean value';
```

C++

```
SInvalidBoolean = '''%s'' is not a valid boolean value';
```

File

SysConst

Description

This is constant SysConst.SInvalidBoolean.

36.1.30 SysConst.SInvalidCast Constant

Pascal

```
SInvalidCast = 'Invalid class typecast';
```

C++

```
SInvalidCast = 'Invalid class typecast';
```

File

SysConst

Description

This is constant SysConst.SInvalidCast.

36.1.31 SysConst.SInvalidCharCount Constant

Pascal

```
SInvalidCharCount = 'Invalid count (%d)';
```

C++

```
SInvalidCharCount = 'Invalid count (%d)';
```

File

SysConst

Description

This is constant SysConst.SInvalidCharCount.

36.1.32 SysConst.SInvalidCodePage Constant

Pascal

```
SInvalidCodePage = 'Invalid code page';
```

C++

```
SInvalidCodePage = 'Invalid code page';
```

File

SysConst

Description

This is constant SysConst.SInvalidCodePage.

36.1.33 SysConst.SInvalidCurrency Constant

Pascal

```
SInvalidCurrency = '''%s'' is not a valid currency value';
```

C++

```
SInvalidCurrency = '''%s'' is not a valid currency value';
```

File

SysConst

Description

This is constant SysConst.SInvalidCurrency.

36.1.34 SysConst.SInvalidDate Constant

Pascal

```
SInvalidDate = '''%s'' is not a valid date';
```

C++

```
SInvalidDate = '''%s'' is not a valid date';
```

File

SysConst

Description

This is constant SysConst.SInvalidDate.

36.1.35 SysConst.SInvalidDateTime Constant

Pascal

```
SInvalidDateTime = '''%s'' is not a valid date and time';
```

C++

```
SInvalidDateTime = '''%s'' is not a valid date and time';
```

File

SysConst

Description

This is constant SysConst.SInvalidDateTime.

36.1.36 SysConst.SInvalidDateTimeFloat Constant

Pascal

```
SInvalidDateTimeFloat = '''%g'' is not a valid date and time';
```

C++

```
SInvalidDateTimeFloat = '''%g'' is not a valid date and time';
```

File

SysConst

Description

This is constant SysConst.SInvalidDateTimeFloat.

36.1.37 SysConst.SInvalidDestinationArray Constant

Pascal

```
SInvalidDestinationArray = 'Invalid destination array';
```

C++

```
SInvalidDestinationArray = 'Invalid destination array';
```

File

SysConst

Description

This is constant SysConst.SInvalidDestinationArray.

36.1.38 SysConst.SInvalidDestinationIndex Constant

Pascal

```
SInvalidDestinationIndex = 'Invalid destination index (%d)';
```

C++

```
SInvalidDestinationIndex = 'Invalid destination index (%d)';
```

File

SysConst

Description

This is constant SysConst.SInvalidDestinationIndex.

36.1.39 SysConst.SInvalidFilename Constant

Pascal

```
SInvalidFilename = 'Invalid filename';
```

C++

```
SInvalidFilename = 'Invalid filename';
```

File

SysConst

Description

This is constant SysConst.SInvalidFilename.

36.1.40 SysConst.SInvalidFloat Constant

Pascal

```
SInvalidFloat = '''%s'' is not a valid floating point value';
```

C++

```
SInvalidFloat = '''%s'' is not a valid floating point value';
```

File

SysConst

Description

This is constant SysConst.SInvalidFloat.

36.1.41 SysConst.SInvalidFormat Constant

Pascal

```
SInvalidFormat = 'Format ''%s'' invalid or incompatible with argument';
```

C++

```
SInvalidFormat = 'Format ''%s'' invalid or incompatible with argument';
```

File

SysConst

Description

This is constant SysConst.SInvalidFormat.

36.1.42 SysConst.SInvalidGUID Constant

Pascal

```
SInvalidGUID = '''%s'' is not a valid GUID value';
```

C++

```
SInvalidGUID = '''%s'' is not a valid GUID value';
```

File

SysConst

Description

This is constant SysConst.SInvalidGUID.

36.1.43 SysConst.SInvalidInput Constant

Pascal

```
SInvalidInput = 'Invalid numeric input';
```

C++

```
SInvalidInput = 'Invalid numeric input';
```

File

SysConst

Description

This is constant SysConst.SInvalidInput.

36.1.44 SysConst.SInvalidInteger Constant

Pascal

```
SInvalidInteger = '''%s'' is not a valid integer value';
```

C++

```
SInvalidInteger = '''%s'' is not a valid integer value';
```

File

SysConst

Description

This is constant SysConst.SInvalidInteger.

36.1.45 SysConst.SInvalidOp Constant

Pascal

```
SInvalidOp = 'Invalid floating point operation';
```

C++

```
SInvalidOp = 'Invalid floating point operation';
```

File

SysConst

Description

This is constant SysConst.SInvalidOp.

36.1.46 SysConst.SInvalidPackageFile Constant

Pascal

```
SInvalidPackageFile = 'Invalid package file ''%s''';
```

C++

```
SInvalidPackageFile = 'Invalid package file ''%s''';
```

File

SysConst

Description

This is constant SysConst.SInvalidPackageFile.

36.1.47 SysConst.SInvalidPackageHandle Constant

Pascal

```
SInvalidPackageHandle = 'Invalid package handle';
```

C++

```
SInvalidPackageHandle = 'Invalid package handle';
```

File

SysConst

Description

This is constant SysConst.SInvalidPackageHandle.

36.1.48 SysConst.SInvalidPointer Constant

Pascal

```
SInvalidPointer = 'Invalid pointer operation';
```

C++

```
SInvalidPointer = 'Invalid pointer operation';
```

File

SysConst

Description

This is constant SysConst.SInvalidPointer.

36.1.49 SysConst.SInvalidSourceArray Constant

Pascal

```
SInvalidSourceArray = 'Invalid source array';
```

C++

```
SInvalidSourceArray = 'Invalid source array';
```

File

SysConst

Description

This is constant SysConst.SInvalidSourceArray.

36.1.50 SysConst.SInvalidTime Constant

Pascal

```
SInvalidTime = '''%s'' is not a valid time';
```

C++

```
SInvalidTime = '''%s'' is not a valid time';
```

File

SysConst

Description

This is constant SysConst.SInvalidTime.

36.1.51 SysConst.SInvalidTimeStamp Constant

Pascal

```
SInvalidTimeStamp = '''%d.%d'' is not a valid timestamp';
```

C++

```
SInvalidTimeStamp = '''%d.%d'' is not a valid timestamp';
```

File

SysConst

Description

This is constant SysConst.SInvalidTimeStamp.

36.1.52 SysConst.SInvalidVarCast Constant

Pascal

```
SInvalidVarCast = 'Invalid variant type conversion';
```

C++

```
SInvalidVarCast = 'Invalid variant type conversion';
```

File

SysConst

Description

This is constant SysConst.SInvalidVarCast.

36.1.53 SysConst.SInvalidVarNullOp Constant

Pascal

```
SInvalidVarNullOp = 'Invalid NULL variant operation';
```

C++

```
SInvalidVarNullOp = 'Invalid NULL variant operation';
```

File

SysConst

Description

This is constant SysConst.SInvalidVarNullOp.

36.1.54 SysConst.SInvalidVarOp Constant

Pascal

```
SInvalidVarOp = 'Invalid variant operation';
```

C++

```
SInvalidVarOp = 'Invalid variant operation';
```

File

SysConst

Description

This is constant SysConst.SInvalidVarOp.

36.1.55 SysConst.SInvalidVarOpWithHResult Constant

Pascal

```
SInvalidVarOpWithHResult = 'Invalid variant operation ($%.8x)' deprecated;
```

C++

```
SInvalidVarOpWithHResult = 'Invalid variant operation ($%.8x)' deprecated;
```

File

SysConst

Description

This is constant SysConst.SInvalidVarOpWithHResult.

36.1.56 SysConst.SInvalidVarOpWithHResultWithPrefix Constant

Pascal

```
SInvalidVarOpWithHResultWithPrefix = 'Invalid variant operation (%s%.8x)'#10'%s';
```

C++

```
SInvalidVarOpWithHResultWithPrefix = 'Invalid variant operation (%s%.8x)'#10'%s';
```

File

SysConst

Description

This is constant SysConst.SInvalidVarOpWithHResultWithPrefix.

36.1.57 SysConst.SLongDayNameFri Constant

Pascal

```
SLongDayNameFri = 'Friday';
```

C++

```
SLongDayNameFri = 'Friday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameFri.

36.1.58 SysConst.SLongDayNameMon Constant

Pascal

```
SLongDayNameMon = 'Monday';
```

C++

```
SLongDayNameMon = 'Monday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameMon.

36.1.59 SysConst.SLongDayNameSat Constant

Pascal

```
SLongDayNameSat = 'Saturday';
```

C++

```
SLongDayNameSat = 'Saturday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameSat.

36.1.60 SysConst.SLongDayNameSun Constant

Pascal

```
SLongDayNameSun = 'Sunday';
```

C++

```
SLongDayNameSun = 'Sunday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameSun.

36.1.61 SysConst.SLongDayNameThu Constant

Pascal

```
SLongDayNameThu = 'Thursday';
```

C++

```
SLongDayNameThu = 'Thursday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameThu.

36.1.62 SysConst.SLongDayNameTue Constant

Pascal

```
SLongDayNameTue = 'Tuesday';
```

C++

```
SLongDayNameTue = 'Tuesday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameTue.

36.1.63 SysConst.SLongDayNameWed Constant

Pascal

```
SLongDayNameWed = 'Wednesday';
```

C++

```
SLongDayNameWed = 'Wednesday';
```

File

SysConst

Description

This is constant SysConst.SLongDayNameWed.

36.1.64 SysConst.SLongMonthNameApr Constant

Pascal

```
SLongMonthNameApr = 'April';
```

C++

```
SLongMonthNameApr = 'April';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameApr.

36.1.65 SysConst.SLongMonthNameAug Constant

Pascal

```
SLongMonthNameAug = 'August';
```

C++

```
SLongMonthNameAug = 'August';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameAug.

36.1.66 SysConst.SLongMonthNameDec Constant

Pascal

```
SLongMonthNameDec = 'December';
```

C++

```
SLongMonthNameDec = 'December';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameDec.

36.1.67 SysConst.SLongMonthNameFeb Constant

Pascal

```
SLongMonthNameFeb = 'February';
```

C++

```
SLongMonthNameFeb = 'February';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameFeb.

36.1.68 SysConst.SLongMonthNameJan Constant

Pascal

```
SLongMonthNameJan = 'January';
```

C++

```
SLongMonthNameJan = 'January';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameJan.

36.1.69 SysConst.SLongMonthNameJul Constant

Pascal

```
SLongMonthNameJul = 'July';
```

C++

```
SLongMonthNameJul = 'July';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameJul.

36.1.70 SysConst.SLongMonthNameJun Constant

Pascal

```
SLongMonthNameJun = 'June';
```

C++

```
SLongMonthNameJun = 'June';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameJun.

36.1.71 SysConst.SLongMonthNameMar Constant

Pascal

```
SLongMonthNameMar = 'March';
```

C++

```
SLongMonthNameMar = 'March';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameMar.

36.1.72 SysConst.SLongMonthNameMay Constant

Pascal

```
SLongMonthNameMay = 'May';
```

C++

```
SLongMonthNameMay = 'May';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameMay.

36.1.73 SysConst.SLongMonthNameNov Constant

Pascal

```
SLongMonthNameNov = 'November';
```

C++

```
SLongMonthNameNov = 'November';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameNov.

36.1.74 SysConst.SLongMonthNameOct Constant

Pascal

```
SLongMonthNameOct = 'October';
```

C++

```
SLongMonthNameOct = 'October';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameOct.

36.1.75 SysConst.SLongMonthNameSep Constant

Pascal

```
SLongMonthNameSep = 'September';
```

C++

```
SLongMonthNameSep = 'September';
```

File

SysConst

Description

This is constant SysConst.SLongMonthNameSep.

36.1.76 SysConst.SModuleAccessViolation Constant

Pascal

```
SModuleAccessViolation = 'Access violation at address %p in module ''%s''. %s of address %p';
```

C++

```
SModuleAccessViolation = 'Access violation at address %p in module ''%s''. %s of address %p';
```

File

SysConst

Description

This is constant SysConst.SModuleAccessViolation.

36.1.77 SysConst.SMonitorLockException Constant

Pascal

```
SMonitorLockException = 'Object lock not owned';
```

C++

```
SMonitorLockException = 'Object lock not owned';
```

File

SysConst

Description

This is constant SysConst.SMonitorLockException.

36.1.78 SysConst.SNL Constant

Pascal

```
SNL = 'Application is not licensed to use this feature';
```

C++

```
SNL = 'Application is not licensed to use this feature';
```

File

SysConst

Description

This is constant SysConst.SNL.

36.1.79 SysConst.SNoMonitorSupportException Constant

Pascal

```
SNoMonitorSupportException = 'Monitor support function not initialized';
```

C++

```
SNoMonitorSupportException = 'Monitor support function not initialized';
```

File

SysConst

Description

This is constant SysConst.SNoMonitorSupportException.

36.1.80 SysConst.SOperationAborted Constant

Pascal

```
SOperationAborted = 'Operation aborted';
```

C++

```
SOperationAborted = 'Operation aborted';
```

File

SysConst

Description

This is constant SysConst.SOperationAborted.

36.1.81 SysConst.SOSError Constant

Pascal

```
SOSError = 'System Error.  Code: %d.'+sLineBreak+'%s';
```

C++

```
SOSError = 'System Error.  Code: %d.'+sLineBreak+'%s';
```

File

SysConst

Description

This is constant SysConst.SOSError.

36.1.82 SysConst.SOutOfMemory Constant

Pascal

```
SOutOfMemory = 'Out of memory';
```

C++

```
SOutOfMemory = 'Out of memory';
```

File

SysConst

Description

This is constant SysConst.SOutOfMemory.

36.1.83 SysConst.SOverflow Constant

Pascal

```
SOverflow = 'Floating point overflow';
```

C++

```
SOverflow = 'Floating point overflow';
```

File

SysConst

Description

This is constant SysConst.SOverflow.

36.1.84 SysConst.SPrivilege Constant

Pascal

```
SPrivilege = 'Privileged instruction';
```

C++

```
SPrivilege = 'Privileged instruction';
```

File

SysConst

Description

This is constant SysConst.SPrivilege.

36.1.85 SysConst.SQuit Constant

Pascal

```
SQuit = 'Quit key hit';
```

C++

```
SQuit = 'Quit key hit';
```

File

SysConst

Description

This is constant SysConst.SQuit.

36.1.86 SysConst.SRangeError Constant

Pascal

```
SRangeError = 'Range check error';
```

C++

```
SRangeError = 'Range check error';
```

File

SysConst

Description

This is constant SysConst.SRangeError.

36.1.87 SysConst.SReadAccess Constant

Pascal

```
SReadAccess = 'Read';
```

C++

```
SReadAccess = 'Read';
```

File

SysConst

Description

This is constant SysConst.SReadAccess.

36.1.88 SysConst.SResultTooLong Constant

Pascal

```
SResultTooLong = 'Format result longer than 4096 characters';
```

C++

```
SResultTooLong = 'Format result longer than 4096 characters';
```

File

SysConst

Description

This is constant SysConst.SResultTooLong.

36.1.89 SysConst.SSafecallException Constant

Pascal

```
SSafecallException = 'Exception in safecall method';
```

C++

```
SSafecallException = 'Exception in safecall method';
```

File

SysConst

Description

This is constant SysConst.SSafecallException.

36.1.90 SysConst.SShortDayNameFri Constant

Pascal

```
SShortDayNameFri = 'Fri';
```

C++

```
SShortDayNameFri = 'Fri';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameFri.

36.1.91 SysConst.SShortDayNameMon Constant

Pascal

```
SShortDayNameMon = 'Mon';
```

C++

```
SShortDayNameMon = 'Mon';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameMon.

36.1.92 SysConst.SShortDayNameSat Constant

Pascal

```
SShortDayNameSat = 'Sat';
```

C++

```
SShortDayNameSat = 'Sat';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameSat.

36.1.93 SysConst.SShortDayNameSun Constant

Pascal

```
SShortDayNameSun = 'Sun';
```

C++

```
SShortDayNameSun = 'Sun';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameSun.

36.1.94 SysConst.SShortDayNameThu Constant

Pascal

```
SShortDayNameThu = 'Thu';
```

C++

```
SShortDayNameThu = 'Thu';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameThu.

36.1.95 SysConst.SShortDayNameTue Constant

Pascal

```
SShortDayNameTue = 'Tue';
```

C++

```
SShortDayNameTue = 'Tue';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameTue.

36.1.96 SysConst.SShortDayNameWed Constant

Pascal

```
SShortDayNameWed = 'Wed';
```

C++

```
SShortDayNameWed = 'Wed';
```

File

SysConst

Description

This is constant SysConst.SShortDayNameWed.

36.1.97 SysConst.SShortMonthNameApr Constant

Pascal

```
SShortMonthNameApr = 'Apr';
```

C++

```
SShortMonthNameApr = 'Apr';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameApr.

36.1.98 SysConst.SShortMonthNameAug Constant

Pascal

```
SShortMonthNameAug = 'Aug';
```

C++

```
SShortMonthNameAug = 'Aug';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameAug.

36.1.99 SysConst.SShortMonthNameDec Constant

Pascal

```
SShortMonthNameDec = 'Dec';
```

C++

```
SShortMonthNameDec = 'Dec';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameDec.

36.1.100 SysConst.SShortMonthNameFeb Constant

Pascal

```
SShortMonthNameFeb = 'Feb';
```

C++

```
SShortMonthNameFeb = 'Feb';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameFeb.

36.1.101 SysConst.SShortMonthNameJan Constant

Pascal

```
SShortMonthNameJan = 'Jan';
```

C++

```
SShortMonthNameJan = 'Jan';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameJan.

36.1.102 SysConst.SShortMonthNameJul Constant

Pascal

```
SShortMonthNameJul = 'Jul';
```

C++

```
SShortMonthNameJul = 'Jul';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameJul.

36.1.103 SysConst.SShortMonthNameJun Constant

Pascal

```
SShortMonthNameJun = 'Jun';
```

C++

```
SShortMonthNameJun = 'Jun';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameJun.

36.1.104 SysConst.SShortMonthNameMar Constant

Pascal

```
SShortMonthNameMar = 'Mar';
```

C++

```
SShortMonthNameMar = 'Mar';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameMar.

36.1.105 SysConst.SShortMonthNameMay Constant

Pascal

```
SShortMonthNameMay = 'May';
```

C++

```
SShortMonthNameMay = 'May';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameMay.

36.1.106 SysConst.SShortMonthNameNov Constant

Pascal

```
SShortMonthNameNov = 'Nov';
```

C++

```
SShortMonthNameNov = 'Nov';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameNov.

36.1.107 SysConst.SShortMonthNameOct Constant

Pascal

```
SShortMonthNameOct = 'Oct';
```

C++

```
SShortMonthNameOct = 'Oct';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameOct.

36.1.108 SysConst.SShortMonthNameSep Constant

Pascal

```
SShortMonthNameSep = 'Sep';
```

C++

```
SShortMonthNameSep = 'Sep';
```

File

SysConst

Description

This is constant SysConst.SShortMonthNameSep.

36.1.109 SysConst.SStackOverflow Constant

Pascal

```
SStackOverflow = 'Stack overflow';
```

C++

```
SStackOverflow = 'Stack overflow';
```

File

SysConst

Description

This is constant SysConst.SStackOverflow.

36.1.110 SysConst.STimeEncodeError Constant

Pascal

```
STimeEncodeError = 'Invalid argument to time encode';
```

C++

```
STimeEncodeError = 'Invalid argument to time encode';
```

File

SysConst

Description

This is constant SysConst.STimeEncodeError.

36.1.111 SysConst.STooManyOpenFiles Constant

Pascal

```
STooManyOpenFiles = 'Too many open files';
```

C++

```
STooManyOpenFiles = 'Too many open files';
```

File

SysConst

Description

This is constant SysConst.STooManyOpenFiles.

36.1.112 SysConst.SUnderflow Constant

Pascal

```
SUnderflow = 'Floating point underflow';
```

C++

```
SUnderflow = 'Floating point underflow';
```

File

SysConst

Description

This is constant SysConst.SUnderflow.

36.1.113 SysConst.SUnknown Constant

Pascal

```
SUnknown = '<unknown>';
```

C++

```
SUnknown = '<unknown>';
```

File

SysConst

Description

This is constant SysConst.SUnknown.

36.1.114 SysConst.sUnknownFromType Constant

Pascal

```
sUnknownFromType = 'Cannot convert from the specified type';
```

C++

```
sUnknownFromType = 'Cannot convert from the specified type';
```

File

SysConst

Description

This is constant SysConst.sUnknownFromType.

36.1.115 SysConst.sUnknownToType Constant

Pascal

```
sUnknownToType = 'Cannot convert to the specified type';
```

C++

```
sUnknownToType = 'Cannot convert to the specified type';
```

File

SysConst

Description

This is constant SysConst.sUnknownToType.

36.1.116 SysConst.SUnkOSError Constant

Pascal

```
SUnkOSError = 'A call to an OS function failed';
```

C++

```
SUnkOSError = 'A call to an OS function failed';
```

File

SysConst

Description

This is constant SysConst.SUnkOSError.

36.1.117 SysConst.SUnkWin32Error Constant

Pascal

```
SUnkWin32Error = 'A Win32 API function failed' deprecated 'use SUnkOSError';
```

C++

```
SUnkWin32Error = 'A Win32 API function failed' deprecated 'use SUnkOSError';
```

File

SysConst

Description

This is constant SysConst.SUnkWin32Error.

36.1.118 SysConst.SVarArrayBounds Constant

Pascal

```
SVarArrayBounds = 'Variant or safe array index out of bounds';
```

C++

```
SVarArrayBounds = 'Variant or safe array index out of bounds';
```

File

SysConst

Description

This is constant SysConst.SVarArrayBounds.

36.1.119 SysConst.SVarArrayCreate Constant

Pascal

```
SVarArrayCreate = 'Error creating variant or safe array';
```

C++

```
SVarArrayCreate = 'Error creating variant or safe array';
```

File

SysConst

Description

This is constant SysConst.SVarArrayCreate.

36.1.120 SysConst.SVarArrayLocked Constant

Pascal

```
SVarArrayLocked = 'Variant or safe array is locked';
```

C++

```
SVarArrayLocked = 'Variant or safe array is locked';
```

File

SysConst

Description

This is constant SysConst.SVarArrayLocked.

36.1.121 SysConst.SVarArrayWithHResult Constant

Pascal

```
SVarArrayWithHResult = 'Unexpected variant or safe array error: %s%.8x';
```

C++

```
SVarArrayWithHResult = 'Unexpected variant or safe array error: %s%.8x';
```

File

SysConst

Description

This is constant SysConst.SVarArrayWithHResult.

36.1.122 SysConst.SVarBadType Constant

Pascal

```
SVarBadType = 'Invalid variant type';
```

C++

```
SVarBadType = 'Invalid variant type';
```

File

SysConst

Description

This is constant SysConst.SVarBadType.

36.1.123 SysConst.SVarDataCastToRecursing Constant

Pascal

```
SVarDataCastToRecursing = 'Recursion while doing a VarDataCastTo';
```

C++

```
SVarDataCastToRecursing = 'Recursion while doing a VarDataCastTo';
```

File

SysConst

Description

This is constant SysConst.SVarDataCastToRecursing.

36.1.124 SysConst.SVarDataClearRecursing Constant

Pascal

```
SVarDataClearRecursing = 'Recursion while doing a VarDataClear';
```

C++

```
SVarDataClearRecursing = 'Recursion while doing a VarDataClear';
```

File

SysConst

Description

This is constant SysConst.SVarDataClearRecursing.

36.1.125 SysConst.SVarDataCopyNoIndRecursing Constant

Pascal

```
SVarDataCopyNoIndRecursing = 'Recursion while doing a VarDataCopyNoInd';
```

C++

```
SVarDataCopyNoIndRecursing = 'Recursion while doing a VarDataCopyNoInd';
```

File

SysConst

Description

This is constant SysConst.SVarDataCopyNoIndRecursing.

36.1.126 SysConst.SVarDataCopyRecursing Constant

Pascal

```
SVarDataCopyRecursing = 'Recursion while doing a VarDataCopy';
```

C++

```
SVarDataCopyRecursing = 'Recursion while doing a VarDataCopy';
```

File

SysConst

Description

This is constant SysConst.SVarDataCopyRecursing.

36.1.127 SysConst.SVarDataInitRecursing Constant

Pascal

```
SVarDataInitRecursing = 'Recursion while doing a VarDataInit';
```

C++

```
SVarDataInitRecursing = 'Recursion while doing a VarDataInit';
```

File

SysConst

Description

This is constant SysConst.SVarDataInitRecursing.

36.1.128 SysConst.SVarInvalid Constant

Pascal

```
SVarInvalid = 'Invalid argument';
```

C++

```
SVarInvalid = 'Invalid argument';
```

File

SysConst

Description

This is constant SysConst.SVarInvalid.

36.1.129 SysConst.SVarIsEmpty Constant

Pascal

```
SVarIsEmpty = 'Variant is empty';
```

C++

```
SVarIsEmpty = 'Variant is empty';
```

File

SysConst

Description

This is constant SysConst.SVarIsEmpty.

36.1.130 SysConst.SVarNotArray Constant

Pascal

```
SVarNotArray = 'Variant is not an array' deprecated;
```

C++

```
SVarNotArray = 'Variant is not an array' deprecated;
```

File

SysConst

Description

This is constant SysConst.SVarNotArray.

36.1.131 SysConst.SVarNotImplemented Constant

Pascal

```
SVarNotImplemented = 'Operation not supported';
```

C++

```
SVarNotImplemented = 'Operation not supported';
```

File

SysConst

Description

This is constant SysConst.SVarNotImplemented.

36.1.132 SysConst.SVarOutOfMemory Constant

Pascal

```
SVarOutOfMemory = 'Variant operation ran out memory';
```

C++

```
SVarOutOfMemory = 'Variant operation ran out memory';
```

File

SysConst

Description

This is constant SysConst.SVarOutOfMemory.

36.1.133 SysConst.SVarOverflow Constant

Pascal

```
SVarOverflow = 'Variant overflow';
```

C++

```
SVarOverflow = 'Variant overflow';
```

File

SysConst

Description

This is constant SysConst.SVarOverflow.

36.1.134 SysConst.SVarTypeAlreadyUsed Constant

Pascal

```
SVarTypeAlreadyUsed = 'Custom variant type ($%.4x) already used by %s' deprecated;
```

C++

```
SVarTypeAlreadyUsed = 'Custom variant type ($%.4x) already used by %s' deprecated;
```

File

SysConst

Description

This is constant SysConst.SVarTypeAlreadyUsed.

36.1.135 SysConst.SVarTypeAlreadyUsedWithPrefix Constant

Pascal

```
SVarTypeAlreadyUsedWithPrefix = 'Custom variant type (%s%.4x) already used by %s';
```

C++

```
SVarTypeAlreadyUsedWithPrefix = 'Custom variant type (%s%.4x) already used by %s';
```

File

SysConst

Description

This is constant SysConst.SVarTypeAlreadyUsedWithPrefix.

36.1.136 SysConst.SVarTypeConvertOverflow Constant

Pascal

```
SVarTypeConvertOverflow = 'Overflow while converting variant of type (%s) into type (%s)';
```

C++

```
SVarTypeConvertOverflow = 'Overflow while converting variant of type (%s) into type (%s)';
```

File

SysConst

Description

This is constant SysConst.SVarTypeConvertOverflow.

36.1.137 SysConst.SVarTypeCouldNotConvert Constant

Pascal

```
SVarTypeCouldNotConvert = 'Could not convert variant of type (%s) into type (%s)';
```

C++

```
SVarTypeCouldNotConvert = 'Could not convert variant of type (%s) into type (%s)';
```

File

SysConst

Description

This is constant SysConst.SVarTypeCouldNotConvert.

36.1.138 SysConst.SVarTypeNotUsable Constant

Pascal

```
SVarTypeNotUsable = 'Custom variant type ($%.4x) is not usable' deprecated;
```

C++

```
SVarTypeNotUsable = 'Custom variant type ($%.4x) is not usable' deprecated;
```

File

SysConst

Description

This is constant SysConst.SVarTypeNotUsable.

36.1.139 SysConst.SVarTypeNotUsableWithPrefix Constant

Pascal

```
SVarTypeNotUsableWithPrefix = 'Custom variant type (%s%.4x) is not usable';
```

C++

```
SVarTypeNotUsableWithPrefix = 'Custom variant type (%s%.4x) is not usable';
```

File

SysConst

Description

This is constant SysConst.SVarTypeNotUsableWithPrefix.

36.1.140 SysConst.SVarTypeOutOfRange Constant

Pascal

```
SVarTypeOutOfRange = 'Custom variant type ($%.4x) is out of range' deprecated;
```

C++

```
SVarTypeOutOfRange = 'Custom variant type ($%.4x) is out of range' deprecated;
```

File

SysConst

Description

This is constant SysConst.SVarTypeOutOfRange.

36.1.141 SysConst.SVarTypeOutOfRangeWithPrefix Constant

Pascal

```
SVarTypeOutOfRangeWithPrefix = 'Custom variant type (%s%.4x) is out of range';
```

C++

```
SVarTypeOutOfRangeWithPrefix = 'Custom variant type (%s%.4x) is out of range';
```

File

SysConst

Description

This is constant SysConst.SVarTypeOutOfRangeWithPrefix.

36.1.142 SysConst.SVarTypeRangeCheck1 Constant

Pascal

```
SVarTypeRangeCheck1 = 'Range check error for variant of type (%s)';
```

C++

```
SVarTypeRangeCheck1 = 'Range check error for variant of type (%s)';
```

File

SysConst

Description

This is constant SysConst.SVarTypeRangeCheck1.

36.1.143 SysConst.SVarTypeRangeCheck2 Constant

Pascal

```
SVarTypeRangeCheck2 = 'Range check error while converting variant of type (%s) into type (%s)';
```

C++

```
SVarTypeRangeCheck2 = 'Range check error while converting variant of type (%s) into type (%s)';
```

File

SysConst

Description

This is constant SysConst.SVarTypeRangeCheck2.

36.1.144 SysConst.SVarTypeTooManyCustom Constant

Pascal

```
SVarTypeTooManyCustom = 'Too many custom variant types have been registered';
```

C++

```
SVarTypeTooManyCustom = 'Too many custom variant types have been registered';
```

File

SysConst

Description

This is constant SysConst.SVarTypeTooManyCustom.

36.1.145 SysConst.SVarTypeUnknown Constant

Pascal

```
SVarTypeUnknown = 'Unknown custom variant type ($%.4x)' deprecated;
```

C++

```
SVarTypeUnknown = 'Unknown custom variant type ($%.4x)' deprecated;
```

File

SysConst

Description

This is constant SysConst.SVarTypeUnknown.

36.1.146 SysConst.SVarUnexpected Constant

Pascal

```
SVarUnexpected = 'Unexpected variant error';
```

C++

```
SVarUnexpected = 'Unexpected variant error';
```

File

SysConst

Description

This is constant SysConst.SVarUnexpected.

36.1.147 SysConst.SWin32Error Constant

Pascal

```
SWin32Error = 'Win32 Error.  Code: %d.'#10'%s' deprecated 'Use SOSError';
```

C++

```
SWin32Error = 'Win32 Error.  Code: %d.'#10'%s' deprecated 'Use SOSError';
```

File

SysConst

Description

This is constant SysConst.SWin32Error.

36.1.148 SysConst.SWriteAccess Constant

Pascal

```
SWriteAccess = 'Write';
```

C++

```
SWriteAccess = 'Write';
```

File

SysConst

Description

This is constant SysConst.SWriteAccess.

36.1.149 SysConst.SZeroDivide Constant

Pascal

```
SZeroDivide = 'Floating point division by zero';
```

C++

```
SZeroDivide = 'Floating point division by zero';
```

File

SysConst

Description

This is constant SysConst.SZeroDivide.

37 SysInit Namespace

This is namespace SysInit.

Topics

Name	Description
Functions (see page 3156)	The following table lists functions in this documentation.
Variables (see page 3159)	The following table lists variables in this documentation.
Constants (see page 3163)	The following table lists constants in this documentation.

Constants

Name	Description
PtrToNil (see page 3163)	This is constant SysInit.PtrToNil.

Functions

	Name	Description
	_GetTls (see page 3156)	This is function SysInit._GetTls.
	_InitExe (see page 3156)	This is function SysInit._InitExe.
	_InitLib (see page 3157)	This is function SysInit._InitLib.
	_InitPkg (see page 3157)	This is function SysInit._InitPkg.
	_PackageLoad (see page 3157)	This is function SysInit._PackageLoad.
	_PackageUnload (see page 3158)	This is function SysInit._PackageUnload.
	VclExit (see page 3158)	This is function SysInit.VclExit.
	VclInit (see page 3158)	This is function SysInit.VclInit.

Variables

Name	Description
CoverageLibraryName (see page 3159)	This is variable SysInit.CoverageLibraryName.
DataMark (see page 3159)	This is variable SysInit.DataMark.
DllProc (see page 3160)	This is variable SysInit.DllProc.
DllProcEx (see page 3160)	This is variable SysInit.DllProcEx.
HInstance (see page 3160)	This is variable SysInit.HInstance.
ModuleIsCpp (see page 3161)	This is variable SysInit.ModuleIsCpp.
ModuleIsLib (see page 3161)	This is variable SysInit.ModuleIsLib.
ModuleIsPackage (see page 3161)	This is variable SysInit.ModuleIsPackage.
TlsIndex (see page 3162)	This is variable SysInit.TlsIndex.
TlsLast (see page 3162)	This is variable SysInit.TlsLast.

37.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	_GetTls (see page 3156)	This is function SysInit._GetTls.
	_InitExe (see page 3156)	This is function SysInit._InitExe.
	_InitLib (see page 3157)	This is function SysInit._InitLib.
	_InitPkg (see page 3157)	This is function SysInit._InitPkg.
	_PackageLoad (see page 3157)	This is function SysInit._PackageLoad.
	_PackageUnload (see page 3158)	This is function SysInit._PackageUnload.
	VclExit (see page 3158)	This is function SysInit.VclExit.
	VclInit (see page 3158)	This is function SysInit.VclInit.

37.1.1 SysInit._GetTls Function

Pascal

```
function _GetTls: Pointer;
```

C++

```
void * _GetTls();
```

File

SysInit

Description

This is function SysInit._GetTls.

37.1.2 SysInit._InitExe Function

Pascal

```
procedure _InitExe(InitTable: Pointer);
```

C++

```
_InitExe(void * InitTable);
```

File

SysInit

Description

This is function SysInit._InitExe.

37.1.3 SysInit._InitLib Function

Pascal

```
procedure _InitLib;
```

C++

```
_InitLib();
```

File

SysInit

Description

This is function SysInit._InitLib.

37.1.4 SysInit._InitPkg Function

Pascal

```
function _InitPkg(Hinst: Integer; Reason: Integer; Resvd: Pointer): LongBool; stdcall;
```

C++

```
__stdcall LongBool _InitPkg(int Hinst, int Reason, void * Resvd);
```

File

SysInit

Description

This is function SysInit._InitPkg.

37.1.5 SysInit._PackageLoad Function

Pascal

```
procedure _PackageLoad(const Table: PackageInfo);
```

C++

```
_PackageLoad(const PackageInfo Table);
```

File

SysInit

Description

This is function SysInit._PackageLoad.

37.1.6 SysInit._PackageUnload Function

Pascal

```
procedure _PackageUnload(const Table: PackageInfo);
```

C++

```
_PackageUnload(const PackageInfo Table);
```

File

SysInit

Description

This is function SysInit._PackageUnload.

37.1.7 SysInit.VclExit Function

Pascal

```
procedure VclExit; cdecl;
```

C++

```
__cdecl VclExit();
```

File

SysInit

Description

This is function SysInit.VclExit.

37.1.8 SysInit.VclInit Function

Pascal

```
procedure VclInit(isDLL: Boolean; isPkg: Boolean; hInst: LongInt; isGui: Boolean); cdecl;
```

C++

```
__cdecl VclInit(Boolean isDLL, Boolean isPkg, LongInt hInst, Boolean isGui);
```

File

SysInit

Description

This is function SysInit.VclInit.

37.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CoverageLibraryName (see page 3159)	This is variable SysInit.CoverageLibraryName.
DataMark (see page 3159)	This is variable SysInit.DataMark.
DllProc (see page 3160)	This is variable SysInit.DllProc.
DllProcEx (see page 3160)	This is variable SysInit.DllProcEx.
HInstance (see page 3160)	This is variable SysInit.HInstance.
ModuleIsCpp (see page 3161)	This is variable SysInit.ModuleIsCpp.
ModuleIsLib (see page 3161)	This is variable SysInit.ModuleIsLib.
ModuleIsPackage (see page 3161)	This is variable SysInit.ModuleIsPackage.
TlsIndex (see page 3162)	This is variable SysInit.TlsIndex.
TlsLast (see page 3162)	This is variable SysInit.TlsLast.

37.2.1 SysInit.CoverageLibraryName Variable

Pascal

```
CoverageLibraryName: array [0..128] of AnsiChar = '*';
```

C++

```
array [0..128] of AnsiChar CoverageLibraryName = '*';
```

File

SysInit

Description

This is variable SysInit.CoverageLibraryName.

37.2.2 SysInit.DataMark Variable

Pascal

```
DataMark: Integer = 0;
```

C++

```
int DataMark = 0;
```

File

SysInit

Description

This is variable SysInit.DataMark.

37.2.3 SysInit.DllProc Variable

Pascal

```
DllProc: TDLLProc;
```

C++

```
TDLLProc DllProc;
```

File

SysInit

Description

This is variable SysInit.DllProc.

37.2.4 SysInit.DllProcEx Variable

Pascal

```
DllProcEx: TDLLProcEx absolute DllProc;
```

C++

```
TDLLProcEx absolute DllProc DllProcEx;
```

File

SysInit

Description

This is variable SysInit.DllProcEx.

37.2.5 SysInit.HInstance Variable

Pascal

```
HInstance: LongWord;
```

C++

```
LongWord HInstance;
```

File

SysInit

Description

This is variable SysInit.HInstance.

37.2.6 SysInit.ModuleIsCpp Variable

Pascal

```
ModuleIsCpp: Boolean;
```

C++

```
Boolean ModuleIsCpp;
```

File

SysInit

Description

This is variable SysInit.ModuleIsCpp.

37.2.7 SysInit.ModuleIsLib Variable

Pascal

```
ModuleIsLib: Boolean;
```

C++

```
Boolean ModuleIsLib;
```

File

SysInit

Description

This is variable SysInit.ModuleIsLib.

37.2.8 SysInit.ModuleIsPackage Variable

Pascal

```
ModuleIsPackage: Boolean;
```

C++

```
Boolean ModuleIsPackage;
```

File

SysInit

Description

This is variable SysInit.ModuleIsPackage.

37.2.9 SysInit.TlsIndex Variable

Pascal

```
TlsIndex: Integer = -1;
```

C++

```
int TlsIndex = -1;
```

File

SysInit

Description

This is variable SysInit.TlsIndex.

37.2.10 SysInit.TlsLast Variable

Pascal

```
TlsLast: Byte;
```

C++

```
Byte TlsLast;
```

File

SysInit

Description

This is variable SysInit.TlsLast.

37.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
PtrToNil (see page 3163)	This is constant SysInit.PtrToNil.

37.3.1 SysInit.PtrToNil Constant

Pascal

```
PtrToNil: Pointer = nil;
```

C++

```
void * PtrToNil = nil;
```

File

SysInit

Description

This is constant SysInit.PtrToNil.

38 System Namespace

This is namespace System.

Topics

Name	Description
Classes (see page 3185)	The following table lists classes in this documentation.
Interfaces (see page 3497)	The following table lists interfaces in this documentation.
Functions (see page 3507)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3794)	The following table lists structs, records, enums in this documentation.
Types (see page 3822)	The following table lists types in this documentation.
Variables (see page 3874)	The following table lists variables in this documentation.
Constants (see page 3909)	The following table lists constants in this documentation.
Macros (see page 3960)	The following table lists macros in this documentation.

Classes

	Name	Description
	AnsiStringBase (see page 3185)	System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.
	AnsiStringT (see page 3219)	System::AnsiStringT is the C++ analog for the Delphi long string type.
	Currency (see page 3242)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 3262)	This is class System::CurrencyBase.
	DelphiInterface (see page 3263)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 3268)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	DynArrayException (see page 3279)	This is class System::DynArrayException.
	DynArrayNullData (see page 3279)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 3279)	This is class System::DynArrayOutOfRange.
	OleVariant (see page 3281)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	SmallString (see page 3322)	This is class System::SmallString.
	SmallStringBase (see page 3324)	This is class System::SmallStringBase.
	sPointer (see page 3326)	This is class System::sPointer.
	TAggregatedObject (see page 3328)	This is class System::TAggregatedObject.
	TClassHelperBase (see page 3331)	This is class System.TClassHelperBase.
	TContainedObject (see page 3333)	This is class System::TContainedObject.
	TDateTime (see page 3335)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.

	TDateTimeBase (see page 3354)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 3355)	This is class System::TInterfacedObject.
	TMetaClass (see page 3360)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TMonitor (see page 3361)	This is class System.TMonitor.
	TObject (see page 3364)	System::TObject is the ultimate ancestor of all objects and components.
	UnicodeString (see page 3393)	System::UnicodeString is the C++ analog for the Delphi UnicodeString type.
	Variant (see page 3430)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	WideString (see page 3476)	System::WideString represents strings of 16-bit characters.

Constants

Name	Description
CompilerVersion (see page 3911)	Designates the version number of the Delphi compiler.
CPUi386 (see page 3911)	This is constant System.CPUi386.
CPUi486 (see page 3912)	This is constant System.CPUi486.
CPUPentium (see page 3912)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 3912)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 3913)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 3913)	This is constant System.E_UNEXPECTED.
fmClosed (see page 3913)	This is constant System.fmClosed.
fmInOut (see page 3914)	This is constant System.fmInOut.
fmInput (see page 3914)	This is constant System.fmInput.
fmOutput (see page 3914)	This is constant System.fmOutput.
hfFieldSize (see page 3915)	This is constant System.hfFieldSize.
hfMonitorOffset (see page 3915)	This is constant System.hfMonitorOffset.
INFINITE (see page 3915)	This is constant System.INFINITE.
MaxInt (see page 3916)	The maximum value of the Integer data type.
MaxLongint (see page 3916)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 3916)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 3917)	opAdd defines an addition operation.
opAnd (see page 3917)	opAnd defines a logical and operation.
opCmpEQ (see page 3918)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 3918)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 3919)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 3919)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 3920)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 3920)	opCmpNE defines an inequality comparison operator.
opCompare (see page 3921)	opCompare defines a comparison operation.
opDivide (see page 3921)	opDivide defines a divide operation.
opIntDivide (see page 3922)	opIntDivide defines an integer divide operation.
opModulus (see page 3922)	opModulus defines a modulus operation.
opMultiply (see page 3923)	opMultiply defines a multiply operation.
opNegate (see page 3923)	opNegate defines a negate operation.
opNot (see page 3924)	opNot defines a binary negation operation.
opOr (see page 3924)	opOr defines a logical or operation.
opShiftLeft (see page 3925)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 3925)	opShiftRight defines a bitwise shift right operation.
opSubtract (see page 3926)	opSubtract defines a subtraction operation.
opXor (see page 3926)	opXor defines a logical exclusive or operation.
RTLVersion (see page 3927)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 3927)	This is constant System.S_FALSE.
S_OK (see page 3927)	This is constant System.S_OK.
sLineBreak (see page 3928)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 3928)	This is constant System.tfCRLF.

varAny (see page 3928)	Represents a Variant (see page 3430) that can hold any value.
varArray (see page 3929)	Represents a Variant (see page 3430) that holds dynamic array data.
varBoolean (see page 3930)	Represents a Variant (see page 3430) that holds WordBool data.
varByRef (see page 3930)	Represents a Variant (see page 3430) that points to another variable.
varByte (see page 3931)	Represents a Variant (see page 3430) that holds byte data.
varCurrency (see page 3931)	Represents a Variant (see page 3430) that holds a monetary floating point value.
varDate (see page 3932)	Represents a Variant (see page 3430) that holds a TDateTime (see page 3335) value.
varDispatch (see page 3932)	Represents a Variant (see page 3430) that has been assigned an IDispatch (see page 3498) value.
varDouble (see page 3933)	Represents a Variant (see page 3430) that holds a floating point value.
varEmpty (see page 3933)	Represents a Variant (see page 3430) that has not yet been assigned a value.
varError (see page 3934)	Represents a Variant (see page 3430) that holds a 32 bit error code.
varInt64 (see page 3934)	Represents a Variant (see page 3430) that holds Int64 data.
varInteger (see page 3935)	Represents a Variant (see page 3430) that holds Integer data.
varLongWord (see page 3935)	Represents a Variant (see page 3430) that holds LongWord data.
varNull (see page 3936)	Represents a Variant (see page 3430) that holds a null value.
varOleStr (see page 3936)	Represents a Variant (see page 3430) that has been assigned a COM interface type string value.
varShortInt (see page 3937)	Represents a Variant (see page 3430) that holds ShortInt data.
varSingle (see page 3937)	Represents a Variant (see page 3430) that holds byte data.
varSmallint (see page 3938)	This is constant System.varSmallint.
varStrArg (see page 3938)	Represents a Variant (see page 3430) that has holds a COM compatible string value.
varString (see page 3938)	Represents a Variant (see page 3430) that holds string data.
varTypeMask (see page 3939)	Defines a mask for filtering out the meta data part of a Variant (see page 3430) data type.
varUInt64 (see page 3940)	This is constant System.varUInt64.
varUnknown (see page 3940)	Represents a Variant (see page 3430) that holds an unknown value.
varUString (see page 3940)	This is constant System.varUString.
varVariant (see page 3941)	Represents a Variant (see page 3430) data type.
varWord (see page 3941)	Represents a Variant (see page 3430) that holds Word data.
vmtAddRef (see page 3942)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 3942)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 3942)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 3943)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 3943)	This is constant System.vmtClassName.
vmtCreateObject (see page 3943)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 3944)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 3944)	This is constant System.vmtDestroy.
vmtDispatch (see page 3944)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 3945)	This is constant System.vmtDynamicTable.
vmtEquals (see page 3945)	This is constant System.vmtEquals.
vmtFieldTable (see page 3945)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 3946)	This is constant System.vmtFreeInstance.
vmtGetHashCode (see page 3946)	This is constant System.vmtGetHashCode.
vmtInitTable (see page 3946)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 3947)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 3947)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 3947)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 3948)	This is constant System.vmtNewInstance.
vmtParent (see page 3948)	This is constant System.vmtParent.
vmtQueryInterface (see page 3948)	This is constant System.vmtQueryInterface.
vmtRelease (see page 3949)	This is constant System.vmtRelease.
vmtSafeCallException (see page 3949)	This is constant System.vmtSafeCallException.

vmtSelfPtr (see page 3949)	This is constant System.vmtSelfPtr.
vmtToString (see page 3950)	This is constant System.vmtToString.
vmtTypeInfo (see page 3950)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 3950)	vtAnsiString defines an AnsiString TVarData (see page 3813) VType value.
vtBoolean (see page 3951)	vtBoolean defines a Boolean TVarData (see page 3813) VType value.
vtChar (see page 3951)	vtChar defines a Char TVarData (see page 3813) VType value.
vtClass (see page 3952)	vtClass defines a TClass (see page 3856) TVarData (see page 3813) VType value.
vtCurrency (see page 3952)	vtCurrency defines a Currency (see page 3242) TVarData (see page 3813) VType value.
vtExtended (see page 3953)	vtExtended defines an Extended TVarData (see page 3813) VType value.
vtInt64 (see page 3953)	vtInt64 defines an Int64 TVarData (see page 3813) VType value.
vtInteger (see page 3954)	vtInteger defines an Integer TVarData (see page 3813) VType value.
vtInterface (see page 3954)	vtInterface defines an Interface pointer TVarData (see page 3813) VType value.
vtObject (see page 3955)	vtObject defines a TObject (see page 3364) TVarData (see page 3813) VType value.
vtPChar (see page 3955)	vtPChar defines a PChar TVarData (see page 3813) VType value.
vtPointer (see page 3956)	vtPointer defines a Pointer TVarData (see page 3813) VType value.
vtPWideChar (see page 3956)	vtPWideChar defines a PWideChar TVarData (see page 3813) VType value.
vtString (see page 3957)	vtString defines a Variant (see page 3430) TVarData (see page 3813) VType value.
vtUnicodeString (see page 3957)	This is constant System.vtUnicodeString.
vtVariant (see page 3957)	This is constant System.vtVariant.
vtWideChar (see page 3958)	vtWideChar defines a WideChar TVarData (see page 3813) VType value.
vtWideString (see page 3958)	vtWideString defines a WideString (see page 3476) TVarData (see page 3813) VType value.

Functions

	Name	Description
	__ArrayToPasStr (see page 3518)	This is function System.__ArrayToPasStr.
	__CLenToPasStr (see page 3518)	This is function System.__CLenToPasStr.
	__CToPasStr (see page 3518)	This is function System.__CToPasStr.
	__currMul (see page 3519)	This is function System::__currMul.
	__IOTest (see page 3519)	This is function System.__IOTest.
	__lldiv (see page 3519)	This is function System.__lldiv.
	__lldivo (see page 3520)	This is function System.__lldivo.
	__llmod (see page 3520)	This is function System.__llmod.
	__llmodo (see page 3520)	This is function System.__llmodo.
	__llmul (see page 3521)	This is function System.__llmul.
	__llmulo (see page 3521)	This is function System.__llmulo.
	__llshl (see page 3521)	This is function System.__llshl.
	__ludiv (see page 3522)	This is function System.__ludiv.
	__lumod (see page 3522)	This is function System.__lumod.
	__lushr (see page 3522)	This is function System.__lushr.
	__PasToCStr (see page 3523)	This is function System.__PasToCStr.
	_AbstractError (see page 3523)	This is function System._AbstractError.
	_AddRef (see page 3523)	This is function System._AddRef.
	_AddRefArray (see page 3524)	This is function System._AddRefArray.
	_AddRefRecord (see page 3524)	This is function System._AddRefRecord.
	_AfterConstruction (see page 3524)	This is function System._AfterConstruction.
	_Append (see page 3525)	This is function System._Append.
	_AsClass (see page 3525)	This is function System._AsClass.
	_Assert (see page 3525)	This is function System._Assert.
	_Assign (see page 3526)	This is function System._Assign.
	_AStrCmp (see page 3526)	This is function System._AStrCmp.

	_BeforeDestruction (see page 3526)	This is function System._BeforeDestruction.
	_BlockRead (see page 3527)	This is function System._BlockRead.
	_BlockWrite (see page 3527)	This is function System._BlockWrite.
	_BoundErr (see page 3527)	This is function System._BoundErr.
	_CallDynaClass (see page 3528)	This is function System._CallDynaClass.
	_CallDynaInst (see page 3528)	This is function System._CallDynaInst.
	_CheckAutoResult (see page 3528)	This is function System._CheckAutoResult.
	_ClassCreate (see page 3529)	This is function System._ClassCreate.
	_ClassDestroy (see page 3529)	This is function System._ClassDestroy.
	_Close (see page 3529)	This is function System._Close.
	_Copy (see page 3530)	This is function System._Copy.
	_CopyArray (see page 3530)	This is function System._CopyArray.
	_CopyObject (see page 3530)	This is function System._CopyObject.
	_CopyRecord (see page 3531)	This is function System._CopyRecord.
	_CVR_PROBE (see page 3531)	This is function System._CVR_PROBE.
	_CVR_STMTPROBE (see page 3531)	This is function System._CVR_STMTPROBE.
	_Delete (see page 3532)	This is function System._Delete.
	_Dispose (see page 3532)	This is function System._Dispose.
	_DoneExcept (see page 3532)	This is function System._DoneExcept.
	_DynArrayAddRef (see page 3533)	This is function System._DynArrayAddRef.
	_DynArrayAsg (see page 3533)	This is function System._DynArrayAsg.
	_DynArrayClear (see page 3533)	This is function System._DynArrayClear.
	_DynArrayCopy (see page 3534)	This is function System._DynArrayCopy.
	_DynArrayCopyRange (see page 3534)	This is function System._DynArrayCopyRange.
	_DynArrayHigh (see page 3534)	This is function System._DynArrayHigh.
	_DynArrayLength (see page 3535)	This is function System._DynArrayLength.
	_DynArraySetLength (see page 3535)	This is function System._DynArraySetLength.
	_EnsureAnsiString (see page 3535)	This is function System._EnsureAnsiString.
	_EnsureUnicodeString (see page 3536)	This is function System._EnsureUnicodeString.
	_EofFile (see page 3536)	This is function System._EofFile.
	_EofText (see page 3536)	This is function System._EofText.
	_Eoln (see page 3537)	This is function System._Eoln.
	_Erase (see page 3537)	This is function System._Erase.
	_Ext2Real (see page 3537)	This is function System._Ext2Real.
	_Fail (see page 3538)	This is function System._Fail.
	_FilePos (see page 3538)	This is function System._FilePos.
	_FileSize (see page 3538)	This is function System._FileSize.
	_FillChar (see page 3539)	This is function System._FillChar.
	_Finalize (see page 3539)	This is function System._Finalize.
	_FinalizeArray (see page 3539)	This is function System._FinalizeArray.
	_FinalizeRecord (see page 3540)	This is function System._FinalizeRecord.
	_FindDynaClass (see page 3540)	This is function System._FindDynaClass.
	_FindDynaInst (see page 3540)	This is function System._FindDynaInst.
	_Flush (see page 3541)	This is function System._Flush.
	_FreeMem (see page 3541)	This is function System._FreeMem.
	_FSafeDivide (see page 3541)	This is function System._FSafeDivide.
	_FSafeDivideR (see page 3542)	This is function System._FSafeDivideR.
	_GetHelperIntf (see page 3542)	This is function System._GetHelperIntf.
	_GetMem (see page 3542)	This is function System._GetMem.
	_Halt (see page 3543)	This is function System._Halt.
	_Halt0 (see page 3543)	This is function System._Halt0.
	_HandleAnyException (see page 3543)	This is function System._HandleAnyException.
	_HandleAutoException (see page 3544)	This is function System._HandleAutoException.
	_HandleFinally (see page 3544)	This is function System._HandleFinally.

	_HandleOnException (see page 3544)	This is function System._HandleOnException.
	_Initialize (see page 3545)	This is function System._Initialize.
	_InitializeArray (see page 3545)	This is function System._InitializeArray.
	_InitializeRecord (see page 3545)	This is function System._InitializeRecord.
	_InitImports (see page 3546)	This is function System._InitImports.
	_InitResStringImports (see page 3546)	This is function System._InitResStringImports.
	_InitResStrings (see page 3546)	This is function System._InitResStrings.
	_InitUnicodeStrings (see page 3547)	This is function System._InitUnicodeStrings.
	_InitWideStrings (see page 3547)	This is function System._InitWideStrings.
	_Insert (see page 3547)	This is function System._Insert.
	_InternalLStrFromUStr (see page 3548)	This is function System._InternalLStrFromUStr.
	_InternalUStrFromLStr (see page 3548)	This is function System._InternalUStrFromLStr.
	_IntfAddRef (see page 3548)	This is function System._IntfAddRef.
	_IntfCast (see page 3549)	This is function System._IntfCast.
	_IntfClear (see page 3549)	This is function System._IntfClear.
	_IntfCopy (see page 3549)	This is function System._IntfCopy.
	_IntfDispCall (see page 3550)	This is function System._IntfDispCall.
	_IntfVarCall (see page 3550)	This is function System._IntfVarCall.
	_IntOver (see page 3550)	This is function System._IntOver.
	_IsClass (see page 3551)	This is function System._IsClass.
	_LGetDir (see page 3551)	This is function System._LGetDir.
	_LStrAddRef (see page 3551)	This is function System._LStrAddRef.
	_LStrArrayClr (see page 3552)	This is function System._LStrArrayClr.
	_LStrAsg (see page 3552)	This is function System._LStrAsg.
	_LStrCat (see page 3552)	This is function System._LStrCat.
	_LStrCat3 (see page 3553)	This is function System._LStrCat3.
	_LStrCatN (see page 3553)	This is function System._LStrCatN.
	_LStrClr (see page 3553)	This is function System._LStrClr.
	_LStrCmp (see page 3554)	This is function System._LStrCmp.
	_LStrCopy (see page 3554)	This is function System._LStrCopy.
	_LStrDelete (see page 3554)	This is function System._LStrDelete.
	_LStrEqual (see page 3555)	This is function System._LStrEqual.
	_LStrFromArray (see page 3555)	This is function System._LStrFromArray.
	_LStrFromChar (see page 3555)	This is function System._LStrFromChar.
	_LStrFromPChar (see page 3556)	This is function System._LStrFromPChar.
	_LStrFromPCharLen (see page 3556)	This is function System._LStrFromPCharLen.
	_LStrFromPWChar (see page 3556)	This is function System._LStrFromPWChar.
	_LStrFromPWCharLen (see page 3557)	This is function System._LStrFromPWCharLen.
	_LStrFromString (see page 3557)	This is function System._LStrFromString.
	_LStrFromUStr (see page 3557)	This is function System._LStrFromUStr.
	_LStrFromWArray (see page 3558)	This is function System._LStrFromWArray.
	_LStrFromWChar (see page 3558)	This is function System._LStrFromWChar.
	_LStrFromWStr (see page 3558)	This is function System._LStrFromWStr.
	_LStrInsert (see page 3559)	This is function System._LStrInsert.
	_LStrLAsg (see page 3559)	This is function System._LStrLAsg.
	_LStrLen (see page 3559)	This is function System._LStrLen.
	_LStrPos (see page 3560)	This is function System._LStrPos.
	_LStrSetLength (see page 3560)	This is function System._LStrSetLength.
	_LStrToPChar (see page 3560)	This is function System._LStrToPChar.
	_LStrToString (see page 3561)	This is function System._LStrToString.
	_New (see page 3561)	This is function System._New.
	_NewAnsiString (see page 3561)	This is function System._NewAnsiString.
	_NewUnicodeString (see page 3562)	This is function System._NewUnicodeString.
	_NewWideString (see page 3562)	This is function System._NewWideString.

	_ObjCopy (see page 3562)	This is function System._ObjCopy.
	_ObjSetup (see page 3563)	This is function System._ObjSetup.
	_PackageLoad (see page 3563)	This is function System._PackageLoad.
	_PackageUnload (see page 3563)	This is function System._PackageUnload.
	_PCharLen (see page 3564)	This is function System._PCharLen.
	_Pos (see page 3564)	This is function System._Pos.
	_Pow10 (see page 3564)	This is function System._Pow10.
	_PStrCat (see page 3565)	This is function System._PStrCat.
	_PStrCmp (see page 3565)	This is function System._PStrCmp.
	_PStrCpy (see page 3565)	This is function System._PStrCpy.
	_PStrNCat (see page 3566)	This is function System._PStrNCat.
	_PStrNCpy (see page 3566)	This is function System._PStrNCpy.
	_PWCharLen (see page 3566)	This is function System._PWCharLen.
	_RaiseAgain (see page 3567)	This is function System._RaiseAgain.
	_RaiseExcept (see page 3567)	This is function System._RaiseExcept.
	_ReadChar (see page 3567)	This is function System._ReadChar.
	_ReadCString (see page 3568)	This is function System._ReadCString.
	_ReadExt (see page 3568)	This is function System._ReadExt.
	_ReadInt64 (see page 3568)	This is function System._ReadInt64.
	_ReadLn (see page 3569)	This is function System._ReadLn.
	_ReadLong (see page 3569)	This is function System._ReadLong.
	_ReadLString (see page 3569)	This is function System._ReadLString.
	_ReadRec (see page 3570)	This is function System._ReadRec.
	_ReadString (see page 3570)	This is function System._ReadString.
	_ReadUString (see page 3570)	This is function System._ReadUString.
	_ReadWChar (see page 3571)	This is function System._ReadWChar.
	_ReadWCString (see page 3571)	This is function System._ReadWCString.
	_ReadWString (see page 3571)	This is function System._ReadWString.
	_Real2Ext (see page 3572)	This is function System._Real2Ext.
	_ReallocMem (see page 3572)	This is function System._ReallocMem.
	_Rename (see page 3572)	This is function System._Rename.
	_ResetFile (see page 3573)	This is function System._ResetFile.
	_ResetText (see page 3573)	This is function System._ResetText.
	_RewritFile (see page 3573)	This is function System._RewritFile.
	_RewritText (see page 3574)	This is function System._RewritText.
	_ROUND (see page 3574)	This is function System._ROUND.
	_roundToInt64 (see page 3574)	This is function System::_roundToInt64.
	_Run0Error (see page 3575)	This is function System._Run0Error.
	_RunError (see page 3575)	This is function System._RunError.
	_Seek (see page 3575)	This is function System._Seek.
	_SeekEof (see page 3576)	This is function System._SeekEof.
	_SeekEoln (see page 3576)	This is function System._SeekEoln.
	_SetElem (see page 3576)	This is function System._SetElem.
	_SetEq (see page 3577)	This is function System._SetEq.
	_SetExpand (see page 3577)	This is function System._SetExpand.
	_SetIntersect (see page 3577)	This is function System._SetIntersect.
	_SetIntersect3 (see page 3578)	This is function System._SetIntersect3.
	_SetLe (see page 3578)	This is function System._SetLe.
	_SetLength (see page 3578)	This is function System._SetLength.
	_SetRange (see page 3579)	This is function System._SetRange.
	_SetString (see page 3579)	This is function System._SetString.
	_SetSub (see page 3579)	This is function System._SetSub.
	_SetSub3 (see page 3580)	This is function System._SetSub3.
	_SetTextBuf (see page 3580)	This is function System._SetTextBuf.

	_SetUnion (see page 3580)	This is function System._SetUnion.
	_SetUnion3 (see page 3581)	This is function System._SetUnion3.
	_SGetDir (see page 3581)	This is function System._SGetDir.
	_StartExe (see page 3581)	This is function System._StartExe.
	_StartLib (see page 3582)	This is function System._StartLib.
	_Str0Ext (see page 3582)	This is function System._Str0Ext.
	_Str0Int64 (see page 3582)	This is function System._Str0Int64.
	_Str0Long (see page 3583)	This is function System._Str0Long.
	_Str0UInt64 (see page 3583)	This is function System._Str0UInt64.
	_Str1Ext (see page 3583)	This is function System._Str1Ext.
	_Str2Ext (see page 3584)	This is function System._Str2Ext.
	_StrInt64 (see page 3584)	This is function System._StrInt64.
	_StrLong (see page 3584)	This is function System._StrLong.
	_StrUInt64 (see page 3585)	This is function System._StrUInt64.
	_TRUNC (see page 3585)	This is function System._TRUNC.
	_Truncate (see page 3585)	This is function System._Truncate.
	_TryFinallyExit (see page 3586)	This is function System._TryFinallyExit.
	_UGetDir (see page 3586)	This is function System._UGetDir.
	_UniqueStringA (see page 3586)	This is function System._UniqueStringA.
	_UniqueStringU (see page 3587)	This is function System._UniqueStringU.
	_UniqueStringW (see page 3587)	This is function System._UniqueStringW.
	_UStrAddRef (see page 3587)	This is function System._UStrAddRef.
	_UStrArrayClr (see page 3588)	This is function System._UStrArrayClr.
	_UStrAsg (see page 3588)	This is function System._UStrAsg.
	_UStrCat (see page 3588)	This is function System._UStrCat.
	_UStrCat3 (see page 3589)	This is function System._UStrCat3.
	_UStrCatN (see page 3589)	This is function System._UStrCatN.
	_UStrClr (see page 3589)	This is function System._UStrClr.
	_UStrCmp (see page 3590)	This is function System._UStrCmp.
	_UStrCopy (see page 3590)	This is function System._UStrCopy.
	_UStrDelete (see page 3590)	This is function System._UStrDelete.
	_UStrEqual (see page 3591)	This is function System._UStrEqual.
	_UStrFromArray (see page 3591)	This is function System._UStrFromArray.
	_UStrFromChar (see page 3591)	This is function System._UStrFromChar.
	_UStrFromLStr (see page 3592)	This is function System._UStrFromLStr.
	_UStrFromPChar (see page 3592)	This is function System._UStrFromPChar.
	_UStrFromPCharLen (see page 3592)	This is function System._UStrFromPCharLen.
	_UStrFromPWChar (see page 3593)	This is function System._UStrFromPWChar.
	_UStrFromPWCharLen (see page 3593)	This is function System._UStrFromPWCharLen.
	_UStrFromString (see page 3593)	This is function System._UStrFromString.
	_UStrFromWArray (see page 3594)	This is function System._UStrFromWArray.
	_UStrFromWChar (see page 3594)	This is function System._UStrFromWChar.
	_UStrFromWStr (see page 3594)	This is function System._UStrFromWStr.
	_UStrInsert (see page 3595)	This is function System._UStrInsert.
	_UStrLAsg (see page 3595)	This is function System._UStrLAsg.
	_UStrLen (see page 3595)	This is function System._UStrLen.
	_UStrSetLength (see page 3596)	This is function System._UStrSetLength.
	_UStrToPWChar (see page 3596)	This is function System._UStrToPWChar.
	_UStrToString (see page 3596)	This is function System._UStrToString.
	_ValExt (see page 3597)	This is function System._ValExt.
	_ValInt64 (see page 3597)	This is function System._ValInt64.
	_ValLong (see page 3597)	This is function System._ValLong.
	_WCharToString (see page 3598)	This is function System._WCharToString.
	_WGetDir (see page 3598)	This is function System._WGetDir.

	_Write0Bool (see page 3598)	This is function System._Write0Bool.
	_Write0Char (see page 3599)	This is function System._Write0Char.
	_Write0CString (see page 3599)	This is function System._Write0CString.
	_Write0Ext (see page 3599)	This is function System._Write0Ext.
	_Write0Int64 (see page 3600)	This is function System._Write0Int64.
	_Write0Long (see page 3600)	This is function System._Write0Long.
	_Write0LString (see page 3600)	This is function System._Write0LString.
	_Write0String (see page 3601)	This is function System._Write0String.
	_Write0UInt64 (see page 3601)	This is function System._Write0UInt64.
	_Write0UString (see page 3601)	This is function System._Write0UString.
	_Write0Variant (see page 3602)	This is function System._Write0Variant.
	_Write0WChar (see page 3602)	This is function System._Write0WChar.
	_Write0WCString (see page 3602)	This is function System._Write0WCString.
	_Write0WString (see page 3603)	This is function System._Write0WString.
	_Write1Ext (see page 3603)	This is function System._Write1Ext.
	_Write2Ext (see page 3603)	This is function System._Write2Ext.
	_WriteBool (see page 3604)	This is function System._WriteBool.
	_WriteChar (see page 3604)	This is function System._WriteChar.
	_WriteCString (see page 3604)	This is function System._WriteCString.
	_WriteInt64 (see page 3605)	This is function System._WriteInt64.
	_WriteLn (see page 3605)	This is function System._WriteLn.
	_WriteLong (see page 3605)	This is function System._WriteLong.
	_WriteLString (see page 3606)	This is function System._WriteLString.
	_WriteRec (see page 3606)	This is function System._WriteRec.
	_WriteString (see page 3606)	This is function System._WriteString.
	_WriteUInt64 (see page 3607)	This is function System._WriteUInt64.
	_WriteUString (see page 3607)	This is function System._WriteUString.
	_WriteVariant (see page 3607)	This is function System._WriteVariant.
	_WriteWChar (see page 3608)	This is function System._WriteWChar.
	_WriteWCString (see page 3608)	This is function System._WriteWCString.
	_WriteWString (see page 3608)	This is function System._WriteWString.
	_WStrAddRef (see page 3609)	This is function System._WStrAddRef.
	_WStrArrayClr (see page 3609)	This is function System._WStrArrayClr.
	_WStrAsg (see page 3609)	This is function System._WStrAsg.
	_WStrCat (see page 3610)	This is function System._WStrCat.
	_WStrCat3 (see page 3610)	This is function System._WStrCat3.
	_WStrCatN (see page 3610)	This is function System._WStrCatN.
	_WStrClr (see page 3611)	This is function System._WStrClr.
	_WStrCmp (see page 3611)	This is function System._WStrCmp.
	_WStrCopy (see page 3611)	This is function System._WStrCopy.
	_WStrDelete (see page 3612)	This is function System._WStrDelete.
	_WStrEqual (see page 3612)	This is function System._WStrEqual.
	_WStrFromArray (see page 3612)	This is function System._WStrFromArray.
	_WStrFromChar (see page 3613)	This is function System._WStrFromChar.
	_WStrFromLStr (see page 3613)	This is function System._WStrFromLStr.
	_WStrFromPChar (see page 3613)	This is function System._WStrFromPChar.
	_WStrFromPCharLen (see page 3614)	This is function System._WStrFromPCharLen.
	_WStrFromPWChar (see page 3614)	This is function System._WStrFromPWChar.
	_WStrFromPWCharLen (see page 3614)	This is function System._WStrFromPWCharLen.
	_WStrFromString (see page 3615)	This is function System._WStrFromString.
	_WStrFromUStr (see page 3615)	This is function System._WStrFromUStr.
	_WStrFromWArray (see page 3615)	This is function System._WStrFromWArray.
	_WStrFromWChar (see page 3616)	This is function System._WStrFromWChar.
	_WStrInsert (see page 3616)	This is function System._WStrInsert.

	_WStrLAsg (see page 3616)	This is function System._WStrLAsg.
	_WStrLCmp (see page 3617)	This is function System._WStrLCmp.
	_WStrLen (see page 3617)	This is function System._WStrLen.
	_WStrSetLength (see page 3617)	This is function System._WStrSetLength.
	_WStrToPWChar (see page 3618)	This is function System._WStrToPWChar.
	_WStrToString (see page 3618)	This is function System._WStrToString.
	+ (see page 3618)	This is function System::+.
	+ (see page 3619)	This is function System::+.
	Abs (see page 3619)	Returns an absolute value.
	AcquireExceptionObject (see page 3619)	Retains access to exception object.
	AddModuleUnloadProc (see page 3620)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 3620)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 3621)	Returns a pointer to a specified object.
	AllocMem (see page 3622)	AllocMem allocates a memory block and initializes each byte to zero.
	AnsiToUtf8 (see page 3624)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 3624)	Prepares an existing file for adding text to the end.
	ArcTan (see page 3625)	Calculates the arctangent of a given number.
	Assert (see page 3627)	Tests whether a Boolean expression is true.
	Assert (see page 3627)	Tests whether a Boolean expression is true.
	Assigned (see page 3628)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 3629)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 3631)	Attempts to use a shared memory manager.
	BeginThread (see page 3632)	Spawns a separate thread of execution.
	BlockRead (see page 3633)	Reads one or more records from an open file into a variable.
	BlockRead (see page 3633)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 3634)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 3634)	Writes one or more records from a variable to an open file.
	Break (see page 3636)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 3638)	Changes the current directory.
	ChDir (see page 3638)	Changes the current directory.
	CheckSafecallResult (see page 3640)	This is function System::CheckSafecallResult.
	Chr (see page 3640)	Returns the character for a specified ASCII value.
	Close (see page 3643)	Terminates the association between a file variable and an external file.
	CloseFile (see page 3643)	Terminates the association between file variable and an external disk file.
	CompareTo (see page 3645)	This is function System.CompareTo.
	CompToCurrency (see page 3645)	Converts a Comp value to a Currency (see page 3242) value.
	CompToDouble (see page 3646)	Converts a Comp value to a double value.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Continue (see page 3647)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 3648)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 3650)	Calculates the cosine of an angle.
	CurrencyToComp (see page 3651)	Converts a Currency (see page 3242) value to a Comp value.
	Dec (see page 3652)	Decrements a variable by 1 or N.
	Dec (see page 3652)	Decrements a variable by 1 or N.
	Delete (see page 3653)	Removes a substring from a string.
	Dispose (see page 3654)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 3656)	Converts a double value to a Comp.
	DynArrayClear (see page 3656)	This is function System.DynArrayClear.
	DynArrayDim (see page 3657)	This is function System.DynArrayDim.
	DynArraySetLength (see page 3657)	This is function System.DynArraySetLength.

	EndThread (see page 3657)	Terminates the execution of a thread.
	EnumModules (see page 3658)	Executes a callback for all the modules in an application.
	EnumModules (see page 3658)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 3658)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 3658)	Executes a callback for all resource modules in an application.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eoln (see page 3660)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 3660)	Tests whether the file pointer is at the end of a line.
	Equals (see page 3661)	This is function System.Equals.
	Erase (see page 3661)	Deletes an external file.
	Error (see page 3662)	Error is used to generate a run-time exception.
	ExceptAddr (see page 3663)	Returns the address at which the current exception was raised.
	ExceptObject (see page 3664)	Returns current exception object.
	Exclude (see page 3664)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 3665)	Exits from the current procedure.
	Exp (see page 3667)	Returns the exponential of X.
	Exp (see page 3667)	Returns the exponential of X.
	FilePos (see page 3668)	Returns the current file position.
	FileSize (see page 3669)	Returns the number of records in a file.
	FillChar (see page 3670)	Fills contiguous bytes with a specified value.
	Finalize (see page 3672)	Uninitializes a dynamically allocated variable.
	Finalize (see page 3672)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 3672)	Returns the instance handle for the module in which a class type is defined.
	FindDynaInst (see page 3673)	This is function System::FindDynaInst.
	FindHInstance (see page 3673)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 3674)	Returns the instance handle of the resource module associated with a specified HInstance.
	Flush (see page 3674)	Empties the buffer of a text file opened for output.
	FPower10 (see page 3675)	This is function System.FPower10.
	Frac (see page 3675)	Returns the fractional part of a real number.
	FreeMem (see page 3676)	FreeMem frees a memory block.
	FreeMem (see page 3676)	FreeMem frees a memory block.
	FreeMemory (see page 3678)	FreeMemory frees a memory block.
	Get8087CW (see page 3679)	Returns the value of the 8087 code word.
	GetCurrent (see page 3679)	This is function System.GetCurrent.
	GetDir (see page 3679)	Returns the current directory.
	GetEnumerator (see page 3681)	This is function System.GetEnumerator.
	GetHeapStatus (see page 3681)	Returns the current status of the memory manager.
	GetLastError (see page 3681)	Returns the last error reported by an operating system API call.
	GetMem (see page 3683)	GetMem allocates a memory block.
	GetMemory (see page 3684)	GetMemory allocates a memory block.
	GetMemoryManager (see page 3686)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 3686)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 3690)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 3691)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 3691)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 3692)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 3692)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 3693)	Initiates abnormal termination of a program.
	Halt (see page 3693)	Initiates abnormal termination of a program.
	Hi (see page 3694)	Returns the high-order byte of X as an unsigned value.
	High (see page 3694)	Returns the highest value in the range of an argument.

	Inc (see page 3696)	Increments an ordinal value by one or N.
	Inc (see page 3696)	Increments an ordinal value by one or N.
	Include (see page 3697)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 3698)	Initializes a dynamically allocated variable.
	Initialize (see page 3698)	Initializes a dynamically allocated variable.
	Insert (see page 3698)	Inserts a substring into a string beginning at a specified point.
	Int (see page 3699)	Returns the integer part of a real number.
	IOResult (see page 3700)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 3701)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3751) procedure.
	IsVariantManagerSet (see page 3701)	Indicates whether variant support is implemented in an application.
	Length (see page 3702)	Returns the number of characters in a string or elements in an array.
	Ln (see page 3703)	Returns the natural log of a real expression.
	Lo (see page 3704)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 3705)	This is function System.LoadResourceModule.
	LoadResourceString (see page 3705)	This is function System::LoadResourceString.
	LoadResString (see page 3705)	This is function System.LoadResString.
	Low (see page 3706)	Returns the lowest value in a range.
	Mark (see page 3707)	This is function System.Mark.
	MemoryBarrier (see page 3707)	This is function System.MemoryBarrier.
	MkDir (see page 3708)	Creates a new subdirectory.
	MkDir (see page 3708)	Creates a new subdirectory.
	MonitorEnter (see page 3709)	This is function System.MonitorEnter.
	MonitorExit (see page 3709)	This is function System.MonitorExit.
	MonitorPulse (see page 3710)	This is function System.MonitorPulse.
	MonitorPulseAll (see page 3710)	This is function System.MonitorPulseAll.
	MonitorTryEnter (see page 3710)	This is function System.MonitorTryEnter.
	MonitorWait (see page 3711)	This is function System.MonitorWait.
	Move (see page 3711)	Copies bytes from a source to a destination.
	MoveChars (see page 3712)	This is function System.MoveChars.
	New (see page 3712)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 3714)	Returns true if argument is an odd number.
	OleStrToString (see page 3715)	Copies data received from a COM interface to a string.
	OleStrToStrVar (see page 3715)	Copies a string from its COM representation to an existing Delphi string.
	OleStrToStrVar (see page 3715)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 3716)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 3716)	Returns the number of parameters passed on the command line.
	ParamStr (see page 3718)	Returns a specified parameter from the command-line.
	Pi (see page 3719)	Returns 3.1415926535897932385.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 3721)	Returns the predecessor of the argument.
	Ptr (see page 3723)	Converts a specified address to a pointer.
	PUCS4Chars (see page 3724)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 3724)	This is function System.RaiseList.
	Random (see page 3725)	Generates random numbers within a specified range.
	Random (see page 3725)	Generates random numbers within a specified range.
	Randomize (see page 3726)	Initializes the random number generator with a random value.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.

	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReallocMem (see page 3729)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 3730)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 3731)	Registers an expected memory leak.
	RegisterModule (see page 3731)	This is function System.RegisterModule.
	Release (see page 3732)	This is function System.Release.
	ReleaseExceptionObject (see page 3732)	Releases exception object acquired by AcquireExceptionObject (see page 3619).
	RemoveModuleUnloadProc (see page 3732)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 3732)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 3733)	Changes the name of an external file.
	Rename (see page 3733)	Changes the name of an external file.
	Reset (see page 3734)	Opens an existing file.
	Reset (see page 3734)	Opens an existing file.
	Reset (see page 3734)	Opens an existing file.
	Reset (see page 3734)	Opens an existing file.
	Rewrite (see page 3736)	Creates a new file and opens it.
	Rewrite (see page 3736)	Creates a new file and opens it.
	RmDir (see page 3738)	Deletes an empty subdirectory.
	RmDir (see page 3738)	Deletes an empty subdirectory.
	Round (see page 3739)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 3740)	Stops execution and generates a runtime error.
	RunError (see page 3740)	Stops execution and generates a runtime error.
	Seek (see page 3741)	Moves the current position of a file to a specified component.
	SeekEof (see page 3742)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 3742)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 3743)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 3743)	Returns the end-of-line status of a file, ignoring whitespace.
	Set8087CW (see page 3744)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3901) declared in the system unit.
	SetCodePage (see page 3746)	This is function System.SetCodePage.
	SetInOutRes (see page 3746)	This is function System.SetInOutRes.
	SetLastError (see page 3747)	SetLastError sets the last API error code.
	SetLength (see page 3748)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 3750)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 3751)	Sets entry points of the memory manager.
	SetMemoryManager (see page 3751)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 3756)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 3756)	SetMultiByteConversionCodePage overrides the default multi byte code conversion page.
	SetRaiseList (see page 3757)	This is function System.SetRaiseList.
	SetString (see page 3757)	Sets the contents and length of the given string.
	SetTextBuf (see page 3758)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 3758)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 3759)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 3759)	Shares the module's memory manager.
	Sin (see page 3760)	Returns the sine of the angle in radians.
	SizeOf (see page 3761)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 3762)	Returns a sub-section of an array.
	Sqr (see page 3762)	Returns the square of a number.
	Sqr (see page 3762)	Returns the square of a number.

	Sqrt (see page 3763)	Returns the square root of X.
	Str (see page 3764)	Formats a string and returns it to a variable.
	Str (see page 3764)	Formats a string and returns it to a variable.
	StringCodePage (see page 3765)	This is function System.StringCodePage.
	StringCodePage (see page 3765)	This is function System.StringCodePage.
	StringElementSize (see page 3765)	This is function System.StringElementSize.
	StringElementSize (see page 3766)	This is function System.StringElementSize.
	StringOfChar (see page 3766)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 3766)	Returns a string with a specified number of repeating characters.
	StringRefCount (see page 3766)	This is function System.StringRefCount.
	StringRefCount (see page 3767)	This is function System.StringRefCount.
	StringToOleStr (see page 3767)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToOleStr (see page 3767)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 3768)	Returns a UNICODE string from an AnsiString.
	Succ (see page 3768)	Returns the successor of an argument.
	Swap (see page 3769)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 3769)	This is function System.SysAllocMem.
	SysFreeMem (see page 3770)	Frees the memory pointed to by a specified pointer.
	SysGetMem (see page 3770)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 3771)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 3772)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 3772)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 3772)	This is function System.TextStart.
	Trunc (see page 3773)	Truncates a real number to an integer.
	Truncate (see page 3774)	Deletes all the records after the current file position.
	UCS4StringToUnicodeString (see page 3775)	This is function System.UCS4StringToUnicodeString.
	UCS4StringToWideString (see page 3775)	Converts a string of 4-byte Unicode characters into a WideString (see page 3476) value.
	UnicodeStringToUCS4String (see page 3775)	This is function System.UnicodeStringToUCS4String.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 3777)	Unregisters an expected memory leak.
	UnregisterModule (see page 3778)	This is function System.UnregisterModule.
	UpCase (see page 3778)	Converts a character to uppercase.
	UpCase (see page 3778)	Converts a character to uppercase.
	UTF8Decode (see page 3779)	Converts a UTF8 string to a Unicode string (WideString (see page 3476)).
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8EncodeToShortString (see page 3780)	This is function System.UTF8EncodeToShortString.
	UTF8EncodeToShortString (see page 3780)	This is function System.UTF8EncodeToShortString.
	UTF8EncodeToShortString (see page 3781)	This is function System.UTF8EncodeToShortString.
	Utf8ToAnsi (see page 3781)	Converts a UTF8 string to a string of Ansi characters.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3783)	This is function System.UTF8ToString.
	Utf8ToUnicode (see page 3783)	Converts a string of UTF-8 characters to a Unicode string.
	Utf8ToUnicode (see page 3783)	Converts a string of UTF-8 characters to a Unicode string.

	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToWideString (see page 3785)	This is function System.UTF8ToWideString.
	Val (see page 3785)	Converts a string to a numeric representation.
	VarArrayRedim (see page 3786)	Resizes a Variant (see page 3430) array.
	VarClear (see page 3786)	Empties a Variant (see page 3430), so that it is unassigned.
	WideCharLenToString (see page 3787)	Converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 3788)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 3788)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharToString (see page 3788)	Converts null-terminated Unicode string to single or multi byte character data.
	WideCharToStrVar (see page 3788)	Converts Unicode string to a single or multi byte character data.
	WideStringToUCS4String (see page 3789)	Converts a WideString (see page 3476) value to a string of 4-byte Unicode characters.
	Write (see page 3789)	Writes to either a typed file or a text file
	Write (see page 3789)	Writes to either a typed file or a text file
	Write (see page 3789)	Writes to either a typed file or a text file
	Write (see page 3789)	Writes to either a typed file or a text file
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	YieldProcessor (see page 3793)	This is function System.YieldProcessor.

Interfaces

	Name	Description
	IComparable (see page 3497)	This is class System.IComparable.
	IDispatch (see page 3498)	This is class System.IDispatch.
	IEnumerable (see page 3500)	This is class System.IEnumerable.
	IEnumerator (see page 3501)	This is class System.IEnumerator.
	IInterface (see page 3503)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 3505)	IInvokable is the base class for invokable interfaces.

Macros

Name	Description
USE (see page 3960)	This is macro System::USE.
USEADDITIONALFILES (see page 3960)	This is macro System::USEADDITIONALFILES.
USEASM (see page 3961)	This is macro System::USEASM.
USEDATAMODULE (see page 3961)	This is macro System::USEDATAMODULE.
USEDATAMODULENS (see page 3961)	This is macro System::USEDATAMODULENS.
USEDEF (see page 3962)	This is macro System::USEDEF.
USEFILE (see page 3962)	This is macro System::USEFILE.
USEFORM (see page 3963)	This is macro System::USEFORM.
USEFORMNS (see page 3963)	This is macro System::USEFORMNS.
USEFORMRES (see page 3963)	This is macro System::USEFORMRES.
USEIDL (see page 3964)	This is macro System::USEIDL.
USELIB (see page 3964)	This is macro System::USELIB.
USEOBJ (see page 3964)	This is macro System::USEOBJ.
USEPACKAGE (see page 3965)	This is macro System::USEPACKAGE.
USERC (see page 3965)	This is macro System::USERC.
USERES (see page 3965)	This is macro System::USERES.
USETLB (see page 3966)	This is macro System::USETLB.
USEUNIT (see page 3966)	This is macro System::USEUNIT.

Structs, Records, Enums

	Name	Description
	PackageInfoTable (see page 3795)	This is record System.PackageInfoTable.
	PackageUnitEntry (see page 3795)	This is record System.PackageUnitEntry.
	TCallDesc (see page 3795)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3830) is a pointer to this descriptior.
	TChunkStatus (see page 3796)	TChunkStatus enumerates the possible states of a chunk of memory.
	TCVModInfo (see page 3797)	This is record System.TCVModInfo.
	TDispatchMessage (see page 3797)	This is record System.TDispatchMessage.
	TDispDesc (see page 3798)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 3798)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 3799)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 3800)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapBlock (see page 3802)	This is record System::THeapBlock.
	THeapStatus (see page 3802)	This is record System::THeapStatus.
	TInitContext (see page 3803)	This is record System.TInitContext.
	TInterfaceEntry (see page 3803)	TInterfaceTable (see page 3804) lists the interfaces that a class supports, and PInterfaceTable (see page 3839) points to this list.
	TInterfaceTable (see page 3804)	This is record System.TInterfaceTable.
	TLibModule (see page 3804)	This is record System::TLibModule.
	TMemoryManager (see page 3805)	This is record System::TMemoryManager.
	TMemoryManagerEx (see page 3805)	This is record System::TMemoryManagerEx.
	TMemoryManagerState (see page 3806)	This is record System::TMemoryManagerState.
	TMethod (see page 3806)	TMethod stores Code and Data fields to represent a method.
	TMinimumBlockAlignment (see page 3807)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TModuleUnloadRec (see page 3808)	This is record System::TModuleUnloadRec.
	TMonitorSupport (see page 3808)	This is record System.TMonitorSupport.
	TResStringRec (see page 3809)	This is record System::TResStringRec.
	TRuntimeError (see page 3809)	This is record System.TRuntimeError.
	TSmallBlockTypeState (see page 3810)	This is record System::TSmallBlockTypeState.
	TTextLineBreakStyle (see page 3811)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.
	TTextRec (see page 3811)	TTextRec is the internal format for text files.
	TVarArray (see page 3812)	This is record System::TVarArray.
	TVarArrayBound (see page 3813)	This is record System::TVarArrayBound.
	TVarData (see page 3813)	TVarData represents Variant (see page 3430) information.
	TVariantManager (see page 3817)	This is record System::TVariantManager.
	TVarRec (see page 3819)	This is class System::TVarRec.
	TCallDesc (see page 3820)	This is type System::TCallDesc.
	TDispDesc (see page 3820)	This is type System::TDispDesc.

Types

Name	Description
CppLongInt (see page 3824)	This is type System.CppLongInt.
CppULongInt (see page 3825)	This is type System.CppULongInt.
GetPackageInfoTable (see page 3825)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 3825)	This is type System.HGLOBAL.
HINST (see page 3826)	This is type System.HINST.
HMODULE (see page 3826)	This is type System.HMODULE.
HRESULT (see page 3826)	This is type System.HRESULT.
HRSRC (see page 3827)	This is type System.HRSRC.
Int16 (see page 3827)	This is type System.Int16.
Int32 (see page 3827)	This is type System.Int32.
Int8 (see page 3828)	This is type System.Int8.

IntegerArray (see page 3828)	This is type System.IntegerArray.
IUnknown (see page 3828)	IUnknown is a special base interface.
PackageInfo (see page 3829)	This is type System.PackageInfo.
PAnsiString (see page 3829)	PAnsiString is a pointer to an AnsiString variable.
PBoolean (see page 3829)	PBoolean is a pointer to a Boolean variable.
PByte (see page 3830)	PByte is a pointer to a Byte variable.
PCallDesc (see page 3830)	This is type System::PCallDesc.
PCardinal (see page 3831)	PCardinal is a pointer to a Cardinal variable.
PComp (see page 3831)	PComp is a pointer to a Comp variable.
PCurrency (see page 3831)	PCurrency is a pointer to a Currency (see page 3242) variable.
PCVModInfo (see page 3832)	This is type System.PCVModInfo.
PDate (see page 3832)	PDate is a pointer to a Date variable.
PDateTime (see page 3833)	PDateTime is a pointer to a TDateTime (see page 3335) variable.
PDispatch (see page 3833)	This is type System.PDispatch.
PDispDesc (see page 3833)	This is type System::PDispDesc.
PDouble (see page 3834)	PDouble is a pointer to a Double variable.
PDynArrayTypeInfo (see page 3834)	This is type System.PDynArrayTypeInfo.
PError (see page 3834)	This is type System.PError.
PExtended (see page 3835)	PExtended is a pointer to an Extended variable.
PGUID (see page 3835)	PGUID is a pointer to a TGUID (see page 3800) variable.
PInitContext (see page 3837)	This is type System.PInitContext.
PInt64 (see page 3837)	PInt64 is a pointer to an Int64 variable.
PInteger (see page 3837)	PInteger is a pointer to an Integer variable.
PIntegerArray (see page 3838)	PIntegerArray is a pointer to an IntegerArray (see page 3828) variable.
PInterfaceEntry (see page 3838)	PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3803) variable.
PInterfaceTable (see page 3839)	PInterfaceTable is a pointer to a TInterfaceTable (see page 3804) variable.
PLibModule (see page 3839)	PLibModule is a pointer to a TLibModule (see page 3804) variable.
PLongint (see page 3839)	This is type System.PLongint.
PLongWord (see page 3840)	PLongWord is a pointer to a LongWord variable.
PMemoryManager (see page 3840)	This is type System::PMemoryManager.
PMemoryManagerEx (see page 3840)	This is type System::PMemoryManagerEx.
PModuleUnloadRec (see page 3841)	This is type System::PModuleUnloadRec.
PMonitor (see page 3841)	This is type System.PMonitor.
PMonitorSupport (see page 3841)	This is type System.PMonitorSupport.
PointerArray (see page 3842)	This is type System.PointerArray.
POleVariant (see page 3842)	This is type System.POleVariant.
PPAnsiChar (see page 3842)	This is type System.PPAnsiChar.
PPCharArray (see page 3843)	This is type System.PPCharArray.
PPDispatch (see page 3843)	This is type System.PPDispatch.
PPMonitor (see page 3843)	This is type System.PPMonitor.
PPointer (see page 3844)	PPointer is a pointer to a Pointer variable.
PPointerArray (see page 3844)	PPointerArray is a pointer to a PointerArray (see page 3842) variable.
PPUnknown (see page 3844)	This is type System.PPUnknown.
PPWideChar (see page 3845)	This is type System.PPWideChar.
PRawByteString (see page 3845)	This is type System.PRawByteString.
PResStringRec (see page 3845)	This is type System::PResStringRec.
PShortInt (see page 3846)	PShortInt is a pointer to a ShortInt variable.
PShortString (see page 3846)	PShortString is a pointer to a ShortString variable.
PSingle (see page 3846)	PSingle is a pointer to a Single variable.
PSmallBlockTypeState (see page 3847)	This is type System::PSmallBlockTypeState.
PSmallInt (see page 3847)	PSmallInt is a pointer to a SmallInt variable.
PTextBuf (see page 3848)	This is type System.PTextBuf.
PUCS4Char (see page 3848)	This is type System.PUCS4Char.
PUCS4CharArray (see page 3848)	This is type System.PUCS4CharArray.

PUInt64 (see page 3849)	This is type System.PUInt64.
PUnicodeString (see page 3849)	This is type System.PUnicodeString.
PUnitEntryTable (see page 3849)	This is type System.PUnitEntryTable.
PUnknown (see page 3850)	PUnknown is a pointer to an IUnknown (see page 3828) variable.
PUTF8String (see page 3850)	This is type System.PUTF8String.
PVarArray (see page 3850)	This is type System.PVarArray.
PVarArrayBoundArray (see page 3851)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 3851)	This is type System.PVarArrayCoorArray.
PVarData (see page 3851)	This is type System.PVarData.
PVariant (see page 3852)	PVariant is a pointer to a Variant (see page 3430) variable.
PVariantManager (see page 3852)	This is type System::PVariantManager.
PVarRec (see page 3852)	This is type System::PVarRec.
PWideString (see page 3853)	PWideString is a pointer to a WideString (see page 3476) variable.
PWord (see page 3853)	PWord is a pointer to a Word variable.
PWordBool (see page 3853)	PWordBool is a pointer to a WordBool variable.
RawByteString (see page 3854)	RawByteString enables the passing of string data of any code page without doing any codepage conversions.
ResourceString (see page 3854)	This is type System::ResourceString.
TAssertErrorProc (see page 3855)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 3855)	This is type System::TBooleanDynArray.
TBoundArray (see page 3855)	This is type System.TBoundArray.
TByteDynArray (see page 3856)	This is type System::TByteDynArray.
TCardinalDynArray (see page 3856)	This is type System::TCardinalDynArray.
TClass (see page 3856)	TClass defines the metaclass for System::TObject (see page 3364).
TClassHelperBaseClass (see page 3857)	This is type System.TClassHelperBaseClass.
TDate (see page 3857)	This is type System.TDate.
TDateTime (see page 3858)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 3859)	This is type System.TDLLProc.
TDLLProcEx (see page 3859)	This is type System.TDLLProcEx.
TDoubleDynArray (see page 3859)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 3860)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 3860)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 3860)	This is type System.TFileIOFunc.
TInt64DynArray (see page 3861)	This is type System::TInt64DynArray.
TIntegerDynArray (see page 3861)	This is type System::TIntegerDynArray.
TInterfacedClass (see page 3861)	TInterfacedClass defines the metaclass for TInterfacedObject (see page 3355).
TLongWordDynArray (see page 3862)	This is type System::TLongWordDynArray.
TMemoryMap (see page 3862)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3796) values.
TModuleUnloadProc (see page 3863)	TModuleUnloadProc dehines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 3863)	This is type System.TModuleUnloadProcLW.
TPCharArray (see page 3863)	This is type System.TPCharArray.
TResourceHandle (see page 3864)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 3864)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 3864)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 3865)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 3865)	This is type System::TSmallBlockTypeStates.
TSmallIntDynArray (see page 3865)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 3866)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 3866)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 3866)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 3867)	This is type System.TTextBuf.
TTextIOFunc (see page 3867)	This is type System.TTextIOFunc.
TThreadFunc (see page 3867)	This is type System::TThreadFunc.

TTime (see page 3868)	This is type System.TTime.
TUCS4CharArray (see page 3868)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 3868)	This is type System.TVarArrayBoundArray.
TVarArrayCoorArray (see page 3869)	This is type System.TVarArrayCoorArray.
TVarOp (see page 3869)	TVarOp defines a Variant (see page 3430) operation.
TWideStringDynArray (see page 3870)	This is type System::TWideStringDynArray.
TWordDynArray (see page 3870)	This is type System::TWordDynArray.
UCS4Char (see page 3871)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 3871)	UCS4String is used for working with 4–byte Unicode characters.
UInt16 (see page 3872)	This is type System.UInt16.
UInt32 (see page 3872)	This is type System.UInt32.
UInt8 (see page 3872)	This is type System.UInt8.
UnitEntryTable (see page 3873)	This is type System.UnitEntryTable.
UTF8String (see page 3873)	UTF8String represents UTF-8 encoded Unicode strings.

Variables

Name	Description
AbstractErrorProc (see page 3875)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 3875)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 3877)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 3878)	Points to the assertion error-handler.
CmdLine (see page 3880)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 3880)	Specifies the flags for displaying the application's main window.
CPUCount (see page 3881)	This is variable System.CPUCount.
DebugHook (see page 3881)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 3881)	Specifies the Default 8087 control word.
DefaultSystemCodePage (see page 3883)	This is variable System.DefaultSystemCodePage.
DefaultTextLineBreakStyle (see page 3883)	Specifies the characters that are used by default to separate lines in text.
DefaultUnicodeCodePage (see page 3884)	This is variable System.DefaultUnicodeCodePage.
DispCallByIDProc (see page 3884)	This is variable System.DispCallByIDProc.
DLLProc (see page 3884)	Points to a procedure invoked by a DLL entry point.
envp (see page 3885)	This is variable System.envp.
ErrorAddr (see page 3885)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 3886)	Points to the RTL runtime error handler.
ErrOutput (see page 3888)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 3888)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionAcquired (see page 3889)	This is variable System.ExceptionAcquired.
ExceptionClass (see page 3889)	This is variable System.ExceptionClass.
ExceptObjProc (see page 3889)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 3890)	Points to the lowest-level exception handler.
ExitCode (see page 3891)	Contains the application's exit code.
ExitProc (see page 3891)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 3892)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 3892)	Indicates the access mode on typed and untyped files opened by the Reset (see page 3734) procedure.
HeapAllocFlags (see page 3893)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 3894)	This is variable System.HPrevInst.
InitProc (see page 3894)	InitProc is the last installed initialization procedure.
Input (see page 3895)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 3896)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 3896)	Indicates whether the module is a shared library.
IsMultiThread (see page 3897)	Indicates whether the application spawned additional threads using BeginThread (see page 3632) or TThread objects.
JITEnable (see page 3897)	Controls when the just-in-time debugger is called.

LibModuleList (see page 3898)	Lists module records for enumeration procedures.
MainInstance (see page 3898)	Indicates the instance handle for the main executable.
MainThreadID (see page 3899)	Indicates the instance handle for the current module's main thread of execution.
ModuleUnloadList (see page 3899)	This is variable System.ModuleUnloadList.
MonitorSupport (see page 3900)	This is variable System.MonitorSupport.
NeverSleepOnMMThreadContention (see page 3900)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 3901)	Controls whether the application displays an error message when a runtime error occurs.
Output (see page 3901)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 3902)	Points to RaiseException Windows function.
RaiseExceptObjProc (see page 3902)	This is variable System.RaiseExceptObjProc.
RandSeed (see page 3902)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 3903)	Report memory leaks on shutdown.
RTLUnwindProc (see page 3903)	This is variable System.RTLUnwindProc.
SafeCallErrorProc (see page 3904)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 3904)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 3905)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 3905)	This is variable System.Test8086.
Test8087 (see page 3905)	This is variable System.Test8087.
TestFDIV (see page 3906)	This is variable System.TestFDIV.
UTF8CompareLocale (see page 3906)	This is variable System.UTF8CompareLocale.
VarAddRefProc (see page 3906)	This is variable System.VarAddRefProc.
VarClearProc (see page 3907)	This is variable System.VarClearProc.
VarCopyProc (see page 3907)	This is variable System.VarCopyProc.
VarToLStrProc (see page 3907)	This is variable System.VarToLStrProc.
VarToWStrProc (see page 3908)	This is variable System.VarToWStrProc.

38.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	AnsiStringBase (see page 3185)	System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.
	Currency (see page 3242)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 3262)	This is class System::CurrencyBase.
	DynArrayException (see page 3279)	This is class System::DynArrayException.
	DynArrayNullData (see page 3279)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 3279)	This is class System::DynArrayOutOfRange.
	OleVariant (see page 3281)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	TAggregatedObject (see page 3328)	This is class System::TAggregatedObject.
	TClassHelperBase (see page 3331)	This is class System.TClassHelperBase.
	TContainedObject (see page 3333)	This is class System::TContainedObject.
	TDateTime (see page 3335)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.
	TDateTimeBase (see page 3354)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 3355)	This is class System::TInterfacedObject.
	TMetaClass (see page 3360)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TObject (see page 3364)	System::TObject is the ultimate ancestor of all objects and components.
	UnicodeString (see page 3393)	System::UnicodeString is the C++ analog for the Delphi UnicodeString type.
	Variant (see page 3430)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	WideString (see page 3476)	System::WideString represents strings of 16-bit characters.

Records

	Name	Description
	TMonitor (see page 3361)	This is class System.TMonitor.

Templates

	Name	Description
	AnsiStringT (see page 3219)	System::AnsiStringT is the C++ analog for the Delphi long string type.
	DelphiInterface (see page 3263)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 3268)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	SmallString (see page 3322)	This is class System::SmallString.
	SmallStringBase (see page 3324)	This is class System::SmallStringBase.
	sPointer (see page 3326)	This is class System::sPointer.

38.1.1 AnsiStringBase Class

System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.

Class Hierarchy

System::AnsiStringBase

Pascal

```
AnsiStringBase = class;
```

C++

```
class AnsiStringBase;
```

File

dstring.h

Description

System::AnsiStringBase exists to provide a place for methods/members that are common to all instantiations of the template class AnsiStringT (see page 3219)<CodePage (see page 3199)>. AnsiString itself is a typedef that is equivalent to AnsiStringT (see page 3219)<0>. You should never deal directly with System::AnsiStringBase.

System::AnsiStringT (see page 3219) is a template class that you use when handling a Delphi AnsiString that has a code page specified with it.

Delphi uses several string types. One important long string type is commonly known as System::AnsiString. Support for this type includes the following features:

- strings can be as large as available memory
- efficient use of memory is ensured through shared references
- routines and operators evaluate strings based on the current locale

Despite its name, System::AnsiStringBase is not restricted to the ANSI character set, and can use any character set supported by the current locale definition, including multi-byte and Unicode character sets.

System::AnsiStringBase variables that have not been assigned an initial value contain a zero-length string.

To use the C++ streaming operators (**<<** and **>>**) with System::AnsiStringBase, you must `iostream` before including `system.hpp` and use `AnsiStringT::c_str()` to return the internal string representation. The following example demonstrates the usage of AnsiString with stream operators:

```
#include <iostream>
#include <system.hpp>
int main() {
  AnsiString HelloStr = "hello";
  std::cout << HelloStr.c_str();
}
```

Note: Delphi also supports System::AnsiString, but implements it as a primitive type rather than a class.

UnicodeString (see page 3393) has become the default string type. By default, variables declared as type String are System::UnicodeString (see page 3393).

See Also

How to Handle Delphi AnsiString Code Page Specification in C++

System::WideString (see page 3476)

System::AnsiStringBase::[] (see page 3216)

System::AnsiStringBase::= (see page 3217)

System::AnsiStringBase::== (see page 3218)

System::AnsiStringBase::!= (see page 3215)

System::AnsiStringBase::> (see page 3218)

System::AnsiStringBase::>= (see page 3218)

System::AnsiStringBase::< (see page 3216)

System::AnsiStringBase::<= (see page 3217)

System::AnsiStringBase::+ (see page 3216)

System::AnsiStringT::+= (see page 3240)

38.1.1.1 AnsiStringBase Members

The following tables list the members exposed by AnsiStringBase.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3193)	This is record System::AnsiStringBase::TStringFloatFormat.
	TStringMbcsByteType (see page 3193)	This is record System::AnsiStringBase::TStringMbcsByteType.

Structures

	Name	Description
	StrRec (see page 3192)	This is record System::AnsiStringBase::StrRec.

Protected Data Members

	Name	Description
	Data (see page 3194)	This is Data, a member of class AnsiStringBase.

Protected Methods

	Name	Description
	_AnsiCat (see page 3195)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3196)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiCompare (see page 3196)	Compares the System::AnsiStringBase::AnsiString to another specified string (case-sensitively).
	AnsiCompareIC (see page 3197)	Compares the System::AnsiStringBase::AnsiString to another specified string (case insensitively).
	AnsiStringBase (see page 3189)	This is the overview for the AnsiStringBase constructor overload.
	GetRec (see page 3204)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3211)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Protected Operators

	Name	Description
	!= (see page 3215)	Compares two strings.
	< (see page 3216)	Compares two strings.
	<= (see page 3217)	Compares two strings.
	= (see page 3217)	Assigns the string rhs to this System::AnsiStringBase::AnsiString.
	== (see page 3218)	Compares the System::AnsiStringBase::AnsiString with a specified string, returning true if they are the same.
	> (see page 3218)	Compares two strings.
	>= (see page 3218)	Compares two strings.

Public Methods

	Name	Description
	~AnsiStringBase (see page 3189)	This is ~AnsiStringBase, a member of class AnsiStringBase.
	AnsiLastChar (see page 3197)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiString.
	AnsiPos (see page 3197)	Returns the index at which the specified substring begins.
	BytesOf (see page 3198)	This is BytesOf, a member of class AnsiStringBase.

	ByteType (see page 3198)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase::AnsiString.
	cat_vprintf (see page 3199)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiString.
	CodePage (see page 3199)	Gets code page of a System::AnsiStringBase (see page 3185).
	CurrToStr (see page 3199)	Converts a currency value to its string representation.
	CurrToStrF (see page 3200)	Converts a currency value to its string representation using the indicated float format convention.
	Delete (see page 3200)	Removes a specified number of characters from the string.
	ElementSize (see page 3200)	Gets element size of data in a System::AnsiStringBase (see page 3185).
	EnsureAnsi (see page 3201)	This is EnsureAnsi, a member of class AnsiStringBase.
	FloatToStrF (see page 3201)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3202)	Formats output using a resourced format string.
	Format (see page 3202)	System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase::AnsiString, format, and a series of array arguments, args.
	FormatFloat (see page 3203)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3205)	System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase::AnsiString.
	IntToHex (see page 3205)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3206)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3206)	Returns true if the System::AnsiStringBase::AnsiString is empty.
	IsLeadByte (see page 3206)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3206)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3207)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3207)	This is LastChar, a member of class AnsiStringBase.
	LastDelimiter (see page 3207)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3208)	Returns the length of the System::AnsiStringBase::AnsiString.
	LoadStr (see page 3208)	This is the overview for the LoadStr method overload.
	LoadString (see page 3209)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3209)	Returns a lower-case version of System::AnsiStringBase::AnsiString.
	Pos (see page 3209)	Returns the index at which a specified substring begins.
	RefCount (see page 3210)	Gets reference count of System::AnsiStringBase (see page 3185).
	SetLength (see page 3210)	Changes the length of the System::AnsiStringBase::AnsiString.
	StringOfChar (see page 3210)	Returns a string with the specified number of characters.
	SubString (see page 3211)	Returns a specified substring of the System::AnsiStringBase::AnsiString.
	swap (see page 3211)	This is swap, a member of class AnsiStringBase.
	ToDouble (see page 3211)	Converts the string to a floating-point value.
	ToInt (see page 3212)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3212)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3212)	Returns a new System::AnsiStringBase::AnsiString without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3212)	Returns a new System::AnsiStringBase::AnsiString without leading spaces or control characters.
	TrimRight (see page 3213)	Returns a new System::AnsiStringBase::AnsiString without trailing spaces or control characters.
	Unique (see page 3213)	Makes the System::AnsiStringBase::AnsiString unique.
	UpperCase (see page 3213)	Returns an upper-case version of this System::AnsiStringBase::AnsiString.
	vprintf (see page 3214)	Sets the value of the System::AnsiStringBase::AnsiString given a format string and its arguments.
	WideChar (see page 3214)	Converts the System::AnsiStringBase::AnsiString to a wide-character array.
	WideCharBufSize (see page 3214)	Returns the size of the buffer required by the WideChar (see page 3214) function.

Public Operators

	Name	Description
	[] (see page 3216)	This is the overview for the [] operator overload.
	+ (see page 3216)	Concatenates two System::AnsiStringBase::AnsiString values.

38.1.1.2 AnsiStringBase::~AnsiStringBase Destructor

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~AnsiStringBase();
```

Description

This is ~AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3 AnsiStringBase Constructor

This is the overview for the AnsiStringBase constructor overload.

Overload List

	Name	Description
	AnsiStringBase::AnsiStringBase () (see page 3190)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (AnsiStringBase&) (see page 3190)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (char*, int) (see page 3190)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (char*, int, int) (see page 3191)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (char16_t*, int, int) (see page 3191)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (char32_t*, int, int) (see page 3191)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (double, int) (see page 3191)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (UnicodeString &, int) (see page 3190)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (wchar_t*, int, int) (see page 3192)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (wchar_t, int) (see page 3192)	This is AnsiStringBase, a member of class AnsiStringBase.
	AnsiStringBase::AnsiStringBase (WideString &, int) (see page 3190)	This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.1 AnsiStringBase::AnsiStringBase Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall AnsiStringBase();
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.2 AnsiStringBase::AnsiStringBase Constructor (AnsiStringBase&)

Pascal

```
constructor Create(const src: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringBase(const AnsiStringBase& src);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.3 AnsiStringBase::AnsiStringBase Constructor (UnicodeString &, int)

Pascal

```
constructor Create(const src: UnicodeString &; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const UnicodeString & src, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.4 AnsiStringBase::AnsiStringBase Constructor (WideString &, int)

Pascal

```
constructor Create(const src: WideString &; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const WideString & src, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.5 AnsiStringBase::AnsiStringBase Constructor (char*, int)

Pascal

```
constructor Create(const src: char*; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const char* src, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.6 AnsiStringBase::AnsiStringBase Constructor (char*, int, int)

Pascal

```
constructor Create(const src: char*; byteLen: Integer; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const char* src, int byteLen, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.7 AnsiStringBase::AnsiStringBase Constructor (char16_t*, int, int)

Pascal

```
constructor Create(const src: char16_t*; numChar16: Integer; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const char16_t* src, int numChar16, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.8 AnsiStringBase::AnsiStringBase Constructor (char32_t*, int, int)

Pascal

```
constructor Create(const src: char32_t*; numChar32: Integer; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const char32_t* src, int numChar32, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.9 AnsiStringBase::AnsiStringBase Constructor (double, int)

Pascal

```
constructor Create(src: double; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(double src, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.10 AnsiStringBase::AnsiStringBase Constructor (wchar_t, int)

Pascal

```
constructor Create(src: wchar_t; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(wchar_t src, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.3.11 AnsiStringBase::AnsiStringBase Constructor (wchar_t*, int, int)

Pascal

```
constructor Create(const src: wchar_t*; numwchar: Integer; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase(const wchar_t* src, int numwchar, int codePage);
```

Description

This is AnsiStringBase, a member of class AnsiStringBase.

38.1.1.4 AnsiStringBase Structures

The structures of the AnsiStringBase class are listed here.

Structures

	Name	Description
	StrRec (see page 3192)	This is record System::AnsiStringBase::StrRec.

38.1.1.4.1 System::AnsiStringBase::StrRec Structure

Pascal

```
StrRec = record
  codePage: unsigned short;
  elemSize: unsigned short;
  refCnt: Integer;
  length: Integer;
end;
```

C++

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

File

dstring.h

Description

This is record System::AnsiStringBase::StrRec.

38.1.1.5 AnsiStringBase Enumerations

The enumerations of the AnsiStringBase class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3193)	This is record System::AnsiStringBase::TStringFloatFormat.
	TStringMbcsByteType (see page 3193)	This is record System::AnsiStringBase::TStringMbcsByteType.

38.1.1.5.1 System::AnsiStringBase::TStringFloatFormat Enumeration

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

dstring.h

Description

This is record System::AnsiStringBase::TStringFloatFormat.

38.1.1.5.2 System::AnsiStringBase::TStringMbcsByteType Enumeration

Pascal

```
TStringMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
);
```

C++

```
enum TStringMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

dstring.h

Description

This is record System::AnsiStringBase::TStringMbcsByteType.

38.1.1.6 AnsiStringBase Data Members

The data members of the AnsiStringBase class are listed here.

Protected Data Members

	Name	Description
	Data (see page 3194)	This is Data, a member of class AnsiStringBase.

38.1.1.6.1 AnsiStringBase::Data Data Member

Pascal

```
Data: char *;
```

C++

```
char * Data;
```

Description

This is Data, a member of class AnsiStringBase.

38.1.1.7 AnsiStringBase Methods

The methods of the AnsiStringBase class are listed here.

Protected Methods

	Name	Description
	_AnsiCat (see page 3195)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3196)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiCompare (see page 3196)	Compares the System::AnsiStringBase::AnsiString to another specified string (case-sensitively).
	AnsiCompareIC (see page 3197)	Compares the System::AnsiStringBase::AnsiString to another specified string (case insensitively).
	GetRec (see page 3204)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3211)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Public Methods

	Name	Description
	AnsiLastChar (see page 3197)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiString.
	AnsiPos (see page 3197)	Returns the index at which the specified substring begins.
	BytesOf (see page 3198)	This is BytesOf, a member of class AnsiStringBase.
	ByteType (see page 3198)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase::AnsiString.
	cat_vprintf (see page 3199)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiString.
	CodePage (see page 3199)	Gets code page of a System::AnsiStringBase (see page 3185).
	CurrToStr (see page 3199)	Converts a currency value to its string representation.
	CurrToStrF (see page 3200)	Converts a currency value to its string representation using the indicated float format convention.
	Delete (see page 3200)	Removes a specified number of characters from the string.

	ElementSize (see page 3200)	Gets element size of data in a System::AnsiStringBase (see page 3185).
	EnsureAnsi (see page 3201)	This is EnsureAnsi, a member of class AnsiStringBase.
	FloatToStrF (see page 3201)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3202)	Formats output using a resourced format string.
	Format (see page 3202)	System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase::AnsiString, format, and a series of array arguments, args.
	FormatFloat (see page 3203)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3205)	System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase::AnsiString.
	IntToHex (see page 3205)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3206)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3206)	Returns true if the System::AnsiStringBase::AnsiString is empty.
	IsLeadByte (see page 3206)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3206)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3207)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3207)	This is LastChar, a member of class AnsiStringBase.
	LastDelimiter (see page 3207)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3208)	Returns the length of the System::AnsiStringBase::AnsiString.
	LoadStr (see page 3208)	This is the overview for the LoadStr method overload.
	LoadString (see page 3209)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3209)	Returns a lower-case version of System::AnsiStringBase::AnsiString.
	Pos (see page 3209)	Returns the index at which a specified substring begins.
	RefCount (see page 3210)	Gets reference count of System::AnsiStringBase (see page 3185).
	SetLength (see page 3210)	Changes the length of the System::AnsiStringBase::AnsiString.
	StringOfChar (see page 3210)	Returns a string with the specified number of characters.
	SubString (see page 3211)	Returns a specified substring of the System::AnsiStringBase::AnsiString.
	swap (see page 3211)	This is swap, a member of class AnsiStringBase.
	ToDouble (see page 3211)	Converts the string to a floating-point value.
	ToInt (see page 3212)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3212)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3212)	Returns a new System::AnsiStringBase::AnsiString without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3212)	Returns a new System::AnsiStringBase::AnsiString without leading spaces or control characters.
	TrimRight (see page 3213)	Returns a new System::AnsiStringBase::AnsiString without trailing spaces or control characters.
	Unique (see page 3213)	Makes the System::AnsiStringBase::AnsiString unique.
	UpperCase (see page 3213)	Returns an upper-case version of this System::AnsiStringBase::AnsiString.
	vprintf (see page 3214)	Sets the value of the System::AnsiStringBase::AnsiString given a format string and its arguments.
	WideChar (see page 3214)	Converts the System::AnsiStringBase::AnsiString to a wide-character array.
	WideCharBufSize (see page 3214)	Returns the size of the buffer required by the WideChar (see page 3214) function.

38.1.1.7.1 _AnsiCat Method

This is the overview for the _AnsiCat method overload.

Overload List

	Name	Description
	AnsiStringBase::_AnsiCat (AnsiStringBase&, AnsiStringBase&) (see page 3196)	This is _AnsiCat, a member of class AnsiStringBase.

	AnsiStringBase::_AnsiCat (AnsiStringBase&, AnsiStringBase&, AnsiStringBase&) (see page 3196)	This is _AnsiCat, a member of class AnsiStringBase.

38.1.1.7.1.1 AnsiStringBase::_AnsiCat Method (AnsiStringBase&, AnsiStringBase&)

Pascal

```
procedure _AnsiCat(var dst: AnsiStringBase; const src: AnsiStringBase&); static;
```

C++

```
static __fastcall void _AnsiCat(AnsiStringBase& dst, const AnsiStringBase& src);
```

Description

This is _AnsiCat, a member of class AnsiStringBase.

38.1.1.7.1.2 AnsiStringBase::_AnsiCat Method (AnsiStringBase&, AnsiStringBase&, AnsiStringBase&)

Pascal

```
procedure _AnsiCat(var dst: AnsiStringBase; const src1: AnsiStringBase&; const src2:
AnsiStringBase&); static;
```

C++

```
static __fastcall void _AnsiCat(AnsiStringBase& dst, const AnsiStringBase& src1, const
AnsiStringBase& src2);
```

Description

This is _AnsiCat, a member of class AnsiStringBase.

38.1.1.7.2 AnsiStringBase::_AnsiFromPWChar Method

Pascal

```
procedure _AnsiFromPWChar(var dst: AnsiStringBase; const src: wchar_t*; len: Integer;
codePage: Integer); static;
```

C++

```
static __fastcall void _AnsiFromPWChar(AnsiStringBase& dst, const wchar_t* src, int len, int
codePage);
```

Description

This is _AnsiFromPWChar, a member of class AnsiStringBase.

38.1.1.7.3 AnsiStringBase::AnsiCompare Method

Compares the System::AnsiStringBase::AnsiString to another specified string (case-sensitively).

Pascal

```
function AnsiCompare(const rhs: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiCompare(const AnsiStringBase& rhs) const;
```

Description

Compares this System::AnsiStringBase::AnsiString to rhs, with case sensitivity. The compare operation is controlled by the current Windows locale and is not necessarily the same as using the System::AnsiStringBase::AnsiString comparison operators. System::AnsiStringBase::AnsiCompare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

38.1.1.7.4 AnsiStringBase::AnsiCompareIC Method

Compares the System::AnsiStringBase::AnsiString to another specified string (case insensitively).

Pascal

```
function AnsiCompareIC(const rhs: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiCompareIC(const AnsiStringBase& rhs) const;
```

Description

System::AnsiStringBase::AnsiCompareIC compares this System::AnsiStringBase::AnsiString to rhs, without case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

38.1.1.7.5 AnsiStringBase::AnsiLastChar Method

Returns a pointer to the last full character in the System::AnsiStringBase::AnsiString.

Pascal

```
function AnsiLastChar: char*;
```

C++

```
__fastcall char* AnsiLastChar() const;
```

Description

Returns a pointer to the last full character in the System::AnsiStringBase::AnsiString. This function supports multibyte characters.

See Also

System::AnsiStringBase::ByteType (see page 3198)

38.1.1.7.6 AnsiStringBase::AnsiPos Method

Returns the index at which the specified substring begins.

Pascal

```
function AnsiPos(const subStr: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiPos(const AnsiStringBase& subStr) const;
```

Description

Returns the index in the System::AnsiStringBase::AnsiString at which the substring subStr begins, where 1 is the first character in the string, 2 is the second character, and so on. If the substring is not contained in the System::AnsiStringBase::AnsiString, Pos (see page 3209) returns 0. This member function is the same as Pos (see page 3209) but supports multibyte character strings.

Note: See ByteType (see page 3198) for more information about multibyte characters.

See Also

System::AnsiStringBase::ByteType (see page 3198)

38.1.1.7.7 AnsiStringBase::BytesOf Method

Pascal

```
function BytesOf: DynamicArray<System::Byte>;
```

C++

```
__fastcall DynamicArray<System::Byte> BytesOf() const;
```

Description

This is BytesOf, a member of class AnsiStringBase.

38.1.1.7.8 AnsiStringBase::ByteType Method

Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase::AnsiString.

Pascal

```
function ByteType(index: Integer): TStringMbcsByteType;
```

C++

```
__fastcall TStringMbcsByteType ByteType(int index) const;
```

Description

System::AnsiStringBase::ByteType indicates what kind of byte exists at the byte indicated by index (counting from 0) in the System::AnsiStringBase::AnsiString. These are the possible return values.

Constant	Meaning
mbSingleByte	Single byte character.
mbLeadByte	Lead byte of a multi-byte character.
mbTrailByte	Trailing byte of a multi-byte character.

Some locales, including all Western locales, do not use multi-byte characters. Under those locales, System::AnsiStringBase::ByteType always returns mbSingleByte.

38.1.1.7.9 AnsiStringBase::cat_vprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiString.

Pascal

```
function cat_vprintf(codePage: Integer; const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(int codePage, const char* format, va_list);
```

Description

Use cat_System::AnsiStringBase::cat_vprintf to append to the value of the System::AnsiStringBase::AnsiString given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the number of bytes appended to the System::AnsiStringBase::AnsiString.

See Also

System::AnsiStringBase::Format (see page 3202)

System::AnsiStringBase::cat_vprintf

System::AnsiStringBase::cat_printf

System::AnsiStringBase::cat_sprintf

arg

start

38.1.1.7.10 AnsiStringBase::CodePage Method

Gets code page of a System::AnsiStringBase (see page 3185).

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::AnsiStringBase::CodePage gets the code page of a System::AnsiStringBase (see page 3185) instance. If the instance is empty, it returns the system default code page, System::DefaultSystemCodePage (see page 3883).

See Also

System::AnsiStringBase::GetRec (see page 3204)

38.1.1.7.11 AnsiStringBase::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase CurrToStr(Currency value, int codePage);
```

Description

System::AnsiStringBase::CurrToStr converts the currency value given by value to its string representation. The conversion using the sffGeneral floating-point conversion rules. For further details, see System::AnsiStringBase::CurrToStrF (see page 3200).

See Also

System::AnsiStringBase::CurrToStrF (see page 3200)

38.1.1.7.12 AnsiStringBase::CurrToStrF Method

Converts a currency value to its string representation using the indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase CurrToStrF(Currency value, TStringFloatFormat format, int
digits, int codePage);
```

Description

System::AnsiStringBase::CurrToStrF converts the currency value given by value to its string representation using the indicated float format convention. A call to System::AnsiStringBase::CurrToStrF corresponds to a call to FloatToStrF (see page 3201) with an implied precision of 19 digits.

38.1.1.7.13 AnsiStringBase::Delete Method

Removes a specified number of characters from the string.

Pascal

```
function Delete(index: Integer; count: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Delete(int index, int count);
```

Description

System::AnsiStringBase::Delete modifies the System::AnsiStringBase::AnsiString to remove count characters from the string beginning with the character at index, where 1 is the index of the first character. It returns the resulting modified string (*this).

If index is larger than the length of the System::AnsiStringBase::AnsiString or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, System::AnsiStringBase::Delete removes the rest of the string. If count is less than 0, no characters are deleted.

38.1.1.7.14 AnsiStringBase::ElementSize Method

Gets element size of data in a System::AnsiStringBase (see page 3185).

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::AnsiStringBase::ElementSize gets the element size in bytes of the data in a System::AnsiStringBase (see page 3185) instance. If the instance is empty, this returns 1.

See Also

System::AnsiStringBase::data

System::AnsiStringBase::GetRec (see page 3204)

38.1.1.7.15 AnsiStringBase::EnsureAnsi Method

Pascal

```
function EnsureAnsi: AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& EnsureAnsi();
```

Description

This is EnsureAnsi, a member of class AnsiStringBase.

38.1.1.7.16 AnsiStringBase::FloatToStrF Method

Returns the string representation of a floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits, int codePage);
```

Description

The format parameter controls the format of the resulting string. The precision parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the digits parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).

sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3242) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3242) Format (see page 3202) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables.

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

38.1.1.7.17 AnsiStringBase::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer; codePage: Integer):
AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FmtLoadStr(int ident, const TVarRec * args, int size, int
codePage);
```

Description

System::AnsiStringBase::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the args array, as a parameter to Format (see page 3202). Ident is the string resource ID of the desired format string.

See Also

System::AnsiStringBase::LoadString (see page 3209)

System::AnsiStringBase::LoadStr (see page 3208)

38.1.1.7.18 AnsiStringBase::Format Method

System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase::AnsiString, format, and a series of array arguments, args.

Pascal

```
function Format(const format: AnsiStringBase&; const args: TVarRec *; size: Integer; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase Format(const AnsiStringBase& format, const TVarRec * args,
int size, int codePage);
```

Description

Use System::AnsiStringBase::Format to obtain an System::AnsiStringBase::AnsiString whose value is the result of applying an

array of arguments (args) to a given a format string (format). The last parameter (size) is the index of the last element in args (one less than the number of arguments).

Note that System::AnsiStringBase::Format is a static method. The returned value is a newly allocated System::AnsiStringBase::AnsiString, even if the method is called from an existing System::AnsiStringBase::AnsiString instance.

For information on how System::AnsiStringBase::Format assembles a final string from its format string and arguments, see System::AnsiStringBase::Format Strings.

See Also

System::AnsiStringBase::printf

System::AnsiStringBase::sprintf

38.1.1.7.19 AnsiStringBase::FormatFloat Method

Formats the floating-point value given by value using the format string given by format.

Pascal

```
function FormatFloat(const format: AnsiStringBase&; const value: long double&; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FormatFloat(const AnsiStringBase& format, const long double&
value, int codePage);
```

Description

The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the DecimalSeparator global variable, whose default value is specified in the Number Format (see page 3202) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the ThousandSeparator global variable, whose default value is specified in the Number Format (see page 3202) of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 3201) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3202) string-1234-12340.50

Format (see page 3202) string-	**1234**	**-1234**	**0.5**	**0**
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

38.1.1.7.20 GetRec Method

This is the overview for the GetRec method overload.

Overload List

	Name	**Description**
	AnsiStringBase::GetRec () (see page 3204)	Gets a struct describing a System::AnsiStringBase (see page 3185).
	AnsiStringBase::GetRec () (see page 3204)	Gets a struct describing a System::AnsiStringBase (see page 3185).

38.1.1.7.20.1 AnsiStringBase::GetRec Method ()

Gets a struct describing a System::AnsiStringBase (see page 3185).

Pascal

```
function GetRec: const StrRec&;
function GetRec: StrRec&;
```

C++

```
const StrRec& GetRec() const;
StrRec& GetRec();
```

Description

System::AnsiStringBase::GetRec gets a struct that describes the contents of a System::AnsiStringBase (see page 3185) instance. The struct has this format:

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

38.1.1.7.21 AnsiStringBase::Insert Method

System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase::AnsiString.

Pascal

```
function Insert(const str: AnsiStringBase&; index: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Insert(const AnsiStringBase& str, int index);
```

Description

System::AnsiStringBase::Insert inserts the string str into this System::AnsiStringBase::AnsiString beginning at the position index, where 1 is the first position in the string. It returns the resulting modified string (*this).

If index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to Length (see page 3208)(), turning the operation into an append.

If the str parameter is an empty string, System::AnsiStringBase::Insert does nothing.

System::AnsiStringBase::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer, string.

Note: The index parameter specifies the character, not the byte, when working with a multi-byte character system.

38.1.1.7.22 AnsiStringBase::IntToHex Method

Converts a number into a string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase IntToHex(int value, int digits, int codePage);
```

Description

Value is the number to convert. Digits indicates the minimum number of hexadecimal digits.

38.1.1.7.23 AnsiStringBase::IsDelimiter Method

Returns true if the specified character matches any character in the delimiters string.

Pascal

```
function IsDelimiter(const delimiters: AnsiStringBase&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const AnsiStringBase& delimiters, int index) const;
```

Description

System::AnsiStringBase::IsDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character contained in delimiters. If the byte is part of a multi-byte character or not in delimiters, System::AnsiStringBase::IsDelimiter returns false.

38.1.1.7.24 AnsiStringBase::IsEmpty Method

Returns true if the System::AnsiStringBase::AnsiString is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::AnsiStringBase::IsEmpty returns true if this System::AnsiStringBase::AnsiString is empty, false otherwise.

38.1.1.7.25 AnsiStringBase::IsLeadByte Method

Returns true if the specified byte is a lead byte.

Pascal

```
function IsLeadByte(index: Integer): bool;
```

C++

```
__fastcall bool IsLeadByte(int index) const;
```

Description

System::AnsiStringBase::IsLeadByte returns true if the byte indicated by index (counting from 0) is the lead byte in a multi-byte character. If the byte is a trailing byte, or not part of a multi-byte character, System::AnsiStringBase::IsLeadByte returns false.

38.1.1.7.26 AnsiStringBase::IsPathDelimiter Method

Returns true if the specified character is '\'.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::AnsiStringBase::IsPathDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character containing the path delimiter ('\' on Windows, '/' on Linux).

See Also

System::AnsiStringBase::ByteType (see page 3198)

38.1.1.7.27 AnsiStringBase::IsTrailByte Method

Returns true if the specified byte is of type mbTrailByte.

Pascal

```
function IsTrailByte(index: Integer): bool;
```

C++

```
__fastcall bool IsTrailByte(int index) const;
```

Description

System::AnsiStringBase::IsTrailByte returns true if the byte indicated by index (counting from 0) is the trailing byte in a multi-byte character. If the byte is a lead byte, or not part of a multi-byte character, System::AnsiStringBase::IsTrailByte returns false.

38.1.1.7.28 AnsiStringBase::LastChar Method

Pascal

```
function LastChar: char*;
```

C++

```
__fastcall char* LastChar() const;
```

Description

This is LastChar, a member of class AnsiStringBase.

38.1.1.7.29 AnsiStringBase::LastDelimiter Method

Returns the index of the rightmost character that matches any character in the delimiters string.

Pascal

```
function LastDelimiter(const delimiters: AnsiStringBase&): Integer;
```

C++

```
__fastcall int LastDelimiter(const AnsiStringBase& delimiters) const;
```

Description

System::AnsiStringBase::LastDelimiter returns the byte index in this System::AnsiStringBase::AnsiString of the rightmost whole character that matches any character in delimiters except NULL The System::AnsiStringBase::AnsiString may contain multibyte characters; delimiters must contain only single byte non-NULL characters.

See Also

System::AnsiStringBase::ByteType (see page 3198)

38.1.1.7.30 AnsiStringBase::Length Method

Returns the length of the System::AnsiStringBase::AnsiString.

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::AnsiStringBase::Length returns the number of bytes in this System::AnsiStringBase::AnsiString.

38.1.1.7.31 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	AnsiStringBase::LoadStr (HINSTANCE, int, int) (see page 3208)	Loads a string from the compiled resources of a specified module.
	AnsiStringBase::LoadStr (int, int) (see page 3208)	Loads a string from the compiled resources of a specified module.

38.1.1.7.31.1 AnsiStringBase::LoadStr Method (HINSTANCE, int, int)

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadStr(ident: Integer; codePage: Integer): AnsiStringBase; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer; codePage: Integer): AnsiStringBase;
static;
```

C++

```
static __fastcall AnsiStringBase LoadStr(int ident, int codePage);
static __fastcall AnsiStringBase LoadStr(HINSTANCE hInstance, int ident, int codePage);
```

Description

System::AnsiStringBase::LoadStr loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource. If hInstance is not given, System::AnsiStringBase::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::AnsiStringBase::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringBase::LoadString (see page 3209)

System::AnsiStringBase::FmtLoadStr (see page 3202)

38.1.1.7.32 AnsiStringBase::LoadString Method

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer; codePage: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& LoadString(HINSTANCE hInstance, int ident, int codePage);
```

Description

System::AnsiStringBase::LoadString loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::AnsiStringBase::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringBase::LoadStr (see page 3208)

System::AnsiStringBase::FmtLoadStr (see page 3202)

38.1.1.7.33 AnsiStringBase::LowerCase Method

Returns a lower-case version of System::AnsiStringBase::AnsiString.

Pascal

```
function LowerCase(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase LowerCase(int codePage) const;
```

Description

System::AnsiStringBase::LowerCase returns a new System::AnsiStringBase::AnsiString that contains all characters in this string, converted to lower case. The conversion uses the current locale . The original System::AnsiStringBase::AnsiString remains unchanged.

38.1.1.7.34 AnsiStringBase::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: AnsiStringBase&): Integer;
```

C++

```
__fastcall int Pos(const AnsiStringBase& subStr) const;
```

Description

System::AnsiStringBase::Pos returns the index in the System::AnsiStringBase::AnsiString at which the substring subStr begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::AnsiStringBase::AnsiString, System::AnsiStringBase::Pos returns 0.

38.1.1.7.35 AnsiStringBase::RefCount Method

Gets reference count of System::AnsiStringBase (see page 3185).

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::AnsiStringBase::RefCount gets the reference count of a System::AnsiStringBase (see page 3185) instance. If the instance is empty, it returns 0.

See Also

System::AnsiStringBase::GetRec (see page 3204)

38.1.1.7.36 AnsiStringBase::SetLength Method

Changes the length of the System::AnsiStringBase::AnsiString.

Pascal

```
function SetLength(newLength: Integer; codePage: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& SetLength(int newLength, int codePage);
```

Description

System::AnsiStringBase::SetLength changes the string's length to newLength, truncating if necessary.

System::AnsiStringBase::SetLength reallocates the underlying array of characters that stores the value of the System::AnsiStringBase::AnsiString. Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::AnsiStringBase::AnsiString, an EOutOfMemory exception is thrown. Following a call to System::AnsiStringBase::SetLength, the System::AnsiStringBase::AnsiString is guaranteed not to be a shared reference.

System::AnsiStringBase::SetLength returns the modified System::AnsiStringBase::AnsiString object (*this)

38.1.1.7.37 AnsiStringBase::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: char; count: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase StringOfChar(char ch, int count);
```

Description

System::AnsiStringBase::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::AnsiStringBase::AnsiString s = System::AnsiStringBase::AnsiString::System::AnsiStringBase::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

38.1.1.7.38 AnsiStringBase::SubString Method

Returns a specified substring of the System::AnsiStringBase::AnsiString.

Pascal

```
function SubString(index: Integer; count: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase SubString(int index, int count) const;
```

Description

System::AnsiStringBase::SubString returns a new System::AnsiStringBase::AnsiString that is a substring of this System::AnsiStringBase::AnsiString. The substring contains count characters beginning at index.

38.1.1.7.39 AnsiStringBase::swap Method

Pascal

```
function swap(var other: AnsiStringBase): AnsiStringBase&;
```

C++

```
AnsiStringBase& swap(AnsiStringBase& other);
```

Description

This is swap, a member of class AnsiStringBase.

38.1.1.7.40 AnsiStringBase::ThrowIfOutOfRange Method

Pascal

```
procedure ThrowIfOutOfRange(idx: Integer); cdecl;
```

C++

```
__cdecl void ThrowIfOutOfRange(int idx) const;
```

Description

This is ThrowIfOutOfRange, a member of class AnsiStringBase.

38.1.1.7.41 AnsiStringBase::ToDouble Method

Converts the string to a floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::AnsiStringBase::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The DecimalSeparator global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

38.1.1.7.42 AnsiStringBase::ToInt Method

Converts a string to an integer, throwing an exception if the string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::AnsiStringBase::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, an EConvertError exception is thrown.

See Also

System::AnsiStringBase::ToIntDef (see page 3212)

38.1.1.7.43 AnsiStringBase::ToIntDef Method

Converts the string to an integer, returning a default value if the string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::AnsiStringBase::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by defaultValue is returned.

See Also

System::AnsiStringBase::ToInt (see page 3212)

38.1.1.7.44 AnsiStringBase::Trim Method

Returns a new System::AnsiStringBase::AnsiString without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase Trim(int codePage) const;
```

Description

System::AnsiStringBase::Trim returns a new System::AnsiStringBase::AnsiString removing leading and trailing spaces and control characters. Use System::AnsiStringBase::Trim to remove blank space before and after the first printing character.

38.1.1.7.45 AnsiStringBase::TrimLeft Method

Returns a new System::AnsiStringBase::AnsiString without leading spaces or control characters.

Pascal

```
function TrimLeft(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase TrimLeft(int codePage) const;
```

Description

System::AnsiStringBase::TrimLeft returns a new System::AnsiStringBase::AnsiString removing leading spaces and control characters. Use System::AnsiStringBase::TrimLeft to remove blank space before the first printing character.

38.1.1.7.46 AnsiStringBase::TrimRight Method

Returns a new System::AnsiStringBase::AnsiString without trailing spaces or control characters.

Pascal

```
function TrimRight(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase TrimRight(int codePage) const;
```

Description

System::AnsiStringBase::TrimRight returns a new System::AnsiStringBase::AnsiString removing trailing spaces and control characters. Use the System::AnsiStringBase::TrimRight function to remove blank space after the last printing character.

38.1.1.7.47 AnsiStringBase::Unique Method

Makes the System::AnsiStringBase::AnsiString unique.

Pascal

```
function Unique: AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::AnsiStringBase::Unique sets the reference count on the string for the System::AnsiStringBase::AnsiString object to 1 so that it is the only object referencing that string.

38.1.1.7.48 AnsiStringBase::UpperCase Method

Returns an upper-case version of this System::AnsiStringBase::AnsiString.

Pascal

```
function UpperCase(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase UpperCase(int codePage) const;
```

Description

System::AnsiStringBase::UpperCase returns a new System::AnsiStringBase::AnsiString that contains all characters in this string, converted to upper case. The conversion uses the current locale . The original System::AnsiStringBase::AnsiString remains unchanged.

38.1.1.7.49 AnsiStringBase::vprintf Method

Sets the value of the System::AnsiStringBase::AnsiString given a format string and its arguments.

Pascal

```
function vprintf(codePage: Integer; const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(int codePage, const char* format, va_list);
```

Description

Use System::AnsiStringBase::vprintf to set to the value of the System::AnsiStringBase::AnsiString given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the length of the final formatted string.

See Also

System::AnsiStringBase::Format (see page 3202)

System::AnsiStringBase::sprintf

System::AnsiStringBase::printf

System::AnsiStringBase::cat_vprintf (see page 3199)

arg

start

38.1.1.7.50 AnsiStringBase::WideChar Method

Converts the System::AnsiStringBase::AnsiString to a wide-character array.

Pascal

```
function WideChar(var dest: wchar_t; destSize: Integer; codePage: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* WideChar(wchar_t* dest, int destSize, int codePage) const;
```

Description

System::AnsiStringBase::WideChar converts the System::AnsiStringBase::AnsiString to an array of wide characters, and returns the resulting wide-character array.

dest is a buffer the caller allocates to hold the return value. Use the System::AnsiStringBase::WideCharBufSize (see page 3214) method to determine the size needed to hold the entire string value.

destSize is the size, in bytes, of dest.

System::AnsiStringBase::WideChar returns a pointer to the wide-character array that is written to dest.

See Also

System::AnsiStringBase::WideCharBufSize (see page 3214)

38.1.1.7.51 AnsiStringBase::WideCharBufSize Method

Returns the size of the buffer required by the WideChar (see page 3214) function.

Pascal

```
function WideCharBufSize(codePage: Integer): Integer;
```

C++

```
__fastcall int WideCharBufSize(int codePage) const;
```

Description

System::AnsiStringBase::WideCharBufSize returns an integer value indicating the size of the buffer required by the WideChar (see page 3214) method in order to convert this string.

See Also

System::AnsiStringBase::WideChar (see page 3214)

38.1.1.8 AnsiStringBase Operators

The operators of the AnsiStringBase class are listed here.

Protected Operators

	Name	Description
	!= (see page 3215)	Compares two strings.
	< (see page 3216)	Compares two strings.
	<= (see page 3217)	Compares two strings.
	= (see page 3217)	Assigns the string rhs to this System::AnsiStringBase::AnsiString.
	== (see page 3218)	Compares the System::AnsiStringBase::AnsiString with a specified string, returning true if they are the same.
	> (see page 3218)	Compares two strings.
	>= (see page 3218)	Compares two strings.

Public Operators

	Name	Description
	[] (see page 3216)	This is the overview for the [] operator overload.
	+ (see page 3216)	Concatenates two System::AnsiStringBase::AnsiString values.

38.1.1.8.1 AnsiStringBase::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator !=(const AnsiStringBase& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::!= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.1.8.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	AnsiStringBase::[] (int) (see page 3216)	Returns the character at a specified index in the string.
	AnsiStringBase::[] (int) (see page 3216)	Returns the character at a specified index in the string.

38.1.1.8.2.1 AnsiStringBase::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall char operator [](const int idx) const;
__fastcall char& operator [](const int idx);
```

Description

This System::AnsiStringBase (see page 3185)::[] returns the byte in the string at index value idx. The System::AnsiStringBase (see page 3185)::[] assumes a base index of 1.

Note: If the string contains multi-byte characters, the System::AnsiStringBase (see page 3185)::[] may return a byte value that represents the wrong character and/or a partial character. For character-level access, use the SubString (see page 3211) method.

See Also

System::AnsiStringBase::substring

38.1.1.8.3 AnsiStringBase::+ Operator

Concatenates two System::AnsiStringBase::AnsiString values.

Pascal

```
operator +(const rhs: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringBase operator +(const AnsiStringBase& rhs) const;
```

Description

The friend function returns an System::AnsiStringBase::AnsiString that is the concatenation of strings lhs and rhs.

The method returns an System::AnsiStringBase::AnsiString that is the concatenation of this System::AnsiStringBase::AnsiString and rhs.

38.1.1.8.4 AnsiStringBase::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator <(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is less than rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::< performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.1.8.5 AnsiStringBase::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator <=(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is less than or equal to rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::<= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.1.8.6 AnsiStringBase::= Operator

Assigns the string rhs to this System::AnsiStringBase::AnsiString.

Pascal

```
operator =(const rhs: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringBase& operator =(const AnsiStringBase& rhs);
```

Description

Use this System::AnsiStringBase (see page 3185)::= to assign the value of the System::AnsiStringBase::AnsiString from another System::AnsiStringBase::AnsiString.

38.1.1.8.7 AnsiStringBase::== Operator

Compares the System::AnsiStringBase::AnsiString with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator ==(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is the same as rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::== performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.1.8.8 AnsiStringBase::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator >(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is greater than rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::> performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.1.8.9 AnsiStringBase::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator >=(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is greater than or equal to rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3185)::>= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3196) or AnsiCompareIC (see page 3197) methods.

See Also

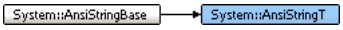
System::AnsiStringBase::AnsiCompare (see page 3196)

System::AnsiStringBase::AnsiCompareIC (see page 3197)

38.1.2 AnsiStringT Template

System::AnsiStringT is the C++ analog for the Delphi long string type.

Class Hierarchy



Pascal

```
AnsiStringT = class(AnsiStringBase);
```

C++

```
template <unsigned short CP>
class AnsiStringT : public AnsiStringBase;
```

File

dstring.h

Description

System::AnsiStringBase (see page 3185) exists to provide a place for methods/members that are common to all instantiations of the template class AnsiStringT<CodePage (see page 3230)>. AnsiString itself is a typedef that is equivalent to AnsiStringT<0>. You should never deal directly with System::AnsiStringBase (see page 3185).

System::AnsiStringT is a template class that you use when handling a Delphi AnsiString that has a code page specified with it.

Delphi uses several string types. One important long string type is commonly known as System::AnsiString. Support for this type includes the following features:

- strings can be as large as available memory
- efficient use of memory is ensured through shared references
- routines and operators evaluate strings based on the current locale

System::AnsiStringT is a template class that holds code for each specific code page. Despite its name, System::AnsiStringT is not restricted to the ANSI character set, and can use any character set supported by the current locale definition, including multi-byte or Unicode character sets.

System::AnsiStringT variables that have not been assigned an initial value contain a zero-length string.

To use the C++ streaming operators (**<<** and **>>**) with System::AnsiStringBase (see page 3185), you must `iostream` before including `system.hpp` and use `AnsiStringT::c_str` (see page 3229)() to return the internal string representation. The following example demonstrates the usage of AnsiString with stream operators:

```
#include <iostream>
#include <system.hpp>
int main() {
  AnsiString HelloStr = "hello";
  std::cout << HelloStr.c_str();
}
```

Note: Delphi also supports System::AnsiString, but implements it as a primitive type rather than a class. By default, variables declared as type String are System::UnicodeString (see page 3393).

See Also

How to Handle Delphi AnsiString Code Page Specification in C++

System::WideString (see page 3476)

System::AnsiStringBase::[] (see page 3216)

System::AnsiStringT::= (see page 3241)

System::AnsiStringT::== (see page 3241)

System::AnsiStringT::!= (see page 3240)

System::AnsiStringT::> (see page 3241)

System::AnsiStringT::>= (see page 3241)

System::AnsiStringT::< (see page 3240)

System::AnsiStringT::<= (see page 3241)

System::AnsiStringT::+ (see page 3240)

System::AnsiStringT::+= (see page 3240)

38.1.2.1 AnsiStringT Members

The following tables list the members exposed by AnsiStringT.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3193)	This is record System::AnsiStringBase::TStringFloatFormat.
	TStringMbcsByteType (see page 3193)	This is record System::AnsiStringBase::TStringMbcsByteType.

Structures

	Name	Description
	StrRec (see page 3192)	This is record System::AnsiStringBase::StrRec.

Protected Data Members

	Name	Description
	Data (see page 3194)	This is Data, a member of class AnsiStringBase.

Protected Methods

	Name	Description
	_AnsiCat (see page 3195)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3196)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiStringBase (see page 3189)	This is the overview for the AnsiStringBase constructor overload.
	GetRec (see page 3204)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3211)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Public Methods

	Name	Description
	~AnsiStringBase (see page 3189)	This is ~AnsiStringBase, a member of class AnsiStringBase.
	AnsiCompare (see page 3229)	This is AnsiCompare, a member of class AnsiStringT.
	AnsiCompareIC (see page 3229)	This is AnsiCompareIC, a member of class AnsiStringT.

	AnsiLastChar (see page 3197)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiString.
	AnsiPos (see page 3229)	This is AnsiPos, a member of class AnsiStringT.
	AnsiStringT (see page 3222)	This is the overview for the AnsiStringT constructor overload.
	BytesOf (see page 3198)	This is BytesOf, a member of class AnsiStringBase.
	ByteType (see page 3198)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase::AnsiString.
	c_str (see page 3229)	Returns the System::AnsiStringT::AnsiString converted to a null-terminated character array.
	cat_printf (see page 3230)	This is cat_printf, a member of class AnsiStringT.
	cat_sprintf (see page 3230)	This is cat_sprintf, a member of class AnsiStringT.
	cat_vprintf (see page 3230)	This is cat_vprintf, a member of class AnsiStringT.
	CodePage (see page 3230)	Gets code page of a System::AnsiStringBase.
	CurrToStr (see page 3231)	This is CurrToStr, a member of class AnsiStringT.
	CurrToStrF (see page 3231)	This is CurrToStrF, a member of class AnsiStringT.
	data (see page 3231)	Returns a pointer to the underlying string System::AnsiStringT::data.
	Delete (see page 3232)	Removes a specified number of characters from the string.
	ElementSize (see page 3232)	Gets element size of data in a System::AnsiStringBase.
	EnsureAnsi (see page 3232)	This is EnsureAnsi, a member of class AnsiStringT.
	FloatToStrF (see page 3233)	This is FloatToStrF, a member of class AnsiStringT.
	FmtLoadStr (see page 3233)	This is FmtLoadStr, a member of class AnsiStringT.
	Format (see page 3233)	This is Format, a member of class AnsiStringT.
	FormatFloat (see page 3233)	This is FormatFloat, a member of class AnsiStringT.
	Insert (see page 3234)	This is Insert, a member of class AnsiStringT.
	IntToHex (see page 3234)	This is IntToHex, a member of class AnsiStringT.
	IsDelimiter (see page 3234)	This is IsDelimiter, a member of class AnsiStringT.
	IsEmpty (see page 3206)	Returns true if the System::AnsiStringBase::AnsiString is empty.
	IsLeadByte (see page 3206)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3206)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3207)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3207)	This is LastChar, a member of class AnsiStringBase.
	LastDelimiter (see page 3234)	This is LastDelimiter, a member of class AnsiStringT.
	Length (see page 3208)	Returns the length of the System::AnsiStringBase::AnsiString.
	LoadStr (see page 3208)	This is the overview for the LoadStr method overload.
	LoadStr (see page 3234)	This is the overview for the LoadStr method overload.
	LoadString (see page 3235)	This is LoadString, a member of class AnsiStringT.
	LowerCase (see page 3235)	This is LowerCase, a member of class AnsiStringT.
	Pos (see page 3236)	This is Pos, a member of class AnsiStringT.
	printf (see page 3236)	This is printf, a member of class AnsiStringT.
	RefCount (see page 3236)	Gets reference count of System::AnsiStringBase.
	SetLength (see page 3236)	This is SetLength, a member of class AnsiStringT.
	sprintf (see page 3237)	This is sprintf, a member of class AnsiStringT.
	StringOfChar (see page 3237)	Returns a string with the specified number of characters.
	SubString (see page 3237)	Returns a specified substring of the System::AnsiStringBase::AnsiString.
	swap (see page 3237)	This is swap, a member of class AnsiStringT.
	ToDouble (see page 3211)	Converts the string to a floating-point value.
	ToInt (see page 3212)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3212)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3238)	This is Trim, a member of class AnsiStringT.
	TrimLeft (see page 3238)	This is TrimLeft, a member of class AnsiStringT.
	TrimRight (see page 3238)	This is TrimRight, a member of class AnsiStringT.
	Unique (see page 3238)	Makes the System::AnsiStringBase::AnsiString unique.
	UpperCase (see page 3238)	This is UpperCase, a member of class AnsiStringT.
	vprintf (see page 3239)	This is vprintf, a member of class AnsiStringT.
	WideChar (see page 3239)	This is WideChar, a member of class AnsiStringT.

	WideCharBufSize (see page 3239)	This is WideCharBufSize, a member of class AnsiStringT.

Public Operators

	Name	Description
	!= (see page 3240)	This is !=, a member of class AnsiStringT.
	[] (see page 3216)	This is the overview for the [] operator overload.
	+ (see page 3240)	This is +, a member of class AnsiStringT.
	+= (see page 3240)	This is +=, a member of class AnsiStringT.
	< (see page 3240)	This is <, a member of class AnsiStringT.
	<= (see page 3241)	This is <=, a member of class AnsiStringT.
	= (see page 3241)	This is =, a member of class AnsiStringT.
	== (see page 3241)	This is ==, a member of class AnsiStringT.
	> (see page 3241)	This is >, a member of class AnsiStringT.
	>= (see page 3241)	This is >=, a member of class AnsiStringT.

38.1.2.2 AnsiStringT Constructor

This is the overview for the AnsiStringT constructor overload.

Overload List

	Name	Description
	AnsiStringT::AnsiStringT () (see page 3223)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (__int64) (see page 3224)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (AnsiStringBase&) (see page 3223)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (AnsiStringT&) (see page 3223)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (AnsiStringT<OTHER_CP>&) (see page 3224)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (char) (see page 3224)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (char*) (see page 3225)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (char*, int) (see page 3225)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (char16_t*, int) (see page 3225)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (char32_t*, int) (see page 3225)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (double) (see page 3226)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (int) (see page 3226)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (long) (see page 3226)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (short) (see page 3226)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (UnicodeString &) (see page 3224)	This is AnsiStringT, a member of class AnsiStringT.

	AnsiStringT::AnsiStringT (unsigned __int64) (see page 3226)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (unsigned int) (see page 3227)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (unsigned long) (see page 3227)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (unsigned short) (see page 3227)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (wchar_t) (see page 3227)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (wchar_t*, int) (see page 3228)	This is AnsiStringT, a member of class AnsiStringT.
	AnsiStringT::AnsiStringT (WideString &) (see page 3224)	This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.1 AnsiStringT::AnsiStringT Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall AnsiStringT();
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.2 AnsiStringT::AnsiStringT Constructor (AnsiStringBase&)

Pascal

```
constructor Create(const src: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringT(const AnsiStringBase& src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.3 AnsiStringT::AnsiStringT Constructor (AnsiStringT&)

Pascal

```
constructor Create(const src: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT(const AnsiStringT& src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.4 AnsiStringT::AnsiStringT Constructor (AnsiStringT<OTHER_CP>&)

Pascal

```
constructor Create(const src: AnsiStringT<OTHER_CP>&);
```

C++

```
template <unsigned short OTHER_CP> AnsiStringT<CP>(const AnsiStringT<OTHER_CP>& src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.5 AnsiStringT::AnsiStringT Constructor (UnicodeString &)

Pascal

```
constructor Create(const src: UnicodeString &);
```

C++

```
__fastcall AnsiStringT(const UnicodeString & src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.6 AnsiStringT::AnsiStringT Constructor (WideString &)

Pascal

```
constructor Create(const src: WideString &);
```

C++

```
__fastcall AnsiStringT(const WideString & src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.7 AnsiStringT::AnsiStringT Constructor (__int64)

Pascal

```
constructor Create(src: __int64);
```

C++

```
__fastcall AnsiStringT(__int64 src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.8 AnsiStringT::AnsiStringT Constructor (char)

Pascal

```
constructor Create(src: char);
```

C++

```
__fastcall AnsiStringT(char src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.9 AnsiStringT::AnsiStringT Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall AnsiStringT(const char* src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.10 AnsiStringT::AnsiStringT Constructor (char*, int)

Pascal

```
constructor Create(const src: char*; byteLen: Integer);
```

C++

```
__fastcall AnsiStringT(const char* src, int byteLen);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.11 AnsiStringT::AnsiStringT Constructor (char16_t*, int)

Pascal

```
constructor Create(const src: char16_t*; numChar16: Integer = -1);
```

C++

```
__fastcall AnsiStringT(const char16_t* src, int numChar16 = -1);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.12 AnsiStringT::AnsiStringT Constructor (char32_t*, int)

Pascal

```
constructor Create(const src: char32_t*; numChar32: Integer = -1);
```

C++

```
__fastcall AnsiStringT(const char32_t* src, int numChar32 = -1);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.13 AnsiStringT::AnsiStringT Constructor (double)

Pascal

```
constructor Create(src: double);
```

C++

```
__fastcall AnsiStringT(double src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.14 AnsiStringT::AnsiStringT Constructor (int)

Pascal

```
constructor Create(src: Integer);
```

C++

```
__fastcall AnsiStringT(int src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.15 AnsiStringT::AnsiStringT Constructor (long)

Pascal

```
constructor Create(src: long);
```

C++

```
__fastcall AnsiStringT(long src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.16 AnsiStringT::AnsiStringT Constructor (short)

Pascal

```
constructor Create(src: short);
```

C++

```
__fastcall AnsiStringT(short src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.17 AnsiStringT::AnsiStringT Constructor (unsigned __int64)

Pascal

```
constructor Create(src: unsigned __int64);
```

C++

```
__fastcall AnsiStringT(unsigned __int64 src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.18 AnsiStringT::AnsiStringT Constructor (unsigned int)

Pascal

```
constructor Create(src: unsigned int);
```

C++

```
__fastcall AnsiStringT(unsigned int src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.19 AnsiStringT::AnsiStringT Constructor (unsigned long)

Pascal

```
constructor Create(src: unsigned long);
```

C++

```
__fastcall AnsiStringT(unsigned long src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.20 AnsiStringT::AnsiStringT Constructor (unsigned short)

Pascal

```
constructor Create(src: unsigned short);
```

C++

```
__fastcall AnsiStringT(unsigned short src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.21 AnsiStringT::AnsiStringT Constructor (wchar_t)

Pascal

```
constructor Create(src: wchar_t);
```

C++

```
__fastcall AnsiStringT(wchar_t src);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.2.22 AnsiStringT::AnsiStringT Constructor (wchar_t*, int)

Pascal

```
constructor Create(const src: wchar_t*; numwchar: Integer = -1);
```

C++

```
__fastcall AnsiStringT(const wchar_t* src, int numwchar = -1);
```

Description

This is AnsiStringT, a member of class AnsiStringT.

38.1.2.3 AnsiStringT Methods

The methods of the AnsiStringT class are listed here.

Public Methods

	Name	Description
	AnsiCompare (see page 3229)	This is AnsiCompare, a member of class AnsiStringT.
	AnsiCompareIC (see page 3229)	This is AnsiCompareIC, a member of class AnsiStringT.
	AnsiPos (see page 3229)	This is AnsiPos, a member of class AnsiStringT.
	c_str (see page 3229)	Returns the System::AnsiStringT::AnsiString converted to a null-terminated character array.
	cat_printf (see page 3230)	This is cat_printf, a member of class AnsiStringT.
	cat_sprintf (see page 3230)	This is cat_sprintf, a member of class AnsiStringT.
	cat_vprintf (see page 3230)	This is cat_vprintf, a member of class AnsiStringT.
	CodePage (see page 3230)	Gets code page of a System::AnsiStringBase.
	CurrToStr (see page 3231)	This is CurrToStr, a member of class AnsiStringT.
	CurrToStrF (see page 3231)	This is CurrToStrF, a member of class AnsiStringT.
	data (see page 3231)	Returns a pointer to the underlying string System::AnsiStringT::data.
	Delete (see page 3232)	Removes a specified number of characters from the string.
	ElementSize (see page 3232)	Gets element size of data in a System::AnsiStringBase.
	EnsureAnsi (see page 3232)	This is EnsureAnsi, a member of class AnsiStringT.
	FloatToStrF (see page 3233)	This is FloatToStrF, a member of class AnsiStringT.
	FmtLoadStr (see page 3233)	This is FmtLoadStr, a member of class AnsiStringT.
	Format (see page 3233)	This is Format, a member of class AnsiStringT.
	FormatFloat (see page 3233)	This is FormatFloat, a member of class AnsiStringT.
	Insert (see page 3234)	This is Insert, a member of class AnsiStringT.
	IntToHex (see page 3234)	This is IntToHex, a member of class AnsiStringT.
	IsDelimiter (see page 3234)	This is IsDelimiter, a member of class AnsiStringT.
	LastDelimiter (see page 3234)	This is LastDelimiter, a member of class AnsiStringT.
	LoadStr (see page 3234)	This is the overview for the LoadStr method overload.
	LoadString (see page 3235)	This is LoadString, a member of class AnsiStringT.
	LowerCase (see page 3235)	This is LowerCase, a member of class AnsiStringT.
	Pos (see page 3236)	This is Pos, a member of class AnsiStringT.
	printf (see page 3236)	This is printf, a member of class AnsiStringT.
	RefCount (see page 3236)	Gets reference count of System::AnsiStringBase.
	SetLength (see page 3236)	This is SetLength, a member of class AnsiStringT.
	sprintf (see page 3237)	This is sprintf, a member of class AnsiStringT.
	StringOfChar (see page 3237)	Returns a string with the specified number of characters.
	SubString (see page 3237)	Returns a specified substring of the System::AnsiStringBase::AnsiString.
	swap (see page 3237)	This is swap, a member of class AnsiStringT.
	Trim (see page 3238)	This is Trim, a member of class AnsiStringT.

	TrimLeft (see page 3238)	This is TrimLeft, a member of class AnsiStringT.
	TrimRight (see page 3238)	This is TrimRight, a member of class AnsiStringT.
	Unique (see page 3238)	Makes the System::AnsiStringBase::AnsiString unique.
	UpperCase (see page 3238)	This is UpperCase, a member of class AnsiStringT.
	vprintf (see page 3239)	This is vprintf, a member of class AnsiStringT.
	WideChar (see page 3239)	This is WideChar, a member of class AnsiStringT.
	WideCharBufSize (see page 3239)	This is WideCharBufSize, a member of class AnsiStringT.

38.1.2.3.1 AnsiStringT::AnsiCompare Method

Pascal

```
function AnsiCompare(const rhs: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiCompare(const AnsiStringT& rhs) const;
```

Description

This is AnsiCompare, a member of class AnsiStringT.

38.1.2.3.2 AnsiStringT::AnsiCompareIC Method

Pascal

```
function AnsiCompareIC(const rhs: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiCompareIC(const AnsiStringT& rhs) const;
```

Description

This is AnsiCompareIC, a member of class AnsiStringT.

38.1.2.3.3 AnsiStringT::AnsiPos Method

Pascal

```
function AnsiPos(const subStr: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiPos(const AnsiStringT& subStr) const;
```

Description

This is AnsiPos, a member of class AnsiStringT.

38.1.2.3.4 AnsiStringT::c_str Method

Returns the System::AnsiStringT::AnsiString converted to a null-terminated character array.

Pascal

```
function c_str: char*;
```

C++

```
__fastcall char* c_str() const;
```

Description

System::AnsiStringT::c_str returns a pointer to a null-terminated character array containing the same characters as the

System::AnsiStringT::AnsiString. If the AnsiString is unassigned, c_str returns an empty string ("").

Usually, the value returned by System::AnsiStringT::c_str points to the internal character array referenced by the data (see page 3231) property. This pointer is valid until the string is next modified (for example when the SetLength (see page 3236) method is called or the AnsiString goes out of scope). However, if the internal array is NULL, c_str() returns a pointer to a string literal.

The System::AnsiStringT::c_str method is intended primarily for reading the value of the AnsiString. To modify the string's value, use the [] operator or System::AnsiStringT::AnsiString methods such as Insert (see page 3234) and Delete (see page 3232).

38.1.2.3.5 AnsiStringT::cat_printf Method

Pascal

```
function cat_printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const char* format, ...);
```

Description

This is cat_printf, a member of class AnsiStringT.

38.1.2.3.6 AnsiStringT::cat_sprintf Method

Pascal

```
function cat_sprintf(const format: char*; : ...): AnsiStringT&; cdecl;
```

C++

```
__cdecl AnsiStringT& cat_sprintf(const char* format, ...);
```

Description

This is cat_sprintf, a member of class AnsiStringT.

38.1.2.3.7 AnsiStringT::cat_vprintf Method

Pascal

```
function cat_vprintf(const format: char*; list: va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const char* format, va_list list);
```

Description

This is cat_vprintf, a member of class AnsiStringT.

38.1.2.3.8 AnsiStringT::CodePage Method

Gets code page of a System::AnsiStringBase.

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::AnsiStringBase::CodePage gets the code page of a System::AnsiStringBase instance. If the instance is empty, it returns the system default code page, System::DefaultSystemCodePage.

See Also

System::AnsiStringBase::GetRec (see page 3204)

38.1.2.3.9 AnsiStringT::CurrToStr Method

Pascal

```
function CurrToStr(value: Currency): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT CurrToStr(Currency value);
```

Description

This is CurrToStr, a member of class AnsiStringT.

38.1.2.3.10 AnsiStringT::CurrToStrF Method

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer):
AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT CurrToStrF(Currency value, TStringFloatFormat format, int
digits);
```

Description

This is CurrToStrF, a member of class AnsiStringT.

38.1.2.3.11 AnsiStringT::data Method

Returns a pointer to the underlying string System::AnsiStringT::data.

Pascal

```
function data: const void*;
```

C++

```
__fastcall const void* data() const;
```

Description

System::AnsiStringT::data returns a pointer to a null-terminated character array that the System::AnsiStringT::AnsiString uses to store its value. This is the same value as the one returned by the System::AnsiStringT::c_str (see page 3229) method, except that System::AnsiStringT::data returns a NULL pointer when the value is an empty string.

Note: Because System::AnsiStringT::data returns NULL instead of an empty string, it is useful for C++ functions that expect either a valid (non-empty) char* or NULL. The following example assumes that TCHAR mapping is not set to wchar_t:

```
AnsiString x;
    ShellExecute(0, static_cast<const char*>(x.data()), "File.txt", NULL, NULL, SW_SHOW);
```

If TCHAR mapping is set to wchar_t, ShellExecute becomes the wide version, ShellExecuteW. If you want to use the Ansi version of ShellExecute when TCHAR mapping is set to wchar_t, you should specify ShellExecuteA.

See Also

System::AnsiStringT::c_str (see page 3229)

TCHAR Mapping

38.1.2.3.12 AnsiStringT::Delete Method

Removes a specified number of characters from the string.

Pascal

```
function Delete(index: Integer; count: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Delete(int index, int count);
```

Description

System::AnsiStringBase::Delete modifies the System::AnsiStringBase::AnsiString to remove count characters from the string beginning with the character at index, where 1 is the index of the first character. It returns the resulting modified string (*this).

If index is larger than the length of the System::AnsiStringBase::AnsiString or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, System::AnsiStringBase::Delete removes the rest of the string. If count is less than 0, no characters are deleted.

38.1.2.3.13 AnsiStringT::ElementSize Method

Gets element size of data in a System::AnsiStringBase.

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::AnsiStringBase::ElementSize gets the element size in bytes of the data in a System::AnsiStringBase instance. If the instance is empty, this returns 1.

See Also

System::AnsiStringBase::data

System::AnsiStringBase::GetRec (see page 3204)

38.1.2.3.14 AnsiStringT::EnsureAnsi Method

Pascal

```
function EnsureAnsi: AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& EnsureAnsi();
```

Description

This is EnsureAnsi, a member of class AnsiStringT.

38.1.2.3.15 AnsiStringT::FloatToStrF Method

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

This is FloatToStrF, a member of class AnsiStringT.

38.1.2.3.16 AnsiStringT::FmtLoadStr Method

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT FmtLoadStr(int ident, const TVarRec * args, int size);
```

Description

This is FmtLoadStr, a member of class AnsiStringT.

38.1.2.3.17 AnsiStringT::Format Method

Pascal

```
function Format(const format: AnsiStringT&; const args: TVarRec *; size: Integer):
AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT Format(const AnsiStringT& format, const TVarRec * args, int
size);
```

Description

This is Format, a member of class AnsiStringT.

38.1.2.3.18 AnsiStringT::FormatFloat Method

Pascal

```
function FormatFloat(const format: AnsiStringT&; const value: long double&): AnsiStringT;
static;
```

C++

```
static __fastcall AnsiStringT FormatFloat(const AnsiStringT& format, const long double& value);
```

Description

This is FormatFloat, a member of class AnsiStringT.

38.1.2.3.19 AnsiStringT::Insert Method

Pascal

```
function Insert(const str: AnsiStringT&; index: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Insert(const AnsiStringT& str, int index);
```

Description

This is Insert, a member of class AnsiStringT.

38.1.2.3.20 AnsiStringT::IntToHex Method

Pascal

```
function IntToHex(value: Integer; digits: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT IntToHex(int value, int digits);
```

Description

This is IntToHex, a member of class AnsiStringT.

38.1.2.3.21 AnsiStringT::IsDelimiter Method

Pascal

```
function IsDelimiter(const delimiters: AnsiStringT&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const AnsiStringT& delimiters, int index) const;
```

Description

This is IsDelimiter, a member of class AnsiStringT.

38.1.2.3.22 AnsiStringT::LastDelimiter Method

Pascal

```
function LastDelimiter(const delimiters: AnsiStringT&): Integer;
```

C++

```
__fastcall int LastDelimiter(const AnsiStringT& delimiters) const;
```

Description

This is LastDelimiter, a member of class AnsiStringT.

38.1.2.3.23 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	AnsiStringT::LoadStr (HINSTANCE, int) (see page 3235)	This is LoadStr, a member of class AnsiStringT.
	AnsiStringT::LoadStr (int) (see page 3235)	This is LoadStr, a member of class AnsiStringT.

38.1.2.3.23.1 AnsiStringT::LoadStr Method (HINSTANCE, int)

Pascal

```
function LoadStr(hInstance: HINSTANCE; ident: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT LoadStr(HINSTANCE hInstance, int ident);
```

Description

This is LoadStr, a member of class AnsiStringT.

38.1.2.3.23.2 AnsiStringT::LoadStr Method (int)

Pascal

```
function LoadStr(ident: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT LoadStr(int ident);
```

Description

This is LoadStr, a member of class AnsiStringT.

38.1.2.3.24 AnsiStringT::LoadString Method

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& LoadString(HINSTANCE hInstance, int ident);
```

Description

This is LoadString, a member of class AnsiStringT.

38.1.2.3.25 AnsiStringT::LowerCase Method

Pascal

```
function LowerCase: AnsiStringT;
```

C++

```
__fastcall AnsiStringT LowerCase() const;
```

Description

This is LowerCase, a member of class AnsiStringT.

38.1.2.3.26 AnsiStringT::Pos Method

Pascal

```
function Pos(const subStr: AnsiStringT&): Integer;
```

C++

```
__fastcall int Pos(const AnsiStringT& subStr) const;
```

Description

This is Pos, a member of class AnsiStringT.

38.1.2.3.27 AnsiStringT::printf Method

Pascal

```
function printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const char* format, ...);
```

Description

This is printf, a member of class AnsiStringT.

38.1.2.3.28 AnsiStringT::RefCount Method

Gets reference count of System::AnsiStringBase.

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::AnsiStringBase::RefCount gets the reference count of a System::AnsiStringBase instance. If the instance is empty, it returns 0.

See Also

System::AnsiStringBase::GetRec (see page 3204)

38.1.2.3.29 AnsiStringT::SetLength Method

Pascal

```
function SetLength(newLength: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& SetLength(int newLength);
```

Description

This is SetLength, a member of class AnsiStringT.

38.1.2.3.30 AnsiStringT::sprintf Method

Pascal

```
function sprintf(const format: char*; : ...): AnsiStringT&; cdecl;
```

C++

```
__cdecl AnsiStringT& sprintf(const char* format, ...);
```

Description

This is sprintf, a member of class AnsiStringT.

38.1.2.3.31 AnsiStringT::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: char; count: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT StringOfChar(char ch, int count);
```

Description

System::AnsiStringBase::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::AnsiStringBase::AnsiString s = System::AnsiStringBase::AnsiString::System::AnsiStringBase::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

38.1.2.3.32 AnsiStringT::SubString Method

Returns a specified substring of the System::AnsiStringBase::AnsiString.

Pascal

```
function SubString(index: Integer; count: Integer): AnsiStringT;
```

C++

```
__fastcall AnsiStringT SubString(int index, int count) const;
```

Description

System::AnsiStringBase::SubString returns a new System::AnsiStringBase::AnsiString that is a substring of this System::AnsiStringBase::AnsiString. The substring contains count characters beginning at index.

38.1.2.3.33 AnsiStringT::swap Method

Pascal

```
function swap(var other: AnsiStringT): AnsiStringT&;
```

C++

```
AnsiStringT& swap(AnsiStringT& other);
```

Description

This is swap, a member of class AnsiStringT.

38.1.2.3.34 AnsiStringT::Trim Method

Pascal

```
function Trim: AnsiStringT;
```

C++

```
__fastcall AnsiStringT Trim() const;
```

Description

This is Trim, a member of class AnsiStringT.

38.1.2.3.35 AnsiStringT::TrimLeft Method

Pascal

```
function TrimLeft: AnsiStringT;
```

C++

```
__fastcall AnsiStringT TrimLeft() const;
```

Description

This is TrimLeft, a member of class AnsiStringT.

38.1.2.3.36 AnsiStringT::TrimRight Method

Pascal

```
function TrimRight: AnsiStringT;
```

C++

```
__fastcall AnsiStringT TrimRight() const;
```

Description

This is TrimRight, a member of class AnsiStringT.

38.1.2.3.37 AnsiStringT::Unique Method

Makes the System::AnsiStringBase::AnsiString unique.

Pascal

```
function Unique: AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::AnsiStringBase::Unique sets the reference count on the string for the System::AnsiStringBase::AnsiString object to 1 so that it is the only object referencing that string.

38.1.2.3.38 AnsiStringT::UpperCase Method

Pascal

```
function UpperCase: AnsiStringT;
```

C++

```
__fastcall AnsiStringT UpperCase() const;
```

Description

This is UpperCase, a member of class AnsiStringT.

38.1.2.3.39 AnsiStringT::vprintf Method

Pascal

```
function vprintf(const format: char*; list: va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const char* format, va_list list);
```

Description

This is vprintf, a member of class AnsiStringT.

38.1.2.3.40 AnsiStringT::WideChar Method

Pascal

```
function WideChar(var dest: wchar_t; destSize: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* WideChar(wchar_t* dest, int destSize) const;
```

Description

This is WideChar, a member of class AnsiStringT.

38.1.2.3.41 AnsiStringT::WideCharBufSize Method

Pascal

```
function WideCharBufSize: Integer;
```

C++

```
__fastcall int WideCharBufSize() const;
```

Description

This is WideCharBufSize, a member of class AnsiStringT.

38.1.2.4 AnsiStringT Operators

The operators of the AnsiStringT class are listed here.

Public Operators

	Name	Description
	!= (see page 3240)	This is !=, a member of class AnsiStringT.
	+ (see page 3240)	This is +, a member of class AnsiStringT.
	+= (see page 3240)	This is +=, a member of class AnsiStringT.
	< (see page 3240)	This is <, a member of class AnsiStringT.
	<= (see page 3241)	This is <=, a member of class AnsiStringT.

	= (see page 3241)	This is =, a member of class AnsiStringT.
	== (see page 3241)	This is ==, a member of class AnsiStringT.
	> (see page 3241)	This is >, a member of class AnsiStringT.
	>= (see page 3241)	This is >=, a member of class AnsiStringT.

38.1.2.4.1 AnsiStringT::!= Operator

Pascal

```
operator !=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator !=(const AnsiStringT& rhs) const;
```

Description

This is !=, a member of class AnsiStringT.

38.1.2.4.2 AnsiStringT::+ Operator

Pascal

```
operator +(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT operator +(const AnsiStringT& rhs) const;
```

Description

This is +, a member of class AnsiStringT.

38.1.2.4.3 AnsiStringT::+= Operator

Pascal

```
operator +=(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT& operator +=(const AnsiStringT& rhs);
```

Description

This is +=, a member of class AnsiStringT.

38.1.2.4.4 AnsiStringT::< Operator

Pascal

```
operator <(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator <(const AnsiStringT& rhs) const;
```

Description

This is <, a member of class AnsiStringT.

38.1.2.4.5 AnsiStringT::<= Operator

Pascal

```
operator <=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator <=(const AnsiStringT& rhs) const;
```

Description

This is <=, a member of class AnsiStringT.

38.1.2.4.6 AnsiStringT::= Operator

Pascal

```
operator =(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT& operator =(const AnsiStringT& rhs);
```

Description

This is =, a member of class AnsiStringT.

38.1.2.4.7 AnsiStringT::== Operator

Pascal

```
operator ==(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator ==(const AnsiStringT& rhs) const;
```

Description

This is ==, a member of class AnsiStringT.

38.1.2.4.8 AnsiStringT::> Operator

Pascal

```
operator >(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator >(const AnsiStringT& rhs) const;
```

Description

This is >, a member of class AnsiStringT.

38.1.2.4.9 AnsiStringT::>= Operator

Pascal

```
operator >=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator >=(const AnsiStringT& rhs) const;
```

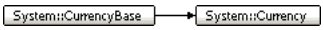
Description

This is >=, a member of class AnsiStringT.

38.1.3 Currency Class

System::Currency is the C++ implementation of the Delphi System::Currency data type.

Class Hierarchy



Pascal

```
Currency = class(CurrencyBase);
```

C++

```
class Currency : public CurrencyBase;
```

File

syscurr.h

Description

System::Currency is a C++ class that implements a Delphi intrinsic type, also called System::Currency. Use System::Currency to hold monetary values.

Note: A System::Currency object is implemented as a 64-bit signed integer. Avoid operations that might cause integer overflow.

Note: Use only the operators defined by System::Currency. The compiler ignores any operators you overload yourself. Use the System::Currency constructors to create System::Currency objects from pointers to other System::Currency objects, and also from numeric and string values..

Note: To utilize the C++ streaming operators (<< and >>) with System::Currency, you must use #include <iostream> before #include <syscurr.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

```
ostream& operator << (ostream& os, const Currency& arg);
istream& operator >> (istream& is, Currency& arg);
```

See Also

System::CurrencyComp

System::CurrencyCurrencyBase

38.1.3.1 Currency Members

The following tables list the members exposed by Currency.

Public Data Members

	Name	Description
	Val (see page 3263)	This is Val, a member of class CurrencyBase.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 3260)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

Public Methods

	Name	Description
	Currency (see page 3244)	This is the overview for the Currency constructor overload.

Public Operators

	Name	Description
	- (see page 3245)	This is the overview for the - operator overload.
	-- (see page 3246)	This is the overview for the -- operator overload.
	! (see page 3247)	Returns a new System::Currency::Currency (see page 3242) object that is the logical negation of this System::Currency::Currency (see page 3242) object.
	!= (see page 3248)	This is the overview for the != operator overload.
	% (see page 3248)	Returns a new System::Currency::Currency (see page 3242) object with the value that is the remainder of this System::Currency::Currency (see page 3242) object divided by a specified integer.
	%= (see page 3249)	Assigns this System::Currency::Currency (see page 3242) object the remainder of its initial value divided by integer rhs.
	* (see page 3249)	This is the overview for the * operator overload.
	*= (see page 3250)	Assigns this System::Currency::Currency (see page 3242) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3242) object rhs.
	/ (see page 3250)	This is the overview for the / operator overload.
	/= (see page 3251)	Assigns this System::Currency::Currency (see page 3242) object the result of its initial value divided by the value of System::Currency::Currency (see page 3242) object rhs.
	+ (see page 3252)	This is the overview for the + operator overload.
	++ (see page 3253)	This is the overview for the ++ operator overload.
	+= (see page 3253)	Increments the value of this System::Currency::Currency (see page 3242) object by the value of rhs.
	< (see page 3254)	This is the overview for the < operator overload.
	<= (see page 3255)	This is the overview for the <= operator overload.
	-= (see page 3256)	Decrements the value of this System::Currency::Currency (see page 3242) object by the value of rhs.
	= (see page 3255)	This is the overview for the = operator overload.
	== (see page 3257)	This is the overview for the == operator overload.
	> (see page 3257)	This is the overview for the > operator overload.
	>= (see page 3258)	This is the overview for the >= operator overload.
	double (see page 3259)	Converts the System::Currency::Currency (see page 3242) object to a double value.
	int (see page 3259)	Converts the System::Currency::Currency (see page 3242) object to an integer value

	System::String (see page 3260)	This is System::String, a member of class Currency.

38.1.3.2 Currency Constructor

This is the overview for the Currency constructor overload.

Overload List

	Name	Description
	Currency::Currency () (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.
	Currency::Currency (Currency&) (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.
	Currency::Currency (CurrencyBase&) (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.
	Currency::Currency (double) (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.
	Currency::Currency (int) (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.
	Currency::Currency (System::String&) (see page 3244)	Creates and initializes a System::Currency::Currency (see page 3242) object.

38.1.3.2.1 Currency::Currency Constructor ()

Creates and initializes a System::Currency::Currency (see page 3242) object.

Pascal

```
constructor Create;
constructor Create(val: double);
constructor Create(val: Integer);
constructor Create(const src: CurrencyBase&);
constructor Create(const src: Currency&);
constructor Create(const src: System::String&);
```

C++

```
__fastcall Currency();
__fastcall Currency(double val);
__fastcall Currency(int val);
__fastcall Currency(const CurrencyBase& src);
__fastcall Currency(const Currency& src);
__fastcall Currency(const System::String& src);
```

Description

Use System::Currency::Currency (see page 3242) to instantiate a System::Currency::Currency (see page 3242) object.

When called with no parameters, the resulting object has a value of 0.

When called with a System::Currency::Currency (see page 3242) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::Currency::CurrencyBase (see page 3262) object as the src parameter, the resulting object has the same value as the src parameter. System::Currency::CurrencyBase (see page 3262) is the base class for the System::Currency::Currency (see page 3242) type. It has a single member, a public Val (see page 3785) field that represents the value of the object.

When called with an AnsiString object as the src parameter, the value of the resulting object is the parsed value of the AnsiString. The AnsiString must represent a numeric value within the range valid for System::Currency::Currency (see page 3242) (-922337203685477.5808 to 922337203685477.5807).

When called with a double (see page 3259) or integer as the val parameter, the value of the resulting object is the value of the val parameter.

38.1.3.3 Currency Operators

The operators of the Currency class are listed here.

Public Operators

	Name	Description
	- (see page 3245)	This is the overview for the - operator overload.
	-- (see page 3246)	This is the overview for the -- operator overload.
	! (see page 3247)	Returns a new System::Currency::Currency (see page 3242) object that is the logical negation of this System::Currency::Currency (see page 3242) object.
	!= (see page 3248)	This is the overview for the != operator overload.
	% (see page 3248)	Returns a new System::Currency::Currency (see page 3242) object with the value that is the remainder of this System::Currency::Currency (see page 3242) object divided by a specified integer.
	%= (see page 3249)	Assigns this System::Currency::Currency (see page 3242) object the remainder of its initial value divided by integer rhs.
	* (see page 3249)	This is the overview for the * operator overload.
	*= (see page 3250)	Assigns this System::Currency::Currency (see page 3242) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3242) object rhs.
	/ (see page 3250)	This is the overview for the / operator overload.
	/= (see page 3251)	Assigns this System::Currency::Currency (see page 3242) object the result of its initial value divided by the value of System::Currency::Currency (see page 3242) object rhs.
	+ (see page 3252)	This is the overview for the + operator overload.
	++ (see page 3253)	This is the overview for the ++ operator overload.
	+= (see page 3253)	Increments the value of this System::Currency::Currency (see page 3242) object by the value of rhs.
	< (see page 3254)	This is the overview for the < operator overload.
	<= (see page 3255)	This is the overview for the <= operator overload.
	= (see page 3255)	This is the overview for the = operator overload.
	-= (see page 3256)	Decrements the value of this System::Currency::Currency (see page 3242) object by the value of rhs.
	== (see page 3257)	This is the overview for the == operator overload.
	> (see page 3257)	This is the overview for the > operator overload.
	>= (see page 3258)	This is the overview for the >= operator overload.
	double (see page 3259)	Converts the System::Currency::Currency (see page 3242) object to a double value.
	int (see page 3259)	Converts the System::Currency::Currency (see page 3242) object to an integer value
	System::String (see page 3260)	This is System::String, a member of class Currency.

38.1.3.3.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Currency::- () (see page 3246)	This is -, a member of class Currency.

	Currency::- (Currency&) (see page 3246)	This is -, a member of class Currency.
	Currency::- (double) (see page 3246)	This is -, a member of class Currency.
	Currency::- (int) (see page 3246)	This is -, a member of class Currency.

38.1.3.3.1.1 Currency::- Operator ()

Pascal

```
operator -;
```

C++

```
__fastcall Currency operator -() const;
```

Description

This is -, a member of class Currency.

38.1.3.3.1.2 Currency::- Operator (Currency&)

Pascal

```
operator -(const rhs: Currency&);
```

C++

```
__fastcall Currency operator -(const Currency& rhs) const;
```

Description

This is -, a member of class Currency.

38.1.3.3.1.3 Currency::- Operator (double)

Pascal

```
operator -(rhs: double);
```

C++

```
__fastcall Currency operator -(double rhs) const;
```

Description

This is -, a member of class Currency.

38.1.3.3.1.4 Currency::- Operator (int)

Pascal

```
operator -(rhs: Integer);
```

C++

```
__fastcall Currency operator -(int rhs) const;
```

Description

This is -, a member of class Currency.

38.1.3.3.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	Currency::-- () (see page 3247)	Decrements the value of the System::Currency::Currency (see page 3242) object by a specified integer value.
	Currency::-- (int) (see page 3247)	Decrements the value of the System::Currency::Currency (see page 3242) object by a specified integer value.

38.1.3.3.2.1 Currency::-- Operator ()

Decrements the value of the System::Currency::Currency (see page 3242) object by a specified integer value.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
Currency& operator --();
Currency operator --(int);
```

Description

Use the unary System::Currency (see page 3242)::-- to decrement the value of the System::Currency::Currency (see page 3242) object by 1.

Use the binary System::Currency (see page 3242)::-- to decrement the value by the integer specified as an operand.

See Also

System::Currency::++ (see page 3253)

System::Currency::-= (see page 3256)

System::Currency::operator_minus_binary

38.1.3.3.3 Currency::! Operator

Returns a new System::Currency::Currency (see page 3242) object that is the logical negation of this System::Currency::Currency (see page 3242) object.

Pascal

```
operator !;
```

C++

```
__fastcall Currency operator !() const;
```

Description

If the value of the System::Currency::Currency (see page 3242) object is nonzero, the System::Currency (see page 3242)::! returns 0. If the value of the System::Currency::Currency (see page 3242) object is 0, the System::Currency (see page 3242)::! returns 1.

See Also

System::Currency::!= (see page 3248)

System::Currency::operator_minus_unary

38.1.3.3.4 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Currency::!= (Currency&) (see page 3248)	Tests for inequality between the System::Currency::Currency (see page 3242) object's value and a specified value.
	Currency::!= (double) (see page 3248)	Tests for inequality between the System::Currency::Currency (see page 3242) object's value and a specified value.
	Currency::!= (int) (see page 3248)	Tests for inequality between the System::Currency::Currency (see page 3242) object's value and a specified value.

38.1.3.3.4.1 Currency::!= Operator (Currency&)

Tests for inequality between the System::Currency::Currency (see page 3242) object's value and a specified value.

Pascal

```
operator !=(const rhs: Currency&);
operator !=(rhs: Integer);
operator !=(rhs: double);
```

C++

```
__fastcall bool operator !=(const Currency& rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
```

Description

The System::Currency (see page 3242)::!= compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. If the values differ, the System::Currency (see page 3242)::!= returns true. If the values are the same, the System::Currency (see page 3242)::!= returns false.

See Also

System::Currency::= (see page 3256)

System::Currency::> (see page 3258)

System::Currency::>= (see page 3258)

System::Currency::< (see page 3254)

System::Currency::<= (see page 3255)

38.1.3.3.5 Currency::% Operator

Returns a new System::Currency::Currency (see page 3242) object with the value that is the remainder of this System::Currency::Currency (see page 3242) object divided by a specified integer.

Pascal

```
operator %(rhs: Integer);
```

C++

```
__fastcall Currency operator %(int rhs) const;
```

Description

Use the System::Currency (see page 3242)::% to obtain the remainder from dividing the value of this System::Currency::Currency (see page 3242) object by the integer specified by the rhs operand.

See Also

System::Currency::%= (see page 3249)

System::Currency::/ (see page 3251)

38.1.3.3.6 Currency::%= Operator

Assigns this System::Currency::Currency (see page 3242) object the remainder of its initial value divided by integer rhs.

Pascal

```
operator %=(rhs: Integer);
```

C++

```
__fastcall Currency& operator %=(int rhs);
```

Description

Use the System::Currency (see page 3242)::%= to obtain the remainder from dividing this System::Currency::Currency (see page 3242) object's value by the Value of the System::Currency::Currency (see page 3242) object specified by rhs.

See Also

System::Currency::= (see page 3256)

System::Currency::/= (see page 3251)

System::Currency::% (see page 3248)

38.1.3.3.7 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Currency::* (Currency&) (see page 3249)	Multiplies the value of the System::Currency::Currency (see page 3242) object by a specified value.
	Currency::* (double) (see page 3249)	Multiplies the value of the System::Currency::Currency (see page 3242) object by a specified value.
	Currency::* (int) (see page 3249)	Multiplies the value of the System::Currency::Currency (see page 3242) object by a specified value.

38.1.3.3.7.1 Currency::* Operator (Currency&)

Multiplies the value of the System::Currency::Currency (see page 3242) object by a specified value.

Pascal

```
operator *(const rhs: Currency&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Currency operator *(const Currency& rhs) const;
__fastcall Currency operator *(int rhs) const;
__fastcall Currency operator *(double rhs) const;
```

Description

Use the System::Currency (see page 3242)::* to multiply the value of the System::Currency::Currency (see page 3242) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency (see page 3242)::*s:

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

multiples the values of integer lhs and System::Currency::Currency (see page 3242) object rhs and returns a new System::Currency::Currency (see page 3242) object with the resulting value.

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

multiples the values of double (see page 3259) lhs and System::Currency::Currency (see page 3242) object rhs and returns a new System::Currency::Currency (see page 3242) object with the resulting value.

See Also

System::Currency::*= (see page 3250)

System::Currency::/ (see page 3251)

38.1.3.3.8 Currency::*= Operator

Assigns this System::Currency::Currency (see page 3242) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3242) object rhs.

Pascal

```
operator *=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator *=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3242)::*= to multiply this System::Currency::Currency (see page 3242) object's value by the Value of the System::Currency::Currency (see page 3242) object specified by rhs.

See Also

System::Currency::= (see page 3256)

System::Currency::/= (see page 3251)

System::Currency::* (see page 3249)

38.1.3.3.9 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Currency::/ (Currency&) (see page 3251)	Divides the value of the System::Currency::Currency (see page 3242) object by a specified value.

	Currency::/ (double) (see page 3251)	Divides the value of the System::Currency::Currency (see page 3242) object by a specified value.
	Currency::/ (int) (see page 3251)	Divides the value of the System::Currency::Currency (see page 3242) object by a specified value.

38.1.3.3.9.1 Currency::/ Operator (Currency&)

Divides the value of the System::Currency::Currency (see page 3242) object by a specified value.

Pascal

```
operator /(const rhs: Currency&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Currency operator /(const Currency& rhs) const;
__fastcall Currency operator /(int rhs) const;
__fastcall Currency operator /(double rhs) const;
```

Description

Use the System::Currency (see page 3242)::/ to divide the value of the System::Currency::Currency (see page 3242) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency::operator_slashs:

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

divides the value of int (see page 3259) lhs by the value of System::Currency::Currency (see page 3242) object rhs and returns a new System::Currency::Currency (see page 3242) object with the resulting value.

```
friend Currency __fastcalloperator /(double lhs, const Currency& rhs);
```

divides the value of double (see page 3259) lhs by the value of System::Currency::Currency (see page 3242) object rhs and returns a System::Currency::Currency (see page 3242) object with the resulting value.

See Also

System::Currency::/= (see page 3251)

System::Currency::* (see page 3249)

System::Currency::% (see page 3248)

38.1.3.3.10 Currency::/= Operator

Assigns this System::Currency::Currency (see page 3242) object the result of its initial value divided by the value of System::Currency::Currency (see page 3242) object rhs.

Pascal

```
operator /=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator /=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3242)::/= to divide this System::Currency::Currency (see page 3242) object's value by the Value of the System::Currency::Currency (see page 3242) object specified by rhs.

See Also

System::Currency::= (see page 3256)

System::Currency::*= (see page 3250)

System::Currency::/ (see page 3251)

38.1.3.3.11 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Currency::+ (Currency&) (see page 3252)	Adds a specified value to the System::Currency::Currency (see page 3242) object.
	Currency::+ (double) (see page 3252)	Adds a specified value to the System::Currency::Currency (see page 3242) object.
	Currency::+ (int) (see page 3252)	Adds a specified value to the System::Currency::Currency (see page 3242) object.

38.1.3.3.11.1 Currency::+ Operator (Currency&)

Adds a specified value to the System::Currency::Currency (see page 3242) object.

Pascal

```
operator +(const rhs: Currency&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Currency operator +(const Currency& rhs) const;
__fastcall Currency operator +(int rhs) const;
__fastcall Currency operator +(double rhs) const;
```

Description

Use the System::Currency (see page 3242)::+ to add the value specified by the rhs operand to the System::Currency::Currency (see page 3242) object. If the rhs operand is another System::Currency::Currency (see page 3242) object, the System::Currency (see page 3242)::+ adds its value to the System::Currency::Currency (see page 3242) object. If it is an integer or double (see page 3259) value, the System::Currency (see page 3242)::+ adds that value to the value of the System::Currency::Currency (see page 3242) object.

Note that there are two related friend System::Currency (see page 3242)::+s:

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 3242) object that is the sum of int (see page 3259) lhs and System::Currency::Currency (see page 3242) object rhs.

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 3242) object that is the sum of double (see page 3259) lhs and System::Currency::Currency (see page 3242) object rhs.

See Also

System::Currency::+= (see page 3253)

System::Currency::++ (see page 3253)

System::Currency::operator_minus_binary

38.1.3.3.12 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	Currency::++ () (see page 3253)	Increments the value of the System::Currency::Currency (see page 3242) object by a specified integer value.
	Currency::++ (int) (see page 3253)	Increments the value of the System::Currency::Currency (see page 3242) object by a specified integer value.

38.1.3.3.12.1 Currency::++ Operator ()

Increments the value of the System::Currency::Currency (see page 3242) object by a specified integer value.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
Currency& operator ++();
Currency operator ++(int);
```

Description

Use the unary System::Currency (see page 3242)::++ to increment the value of the System::Currency::Currency (see page 3242) object by 1.

Use the binary System::Currency (see page 3242)::++ to increment the value by the integer specified as an operand.

See Also

System::Currency::-- (see page 3247)

System::Currency::+= (see page 3253)

System::Currency::+ (see page 3252)

38.1.3.3.13 Currency::+= Operator

Increments the value of this System::Currency::Currency (see page 3242) object by the value of rhs.

Pascal

```
operator +=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator +=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3242)::+= to add the value of another System::Currency::Currency (see page 3242) object to this one.

See Also

System::Currency::= (see page 3256)

System::Currency::-= (see page 3256)

System::Currency::+ (see page 3252)

System::Currency::++ (see page 3253)

38.1.3.3.14 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Currency::< (Currency&) (see page 3254)	Tests whether this System::Currency::Currency (see page 3242) value is less than a specified value.
	Currency::< (double) (see page 3254)	Tests whether this System::Currency::Currency (see page 3242) value is less than a specified value.
	Currency::< (int) (see page 3254)	Tests whether this System::Currency::Currency (see page 3242) value is less than a specified value.

38.1.3.3.14.1 Currency::< Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3242) value is less than a specified value.

Pascal

```
operator <(const rhs: Currency&);
operator <(rhs: Integer);
operator <(rhs: double);
```

C++

```
__fastcall bool operator <(const Currency& rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
```

Description

The < System::Currency (see page 3242)::< compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3242) object has the lesser value, false otherwise.

See Also

System::Currency::!= (see page 3248)

System::Currency::> (see page 3258)

System::Currency::>= (see page 3258)

System::Currency::== (see page 3257)

System::Currency::<= (see page 3255)

38.1.3.3.15 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Currency::<= (Currency&) (see page 3255)	Tests whether this System::Currency::Currency (see page 3242) value is less than or equal to a specified value.
	Currency::<= (double) (see page 3255)	Tests whether this System::Currency::Currency (see page 3242) value is less than or equal to a specified value.
	Currency::<= (int) (see page 3255)	Tests whether this System::Currency::Currency (see page 3242) value is less than or equal to a specified value.

38.1.3.3.15.1 Currency::<= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3242) value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Currency&);
operator <=(rhs: Integer);
operator <=(rhs: double);
```

C++

```
__fastcall bool operator <=(const Currency& rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
```

Description

The <= System::Currency (see page 3242)::<= compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3242) object has a value less than or equal to that of rhs, false otherwise.

See Also

System::Currency::!= (see page 3248)

System::Currency::> (see page 3258)

System::Currency::>= (see page 3258)

System::Currency::< (see page 3254)

System::Currency::== (see page 3257)

38.1.3.3.16 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Currency::= (Currency&) (see page 3256)	Assigns the value of the System::Currency::Currency (see page 3242) object.

	Currency::= (CurrencyBase&) (see page 3256)	Assigns the value of the System::Currency::Currency (see page 3242) object.
	Currency::= (double) (see page 3256)	Assigns the value of the System::Currency::Currency (see page 3242) object.
	Currency::= (int) (see page 3256)	Assigns the value of the System::Currency::Currency (see page 3242) object.

38.1.3.3.16.1 Currency::= Operator (Currency&)

Assigns the value of the System::Currency::Currency (see page 3242) object.

Pascal

```
operator =(rhs: double);
operator =(rhs: Integer);
operator =(const rhs: CurrencyBase&);
operator =(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator =(double rhs);
__fastcall Currency& operator =(int rhs);
__fastcall Currency& operator =(const CurrencyBase& rhs);
__fastcall Currency& operator =(const Currency& rhs);
```

Description

The assignment System::Currency (see page 3242)::= assigns the value of the rhs operand to the System::Currency::Currency (see page 3242) object. The assignment System::Currency (see page 3242)::= can take

a double (see page 3259) or integer value, which is assigned to the System::Currency::Currency (see page 3242) object.

a System::Currency::CurrencyBase (see page 3262) or System::Currency::Currency (see page 3242) object, whose value is assigned to the System::Currency::Currency (see page 3242) object.

38.1.3.3.17 Currency::-= Operator

Decrements the value of this System::Currency::Currency (see page 3242) object by the value of rhs.

Pascal

```
operator -=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator -=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3242)::-= to subtract the value of another System::Currency::Currency (see page 3242) object from this one.

See Also

System::Currency::= (see page 3256)

System::Currency::+= (see page 3253)

System::Currency::operator_minus_binary

System::Currency::-- (see page 3247)

38.1.3.3.18 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Currency::== (Currency&) (see page 3257)	Tests for equality between the System::Currency::Currency (see page 3242) object's value and a specified value.
	Currency::== (double) (see page 3257)	Tests for equality between the System::Currency::Currency (see page 3242) object's value and a specified value.
	Currency::== (int) (see page 3257)	Tests for equality between the System::Currency::Currency (see page 3242) object's value and a specified value.

38.1.3.3.18.1 Currency::== Operator (Currency&)

Tests for equality between the System::Currency::Currency (see page 3242) object's value and a specified value.

Pascal

```
operator ==(const rhs: Currency&);
operator ==(rhs: Integer);
operator ==(rhs: double);
```

C++

```
__fastcall bool operator ==(const Currency& rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(double rhs) const;
```

Description

The == System::Currency (see page 3242)::== compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. If the two values are the same, the System::Currency (see page 3242)::== returns true. If the values differ, the System::Currency (see page 3242)::== returns false.

See Also

System::Currency::!= (see page 3248)

System::Currency::> (see page 3258)

System::Currency::>= (see page 3258)

System::Currency::< (see page 3254)

System::Currency::<= (see page 3255)

38.1.3.3.19 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Currency::> (Currency&) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than a specified value.

	Currency::> (double) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than a specified value.
	Currency::> (int) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than a specified value.

38.1.3.3.19.1 Currency::> Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3242) value is greater than a specified value.

Pascal

```
operator >(const rhs: Currency&);
operator >(rhs: Integer);
operator >(rhs: double);
```

C++

```
__fastcall bool operator >(const Currency& rhs) const;
__fastcall bool operator >(int rhs) const;
__fastcall bool operator >(double rhs) const;
```

Description

The System::Currency (see page 3242)::> compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3242) object has a greater value than rhs, false otherwise.

See Also

System::Currency::!= (see page 3248)

System::Currency::== (see page 3257)

System::Currency::>= (see page 3258)

System::Currency::< (see page 3254)

System::Currency::<= (see page 3255)

38.1.3.3.20 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Currency::>= (Currency&) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than or equal to a specified value.
	Currency::>= (double) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than or equal to a specified value.
	Currency::>= (int) (see page 3258)	Tests whether this System::Currency::Currency (see page 3242) value is greater than or equal to a specified value.

38.1.3.3.20.1 Currency::>= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3242) value is greater than or equal to a specified value.

Pascal

```
operator >=(const rhs: Currency&);
operator >=(rhs: Integer);
operator >=(rhs: double);
```

C++

```
__fastcall bool operator >=(const Currency& rhs) const;
__fastcall bool operator >=(int rhs) const;
__fastcall bool operator >=(double rhs) const;
```

Description

The >= System::Currency (see page 3242)::>= compares the value of the System::Currency::Currency (see page 3242) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3242) object's value is greater than or equal to the value of rhs, false otherwise.

See Also

System::Currency::!= (see page 3248)

System::Currency::> (see page 3258)

System::Currency::== (see page 3257)

System::Currency::< (see page 3254)

System::Currency::<= (see page 3255)

38.1.3.3.21 Currency::double Operator

Converts the System::Currency::Currency (see page 3242) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

double is a conversion System::Currency::double that converts a System::Currency::Currency (see page 3242) object to a double.

See Also

System::Currency::int (see page 3259)

System::Currency::AnsiString

38.1.3.3.22 Currency::int Operator

Converts the System::Currency::Currency (see page 3242) object to an integer value

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

int is a conversion System::Currency::int that converts a System::Currency::Currency (see page 3242) object to an int.

See Also

System::Currency::double (see page 3259)

System::Currency::AnsiString

38.1.3.3.23 Currency::System::String Operator

Pascal

```
operator System::String;
```

C++

```
__fastcall operator System::String() const;
```

Description

This is System::String, a member of class Currency.

38.1.3.4 Currency Friends

The friends of the Currency class are listed here.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 3260)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 3261)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 3262)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

38.1.3.4.1 friend Currency __fastcall operator -(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).

38.1.3.4.2 friend Currency __fastcall operator -(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).

38.1.3.4.3 friend Currency __fastcall operator *(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).

38.1.3.4.4 friend Currency __fastcall operator *(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).

38.1.3.4.5 friend Currency __fastcall operator /(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).

38.1.3.4.6 friend Currency __fastcall operator /(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).

38.1.3.4.7 friend Currency __fastcall operator +(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).

38.1.3.4.8 friend Currency __fastcall operator +(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

38.1.4 CurrencyBase Class

Class Hierarchy

Pascal

```
CurrencyBase = class;
```

C++

```
class CurrencyBase;
```

File

syscurr.h

Description

This is class System::CurrencyBase.

38.1.4.1 CurrencyBase Members

The following tables list the members exposed by CurrencyBase.

Public Data Members

	Name	Description
	Val (see page 3263)	This is Val, a member of class CurrencyBase.

38.1.4.2 CurrencyBase Data Members

The data members of the CurrencyBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 3263)	This is Val, a member of class CurrencyBase.

38.1.4.2.1 CurrencyBase::Val Data Member

Pascal

```
Val: __int64;
```

C++

```
__int64 Val;
```

Description

This is Val, a member of class CurrencyBase.

38.1.5 DelphiInterface Template

System::DelphiInterface is a C++ implementation of the Delphi interface.

Class Hierarchy

Pascal

```
DelphiInterface = class;
```

C++

```
template <class T>
class DelphiInterface;
```

File

systobj.h

Description

A Delphi interface is an abstract collection of properties, methods, and events with no implementation. It resembles a C++ class with only pure virtual methods. However a Delphi interface type supports built-in reference counting.

System::DelphiInterface is a C++ template that defines types compatible with Delphi interfaces, System::DelphiInterface defines reference counting logic in its initialization constructor, copy constructor, assignment operator, and destructor.

Note: Other operators that provide access to the underlying interface pointer do not handle reference counting. You may need to explicitly call AddRef or Release (see page 3266) to ensure proper reference counting.

System::DelphiInterface is a template:

```
template <class T> class RTL_DELPHIRETURN DelphiInterface;
```

where the T parameter is an underlying interface class or COM interface. In the object framework, the resulting System::DelphiInterface class is usually given a name such as _di_IUnknown, where IUnknown (see page 3828) is the name of the corresponding interface. Such interfaces are documented under the interface name (such as IFormDesigner).

See Also

Variant

TVariant

38.1.5.1 DelphiInterface Members

The following tables list the members exposed by DelphiInterface.

Public Methods

	Name	Description
	~DelphiInterface (see page 3264)	Frees an instance of System::DelphiInterface::DelphiInterface (see page 3263).
	DelphiInterface (see page 3265)	This is the overview for the DelphiInterface constructor overload.
	Release (see page 3266)	This is Release, a member of class DelphiInterface.

Public Operators

	Name	Description
	! (see page 3266)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 3263) object represents a NULL interface.
	& (see page 3266)	Returns a pointer to the underlying interface pointer.
	* (see page 3267)	Returns the underlying interface pointer.
	= (see page 3267)	This is the overview for the = operator overload.
	-> (see page 3268)	This is ->, a member of class DelphiInterface.
	T* (see page 3268)	This is T*, a member of class DelphiInterface.

38.1.5.2 DelphiInterface::~DelphiInterface Destructor

Frees an instance of System::DelphiInterface::DelphiInterface (see page 3263).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~DelphiInterface<T>();
```

Description

Before freeing the System::DelphiInterface::DelphiInterface (see page 3263) instance, System::DelphiInterface (see page 3263)::~DelphiInterface (see page 3263) calls the Release (see page 3266) method of the underlying interface.

38.1.5.3 DelphiInterface Constructor

This is the overview for the DelphiInterface constructor overload.

Overload List

	Name	Description
	DelphiInterface::DelphiInterface () (see page 3265)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3263).
	DelphiInterface::DelphiInterface (DelphiInterface<T> &) (see page 3265)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3263).
	DelphiInterface::DelphiInterface (T*) (see page 3265)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3263).

38.1.5.3.1 DelphiInterface::DelphiInterface Constructor ()

Creates an instance of System::DelphiInterface::DelphiInterface (see page 3263).

Pascal

```
constructor Create;
constructor Create(const rhs: DelphiInterface<T> &);
constructor Create(var rhs: T);
```

C++

```
__fastcall DelphiInterface<T>();
__fastcall DelphiInterface<T>(const DelphiInterface<T> & rhs);
__fastcall DelphiInterface<T>(T* rhs);
```

Description

The System::DelphiInterface::DelphiInterface (see page 3263) constructor is overloaded to create a System::DelphiInterface::DelphiInterface (see page 3263) in one of three ways:

When used with no parameters, System::DelphiInterface::DelphiInterface (see page 3263) creates a NULL interface.

When used with another System::DelphiInterface::DelphiInterface (see page 3263) object (the copy constructor), System::DelphiInterface::DelphiInterface (see page 3263) creates a new copy of the interface specified by rhs, incrementing the reference count of the underlying interface.

When used with an interface pointer of the underlying type, System::DelphiInterface::DelphiInterface (see page 3263) wraps the underlying interface specified by rhs and increments its reference count.

38.1.5.4 DelphiInterface Methods

The methods of the DelphiInterface class are listed here.

Public Methods

	Name	Description
	Release (see page 3266)	This is Release, a member of class DelphiInterface.

38.1.5.4.1 DelphiInterface::Release Method

Pascal

```
function Release: Integer;
```

C++

```
int Release();
```

Description

This is Release, a member of class DelphiInterface.

38.1.5.5 DelphiInterface Operators

The operators of the DelphiInterface class are listed here.

Public Operators

	Name	Description
	! (see page 3266)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 3263) object represents a NULL interface.
	& (see page 3266)	Returns a pointer to the underlying interface pointer.
	* (see page 3267)	Returns the underlying interface pointer.
	= (see page 3267)	This is the overview for the = operator overload.
	-> (see page 3268)	This is ->, a member of class DelphiInterface.
	T* (see page 3268)	This is T*, a member of class DelphiInterface.

38.1.5.5.1 DelphiInterface::! Operator

Indicates whether the System::DelphiInterface::DelphiInterface (see page 3263) object represents a NULL interface.

Pascal

```
operator !;
```

C++

```
bool operator !() const;
```

Description

The ! System::DelphiInterface (see page 3263)::! returns true if the System::DelphiInterface::DelphiInterface (see page 3263) object represents a NULL interface, and false otherwise.

38.1.5.5.2 DelphiInterface::& Operator

Returns a pointer to the underlying interface pointer.

Pascal

```
operator &;
```

C++

```
__fastcall T** operator &();
```

Description

Use the & System::DelphiInterface (see page 3263)::& to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 3263)::&, because it provides no reference counting support. For example, when using this System::DelphiInterface (see page 3263)::& to programmatically replace the underlying interface, you must first call Release (see page 3266) to decrement the reference count of the current interface and, depending on whether you are using this as an additional interface reference, you may need to increment the reference count on the interface being added.

38.1.5.5.3 DelphiInterface::* Operator

Returns the underlying interface pointer.

Pascal

```
operator *;
```

C++

```
__fastcall T& operator *();
```

Description

Use the & System::DelphiInterface (see page 3263)::* to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 3263)::*, because it provides no reference counting support. For example, you may need to increment the reference count on the returned value.

38.1.5.5.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	DelphiInterface::= (DelphiInterface<T>&) (see page 3267)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3263).
	DelphiInterface::= (T *) (see page 3267)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3263).

38.1.5.5.4.1 DelphiInterface::= Operator (DelphiInterface<T>&)

Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3263).

Pascal

```
operator =(var rhs: DelphiInterface<T>);
operator =(var rhs: T);
```

C++

```
__fastcall DelphiInterface<T>& operator =(DelphiInterface<T>& rhs);
__fastcall DelphiInterface<T>& operator =(T * rhs);
```

Description

The assignment System::DelphiInterface (see page 3263)::= assigns a value to the underlying interface from an interface pointer of the underlying type or from another System::DelphiInterface::DelphiInterface (see page 3263) object that wraps the same interface type.

This System::DelphiInterface (see page 3263)::= manages all reference counting issues. That is, it decrements the reference count on the current interface (if any) before making the assignment and increments the reference count on the newly assigned

interface (if any).

38.1.5.5.5 DelphiInterface::-> Operator

Pascal

```
operator ->;
```

C++

```
__fastcall T* operator ->() const;
```

Description

This is ->, a member of class DelphiInterface.

38.1.5.5.6 DelphiInterface::T* Operator

Pascal

```
operator T*;
```

C++

```
__fastcall operator T*() const;
```

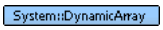
Description

This is T*, a member of class DelphiInterface.

38.1.6 DynamicArray Template

System::DynamicArray is the C++ analog for the Delphi dynamic array type.

Class Hierarchy



Pascal

```
DynamicArray = class;
```

C++

```
template <class T>
class DynamicArray;
```

File

sysdyn.h

Description

A Dynamic Array is a resizable collection implemented as an array. The System::DynamicArray template is a C++ implementation of Delphi Dynamic Array types. The System::DynamicArray template uses a binary layout compatible with the Delphi Dynamic Array implementation.

```
template <class T> class DELPHIRETURN DynamicArray;
```

You must specify one template parameter:

Parameter	Usage
type	the type of the elements

Length (see page 3278) of dynamic arrays

The size of a System::DynamicArray can be set or obtained from the Length (see page 3278) property:

//C++ example

```
DynamicArray<int> arrayOfInt;
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

To free a System::DynamicArray, assign it a Length (see page 3278) of zero.

Accessing data

Access elements of a Dynamic Array via the subscript [] operator:

//C++ example

```
void InitArray(DynamicArray<char> &c_array)
{
  c_array[0] = 'A';
  c_array[1] = 'B';
  cout << "Third char is: " << c_array[2];
}
```

Bounds of dynamic arrays

The Low (see page 3278) and High (see page 3277) properties of System::DynamicArrays return the low and high bounds of the array respectively.

//C++ example

```
int TotalArray(const DynamicArray<int>& arrayOfInt)
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

Note: The Low (see page 3278) property always returns 0. The High (see page 3277) property returns Length (see page 3278)-1. So you may also traverse a dynamic array via the following construct:

for (int i=0; i<arrayOfInt. Length (see page 3278); i++)

Assigning, comparing and copying dynamic arrays

Dynamic Arrays are reference counted. When a System::DynamicArray is assigned to another, only the reference is assigned (and the reference count adjusted), the content of the source is not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To copy the contents of a System::DynamicArray, you must use the Copy (see page 3273) (or CopyRange (see page 3273)) methods.

//C++ example

```
void foo(DynamicArray<int> &i_array)
{
  DynamicArray<int> temp = i_array;
  assert(temp == i_array); // temp and i_array point to same chunk
                           // of data..
  i_array[0] = 20;
  assert(temp[0] == 20); // Temp 'sees' changes to i_array..
```

```
  temp = i_array.Copy(); // Temp gets its own copy of data
  temp[0] = 10;
  assert(temp[0] != i_array[0]); // Above assignment did not
                                 // modify i_array.
}
```

Multidimensional dynamic arrays

Dynamic arrays can be multidimensional. Multidimensional dynamic arrays don't have to be rectangular. The Length (see page 3278) property can be used to set each dimension. The following example illustrates.

//C++ example

```
typedef DynamicArray< DynamicArray < AnsiString > > T2DStringArray;
void foo(T2DStringArray &s_array)
{
  SetLength(s_array, 10);
for (int i=0; i<s_array.Length; i++)
  { // Set lengths of second dimensions.(NOTE: non-rectangular)
    SetLength(s_array[i], i+1);
for (int j=0; j<s_array[i].Length; j++)
    /* */ s_array[i][j] = itoa(i*10+j);
  }
}
```

C++ & Delphi System::DynamicArrays

A dynamic array declared in C++ code can be used in Delphi code and vice versa. The following examples illustrate passing a one-dimensional and two-dimensional array between C++ and Delphi.

C++ System::DynamicArray Example

Delphi System::DynamicArray Example

38.1.6.1 DynamicArray Members

The following tables list the members exposed by DynamicArray.

Protected Methods

	Name	Description
	AllocData (see page 3272)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 3274)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 3274)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 3275)	This is IncRefCount, a member of class DynamicArray.
	SetData (see page 3275)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	~DynamicArray (see page 3271)	Frees a DynamicArray (see page 3268) object.
	Copy (see page 3272)	This is the overview for the Copy method overload.
	CopyRange (see page 3273)	This is the overview for the CopyRange method overload.
	DimCount (see page 3274)	This is DimCount, a member of class DynamicArray.
	DynamicArray (see page 3271)	This is the overview for the DynamicArray constructor overload.
	get_high (see page 3274)	This is get_high, a member of class DynamicArray.
	get_length (see page 3275)	This is get_length, a member of class DynamicArray.
	get_low (see page 3275)	This is get_low, a member of class DynamicArray.
	set_length (see page 3275)	This is set_length, a member of class DynamicArray.

Public Operators

	Name	Description
	[] (see page 3276)	This is the overview for the [] operator overload.
	= (see page 3276)	Assigns data from another Dynamic Array.
	== (see page 3277)	Compares two Dynamic Arrays for equality.

Public Properties

	Name	Description
	High (see page 3277)	Indicates the high bounds of the array.
	Length (see page 3278)	Represents the number of elements in the DynamicArray (see page 3268).
	Low (see page 3278)	Indicates the low bound of a dynamic array.

38.1.6.2 DynamicArray::~DynamicArray Destructor

Frees a DynamicArray (see page 3268) object.

Pascal

```
destructor Destroy;
```

C++

```
~DynamicArray();
```

Description

Do not call the dynamic array destructor. To free a dynamically allocated DynamicArray (see page 3268), assign it a Length (see page 3278) of zero. DynamicArray (see page 3268) instances on the stack are freed when they go out of scope.

38.1.6.3 DynamicArray Constructor

This is the overview for the DynamicArray constructor overload.

Overload List

	Name	Description
	DynamicArray::DynamicArray () (see page 3271)	Creates and initializes a System::DynamicArray::DynamicArray (see page 3268) object.
	DynamicArray::DynamicArray (DynamicArray<T>&) (see page 3271)	Creates and initializes a System::DynamicArray::DynamicArray (see page 3268) object.

38.1.6.3.1 DynamicArray::DynamicArray Constructor ()

Creates and initializes a System::DynamicArray::DynamicArray (see page 3268) object.

Pascal

```
constructor Create;
constructor Create(const src: DynamicArray<T>&);
```

C++

```
DynamicArray();
DynamicArray(const DynamicArray<T>& src);
```

Description

When called with no parameters, the constructor creates an empty dynamic array. Otherwise, it copies another System::DynamicArray::DynamicArray (see page 3268).

38.1.6.4 DynamicArray Methods

The methods of the DynamicArray class are listed here.

Protected Methods

	Name	Description
	AllocData (see page 3272)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 3274)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 3274)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 3275)	This is IncRefCount, a member of class DynamicArray.
	SetData (see page 3275)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	Copy (see page 3272)	This is the overview for the Copy method overload.
	CopyRange (see page 3273)	This is the overview for the CopyRange method overload.
	DimCount (see page 3274)	This is DimCount, a member of class DynamicArray.
	get_high (see page 3274)	This is get_high, a member of class DynamicArray.
	get_length (see page 3275)	This is get_length, a member of class DynamicArray.
	get_low (see page 3275)	This is get_low, a member of class DynamicArray.
	set_length (see page 3275)	This is set_length, a member of class DynamicArray.

38.1.6.4.1 DynamicArray::AllocData Method

Pascal

```
function AllocData(count: Integer): T*; static;
```

C++

```
static T* AllocData(int count);
```

Description

This is AllocData, a member of class DynamicArray.

38.1.6.4.2 Copy Method

This is the overview for the Copy method overload.

Overload List

	Name	Description
	DynamicArray::Copy () (see page 3273)	Copies the contents of a DynamicArray (see page 3268) to the specified DynamicArray (see page 3268).
	DynamicArray::Copy (DynamicArray<T> &) (see page 3273)	Copies the contents of a DynamicArray (see page 3268) to the specified DynamicArray (see page 3268).

38.1.6.4.2.1 DynamicArray::Copy Method ()

Copies the contents of a DynamicArray (see page 3268) to the specified DynamicArray (see page 3268).

Pascal

```
function Copy: DynamicArray<T>;
procedure Copy(var dst: DynamicArray<T>);
```

C++

```
DynamicArray<T> Copy() const;
void Copy(DynamicArray<T> & dst) const;
```

Description

Use System::DynamicArray::Copy to make a copy of a dynamic array. Dynamic arrays are reference counted. Therefore assigning one to another simply copies the reference, not the data. The System::DynamicArray::Copy method, on the other hand, makes a separate copy of the dynamic array. The original contents of the destination array are discarded.

The following code copies the contents of i_array into temp; temp and i_array must be of the same DynamicArray (see page 3268) type.

```
temp = i_array.Copy();
```

See Also

CopyRange

38.1.6.4.3 CopyRange Method

This is the overview for the CopyRange method overload.

Overload List

	Name	Description
	DynamicArray::CopyRange (DynamicArray<T> &, int, int) (see page 3273)	Copies a range of the contents of a DynamicArray (see page 3268) within a specified range.
	DynamicArray::CopyRange (int, int) (see page 3273)	Copies a range of the contents of a DynamicArray (see page 3268) within a specified range.

38.1.6.4.3.1 DynamicArray::CopyRange Method (DynamicArray<T> &, int, int)

Copies a range of the contents of a DynamicArray (see page 3268) within a specified range.

Pascal

```
function CopyRange(startIndex: Integer; count: Integer): DynamicArray<T>;
procedure CopyRange(var dst: DynamicArray<T>; startIndex: Integer; count: Integer);
```

C++

```
DynamicArray<T> CopyRange(int startIndex, int count) const;
void CopyRange(DynamicArray<T> & dst, int startIndex, int count) const;
```

Description

Use the System::DynamicArray::CopyRange method to copy a subrange of a dynamic array. The System::DynamicArray::CopyRange method is similar to the Copy (see page 3273) method except that you can specify the startIndex and the number of items to copy from the source object. The size of the destination array will be truncated or expanded to accommodate what has been copied.

newArray = array.System::DynamicArray::CopyRange(3, 5);

See Also

Copy

38.1.6.4.4 DynamicArray::DecRefCount Method

Pascal

```
procedure DecRefCount;
```

C++

```
void DecRefCount();
```

Description

This is DecRefCount, a member of class DynamicArray.

38.1.6.4.5 DynamicArray::DimCount Method

Pascal

```
function DimCount: Integer; static;
```

C++

```
static int DimCount();
```

Description

This is DimCount, a member of class DynamicArray.

38.1.6.4.6 DynamicArray::FreeData Method

Pascal

```
procedure FreeData;
```

C++

```
void FreeData();
```

Description

This is FreeData, a member of class DynamicArray.

38.1.6.4.7 DynamicArray::get_high Method

Pascal

```
function get_high: Integer;
```

C++

```
int get_high() const;
```

Description

This is get_high, a member of class DynamicArray.

38.1.6.4.8 DynamicArray::get_length Method

Pascal

```
function get_length: Integer;
```

C++

```
int get_length() const;
```

Description

This is get_length, a member of class DynamicArray.

38.1.6.4.9 DynamicArray::get_low Method

Pascal

```
function get_low: Integer;
```

C++

```
int get_low() const;
```

Description

This is get_low, a member of class DynamicArray.

38.1.6.4.10 DynamicArray::IncRefCount Method

Pascal

```
procedure IncRefCount;
```

C++

```
void IncRefCount();
```

Description

This is IncRefCount, a member of class DynamicArray.

38.1.6.4.11 DynamicArray::set_length Method

Pascal

```
procedure set_length(l: Integer);
```

C++

```
void set_length(int l);
```

Description

This is set_length, a member of class DynamicArray.

38.1.6.4.12 DynamicArray::SetData Method

Pascal

```
procedure SetData(var t: T);
```

C++

```
void SetData(T* t);
```

Description

This is SetData, a member of class DynamicArray.

38.1.6.5 DynamicArray Operators

The operators of the DynamicArray class are listed here.

Public Operators

	Name	Description
	[] (see page 3276)	This is the overview for the [] operator overload.
	= (see page 3276)	Assigns data from another Dynamic Array.
	== (see page 3277)	Compares two Dynamic Arrays for equality.

38.1.6.5.1 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	DynamicArray::[] (int) (see page 3276)	Provides access to the data held by the dynamic array.
	DynamicArray::[] (int) (see page 3276)	Provides access to the data held by the dynamic array.

38.1.6.5.1.1 DynamicArray::[] Operator (int)

Provides access to the data held by the dynamic array.

Pascal

```
operator [](index: Integer);
operator [](index: Integer);
```

C++

```
T& operator [](int index);
T operator [](int index) const;
```

Description

The const version of the System::DynamicArray (see page 3268)::[] returns a copy of the data at the specified subscript. The non-const version returns a reference to the data, allowing you to modify the data.

cout << i_array[0]=10;

38.1.6.5.2 DynamicArray::= Operator

Assigns data from another Dynamic Array.

Pascal

```
operator =(const src: DynamicArray<T>&);
```

C++

```
DynamicArray& operator =(const DynamicArray<T>& src);
```

Description

Dynamic Arrays are reference counted. When a DynamicArray (see page 3268) is assigned to another, only the reference is assigned (and the reference count adjusted), the contents of the source are not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To create a separate copy of the contents of a DynamicArray (see page 3268), use the Copy (see page 3273) (or CopyRange (see page 3273)) method.

See Also

Copy

38.1.6.5.3 DynamicArray::== Operator

Compares two Dynamic Arrays for equality.

Pascal

```
operator ==(const other: DynamicArray<T> &);
```

C++

```
bool operator ==(const DynamicArray<T> & other) const;
```

Description

The == System::DynamicArray (see page 3268)::== determines whether two dynamic arrays refer to the same memory. It does not compare the contents of the arrays.

38.1.6.6 DynamicArray Properties

The properties of the DynamicArray class are listed here.

Public Properties

	Name	Description
R	High (see page 3277)	Indicates the high bounds of the array.
	Length (see page 3278)	Represents the number of elements in the DynamicArray (see page 3268).
R	Low (see page 3278)	Indicates the low bound of a dynamic array.

38.1.6.6.1 DynamicArray::High Property

Indicates the high bounds of the array.

Pascal

```
property High: Integer;
```

C++

```
__property int High;
```

Description

Reading the High property returns Length (see page 3278)-1.

The following function returns the sum of the contents of an integer type DynamicArray (see page 3268).

int TotalArray(const DynamicArray (see page 3268)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
```

```
return total;
}
```

See Also

Length

Low

38.1.6.6.2 DynamicArray::Length Property

Represents the number of elements in the DynamicArray (see page 3268).

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

Use the Length property to set or get the length of the dynamic array. The following code sets the length of a DynamicArray (see page 3268) to 10.

DynamicArray (see page 3268)<int> arrayOfInt;

```
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

SetLength (see page 3748)(arrayOfInt, 10);

To free a dynamic array, simply set its Length to 0:

arrayOfInt.Length = 0;

See Also

High

38.1.6.6.3 DynamicArray::Low Property

Indicates the low bound of a dynamic array.

Pascal

```
property Low: Integer;
```

C++

```
__property int Low;
```

Description

The Low property always returns 0. The following function returns the sum of the contents of an integer type DynamicArray (see page 3268).

int TotalArray(const DynamicArray (see page 3268)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

See Also

High

38.1.7 DynArrayException Class

Class Hierarchy

Pascal

```
DynArrayException = class;
```

C++

```
class DynArrayException;
```

File

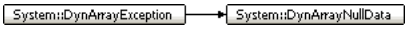
sysdyn.h

Description

This is class System::DynArrayException.

38.1.8 DynArrayNullData Class

Class Hierarchy



Pascal

```
DynArrayNullData = class(DynArrayException);
```

C++

```
class DynArrayNullData : public DynArrayException;
```

File

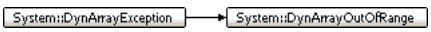
sysdyn.h

Description

This is class System::DynArrayNullData.

38.1.9 DynArrayOutOfRange Class

Class Hierarchy



Pascal

```
DynArrayOutOfRange = class(DynArrayException);
```

C++

```
class DynArrayOutOfRange : public DynArrayException;
```

File

sysdyn.h

Description

This is class System::DynArrayOutOfRange.

38.1.9.1 DynArrayOutOfRange Members

The following tables list the members exposed by DynArrayOutOfRange.

Public Data Members

	Name	Description
	m_Count (see page 3280)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 3281)	This is m_Index, a member of class DynArrayOutOfRange.

Public Methods

	Name	Description
	DynArrayOutOfRange (see page 3280)	This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

38.1.9.2 DynArrayOutOfRange::DynArrayOutOfRange Constructor

Pascal

```
constructor Create(index: Integer; count: Integer);
```

C++

```
DynArrayOutOfRange(int index, int count);
```

Description

This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

38.1.9.3 DynArrayOutOfRange Data Members

The data members of the DynArrayOutOfRange class are listed here.

Public Data Members

	Name	Description
	m_Count (see page 3280)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 3281)	This is m_Index, a member of class DynArrayOutOfRange.

38.1.9.3.1 DynArrayOutOfRange::m_Count Data Member

Pascal

```
m_Count: Integer;
```

C++

```
int m_Count;
```

Description

This is m_Count, a member of class DynArrayOutOfRange.

38.1.9.3.2 DynArrayOutOfRange::m_Index Data Member

Pascal

```
m_Index: Integer;
```

C++

```
int m_Index;
```

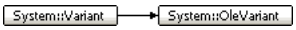
Description

This is m_Index, a member of class DynArrayOutOfRange.

38.1.10 OleVariant Class

System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.

Class Hierarchy



Pascal

```
OleVariant = class(Variant);
```

C++

```
class OleVariant : protected Variant;
```

File

sysvari.h

Description

System::OleVariant is a special descendant of the Variant (see page 3430) class that is used to represent COM interfaces or data that is passed over a COM interface. System::OleVariant inherits from Variant (see page 3430) as a protected class so that it can prevent operations on AnsiString (see page 3468) objects, which are not compatible with COM.

The syntax for using System::OleVariant is the same as that for using the Variant (see page 3430) class. Like the Variant (see page 3430) class, System::OleVariant has the following characteristics:

System::OleVariants can contain integer values, real values, string values, boolean values, date-and-time values, and Automation objects. In addition, System::OleVariants can contain arrays of varying size and dimension with elements of any of these types.

The special System::OleVariant value Unassigned is used to indicate that a System::OleVariant has not yet been assigned a value, and the special System::OleVariant value Null is used to indicate unknown or missing data.

A System::OleVariant can be combined with other System::OleVariants and it can be constructed from any of the following data types. The compiler automatically performs the necessary type conversions.

shortWordBool

intByte

floatVariant

doubleWideString

Currencywchar_t*

TDateTimeOle2::IDispatch (see page 3498)*

boolOle2::IUnknown (see page 3828)*

When an System::OleVariant contains an Automation object, the System::OleVariant can be used to get and set properties of the object, and to invoke methods on the object.

System::OleVariant variables are always initialized to be Unassigned when they are first created. This is true whether a System::OleVariant variable is global, local, or part of a structure such as an array.

Note that while System::OleVariants offer great flexibility, they also consume more memory than regular variables, and operations on System::OleVariants are substantially slower than operations on statically typed values.

Note: Use only the operators declared within System::OleVariant. The compiler will ignore any operators you overload yourself.

Note: For binary operators, if one operand is of type System::OleVariant, the other operand is automatically converted to type System::OleVariant.

See Also

System::Variant (see page 3430)

TVariantT

38.1.10.1 OleVariant Members

The following tables list the members exposed by OleVariant.

Public Data Members

	Name	Description
	TVarData::Reserved1 (see page 3298)	This is TVarData::Reserved1, a member of class OleVariant.
	TVarData::Reserved2 (see page 3298)	This is TVarData::Reserved2, a member of class OleVariant.
	TVarData::Reserved3 (see page 3298)	This is TVarData::Reserved3, a member of class OleVariant.
	TVarData::VArray (see page 3298)	This is TVarData::VArray, a member of class OleVariant.
	TVarData::VBoolean (see page 3299)	This is TVarData::VBoolean, a member of class OleVariant.
	TVarData::VByte (see page 3299)	This is TVarData::VByte, a member of class OleVariant.
	TVarData::VCurrency (see page 3299)	This is TVarData::VCurrency, a member of class OleVariant.
	TVarData::VDate (see page 3299)	This is TVarData::VDate, a member of class OleVariant.
	TVarData::VDispatch (see page 3299)	This is TVarData::VDispatch, a member of class OleVariant.
	TVarData::VDouble (see page 3300)	This is TVarData::VDouble, a member of class OleVariant.
	TVarData::VError (see page 3300)	This is TVarData::VError, a member of class OleVariant.
	TVarData::VInteger (see page 3300)	This is TVarData::VInteger, a member of class OleVariant.
	TVarData::VOleStr (see page 3300)	This is TVarData::VOleStr, a member of class OleVariant.
	TVarData::VPointer (see page 3301)	This is TVarData::VPointer, a member of class OleVariant.
	TVarData::VSingle (see page 3301)	This is TVarData::VSingle, a member of class OleVariant.
	TVarData::VSmallint (see page 3301)	This is TVarData::VSmallint, a member of class OleVariant.
	TVarData::VType (see page 3301)	This is TVarData::VType, a member of class OleVariant.
	TVarData::VUnknown (see page 3301)	This is TVarData::VUnknown, a member of class OleVariant.
	Variant::- (see page 3302)	This is Variant::-, a member of class OleVariant.
	Variant::! (see page 3302)	This is Variant::!, a member of class OleVariant.
	Variant::%= (see page 3302)	This is Variant::%=, a member of class OleVariant.
	Variant::&= (see page 3302)	This is Variant::&=, a member of class OleVariant.

	Variant::*= (see page 3303)	This is Variant::*=, a member of class OleVariant.
	Variant::/= (see page 3303)	This is Variant::/=, a member of class OleVariant.
	Variant::^= (see page 3303)	This is Variant::^=, a member of class OleVariant.
	Variant::__int64 (see page 3303)	This is Variant::__int64, a member of class OleVariant.
	Variant::|= (see page 3303)	This is Variant::|=, a member of class OleVariant.
	Variant::+= (see page 3304)	This is Variant::+=, a member of class OleVariant.
	Variant::<<= (see page 3304)	This is Variant::<<=, a member of class OleVariant.
	Variant::-= (see page 3304)	This is Variant::-=, a member of class OleVariant.
	Variant::>>= (see page 3304)	This is Variant::>>=, a member of class OleVariant.
	Variant::AnsiString (see page 3305)	This is Variant::AnsiString, a member of class OleVariant.
	Variant::ArrayDimCount (see page 3305)	This is Variant::ArrayDimCount, a member of class OleVariant.
	Variant::ArrayHighBound (see page 3305)	This is Variant::ArrayHighBound, a member of class OleVariant.
	Variant::ArrayLock (see page 3305)	This is Variant::ArrayLock, a member of class OleVariant.
	Variant::ArrayLowBound (see page 3305)	This is Variant::ArrayLowBound, a member of class OleVariant.
	Variant::ArrayRedim (see page 3306)	This is Variant::ArrayRedim, a member of class OleVariant.
	Variant::ArrayUnlock (see page 3306)	This is Variant::ArrayUnlock, a member of class OleVariant.
	Variant::AsType (see page 3306)	This is Variant::AsType, a member of class OleVariant.
	Variant::bool (see page 3306)	This is Variant::bool, a member of class OleVariant.
	Variant::Byte* (see page 3307)	This is Variant::Byte*, a member of class OleVariant.
	Variant::ChangeType (see page 3307)	This is Variant::ChangeType, a member of class OleVariant.
	Variant::char (see page 3307)	This is Variant::char, a member of class OleVariant.
	Variant::Clear (see page 3307)	This is Variant::Clear, a member of class OleVariant.
	Variant::Currency (see page 3307)	This is Variant::Currency, a member of class OleVariant.
	Variant::Currency* (see page 3308)	This is Variant::Currency*, a member of class OleVariant.
	Variant::double (see page 3308)	This is Variant::double, a member of class OleVariant.
	Variant::double* (see page 3308)	This is Variant::double*, a member of class OleVariant.
	Variant::float (see page 3308)	This is Variant::float, a member of class OleVariant.
	Variant::float* (see page 3309)	This is Variant::float*, a member of class OleVariant.
	Variant::GetElement (see page 3309)	This is Variant::GetElement, a member of class OleVariant.
	Variant::GetError (see page 3309)	This is Variant::GetError, a member of class OleVariant.
	Variant::IDispatch* (see page 3309)	This is Variant::IDispatch*, a member of class OleVariant.
	Variant::IInterface* (see page 3309)	This is Variant::IInterface*, a member of class OleVariant.
	Variant::int (see page 3310)	This is Variant::int, a member of class OleVariant.
	Variant::int* (see page 3310)	This is Variant::int*, a member of class OleVariant.
	Variant::IsArray (see page 3310)	This is Variant::IsArray, a member of class OleVariant.
	Variant::IsEmpty (see page 3310)	This is Variant::IsEmpty, a member of class OleVariant.
	Variant::IsNull (see page 3311)	This is Variant::IsNull, a member of class OleVariant.
	Variant::IUnknown* (see page 3311)	This is Variant::IUnknown*, a member of class OleVariant.
	Variant::long (see page 3311)	This is Variant::long, a member of class OleVariant.
	Variant::long* (see page 3311)	This is Variant::long*, a member of class OleVariant.
	Variant::PutElement (see page 3311)	This is Variant::PutElement, a member of class OleVariant.
	Variant::SetError (see page 3312)	This is Variant::SetError, a member of class OleVariant.
	Variant::short (see page 3312)	This is Variant::short, a member of class OleVariant.
	Variant::short* (see page 3312)	This is Variant::short*, a member of class OleVariant.
	Variant::signed char (see page 3312)	This is Variant::signed char, a member of class OleVariant.
	Variant::signed char* (see page 3313)	This is Variant::signed char*, a member of class OleVariant.
	Variant::TDateTime (see page 3313)	This is Variant::TDateTime, a member of class OleVariant.
	Variant::TDateTime* (see page 3313)	This is Variant::TDateTime*, a member of class OleVariant.
	Variant::Type (see page 3313)	This is Variant::Type, a member of class OleVariant.
	Variant::UnicodeString (see page 3313)	This is Variant::UnicodeString, a member of class OleVariant.
	Variant::unsigned char (see page 3314)	This is Variant::unsigned char, a member of class OleVariant.
	Variant::unsigned int (see page 3314)	This is Variant::unsigned int, a member of class OleVariant.
	Variant::unsigned int* (see page 3314)	This is Variant::unsigned int*, a member of class OleVariant.
	Variant::unsigned long (see page 3314)	This is Variant::unsigned long, a member of class OleVariant.

	Variant::unsigned short (see page 3315)	This is Variant::unsigned short, a member of class OleVariant.
	Variant::unsigned short* (see page 3315)	This is Variant::unsigned short*, a member of class OleVariant.
	Variant::WideString (see page 3315)	This is Variant::WideString, a member of class OleVariant.

Public Methods

	Name	Description
	~OleVariant (see page 3286)	Frees an instance of System::OleVariant::OleVariant (see page 3281).
	~Variant (see page 3433)	Frees a System::Variant::Variant (see page 3430) object.
	ArrayDimCount (see page 3437)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3438)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3438)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3438)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3439)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3439)	Unlocks a variant array that was previously locked by ArrayLock (see page 3438).
	AsType (see page 3439)	Casts a System::Variant::Variant (see page 3430) object to the type specified by the argument VarType.
	ChangeType (see page 3439)	Changes the data type of a System::Variant::Variant (see page 3430) object once it has been instantiated.
	Clear (see page 3440)	System::Variant::Clears the value of the System::Variant::Variant (see page 3430).
	Empty (see page 3440)	This is Empty, a member of class Variant.
	GetBaseVariant (see page 3440)	Returns the Variant (see page 3430) that is assigned as the value of this Variant (see page 3430) when Type (see page 3444) is varVariant (see page 3941).
	GetElement (see page 3441)	This is the overview for the GetElement method overload.
	GetError (see page 3441)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and returns the integer value.
	IsArray (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varArray (see page 3929) bit.
	IsEmpty (see page 3442)	Queries the VType of the base class TVarData (see page 3813).
	IsNull (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varNull (see page 3936) bit.
	NoParam (see page 3443)	This is NoParam, a member of class Variant.
	OleVariant (see page 3286)	This is the overview for the OleVariant constructor overload.
	PutElement (see page 3443)	This is the overview for the PutElement method overload.
	Sanitized (see page 3444)	This is Sanitized, a member of class Variant.
	SetError (see page 3444)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and assigns the integer err as the value of the varError (see page 3934).
	Type (see page 3444)	This is Type, a member of class Variant.
	Variant (see page 3434)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 3446)	This is the overview for the - operator overload.
	! (see page 3447)	Returns a new System::Variant::Variant (see page 3430) that is the logical negation of this System::Variant::Variant (see page 3430).
	!= (see page 3315)	This is the overview for the != operator overload.
	!= (see page 3448)	This is the overview for the != operator overload.
	% (see page 3448)	This is the overview for the % operator overload.
	%= (see page 3449)	Returns this System::Variant::Variant (see page 3430) as the remainder from dividing its original value and the System::Variant::Variant (see page 3430) specified by rhs.
	& (see page 3449)	This is the overview for the & operator overload.
	&= (see page 3450)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise and operation.
	* (see page 3450)	This is the overview for the * operator overload.

	*= (see page 3451)	Multiplies this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) specified by rhs.
	/ (see page 3452)	This is the overview for the / operator overload.
	/= (see page 3452)	Divides this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) rhs.
	^ (see page 3453)	This is the overview for the ^ operator overload.
	^= (see page 3453)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise xor operation.
	__int64 (see page 3454)	This is __int64, a member of class Variant.
	| (see page 3454)	This is the overview for the | operator overload.
	|= (see page 3454)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise or operation.
	+ (see page 3455)	This is the overview for the + operator overload.
	+= (see page 3455)	Adds the System::Variant::Variant (see page 3430) specified by rhs to this System::Variant::Variant (see page 3430) object.
	< (see page 3316)	This is the overview for the < operator overload.
	< (see page 3456)	This is the overview for the < operator overload.
	<< (see page 3456)	This is the overview for the << operator overload.
	<<= (see page 3458)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3430).
	<= (see page 3317)	This is the overview for the <= operator overload.
	<= (see page 3458)	This is the overview for the <= operator overload.
	-= (see page 3463)	Subtracts the System::Variant::Variant (see page 3430) specified by rhs from this System::Variant::Variant (see page 3430) object.
	= (see page 3317)	This is the overview for the = operator overload.
	= (see page 3459)	This is the overview for the = operator overload.
	== (see page 3320)	This is the overview for the == operator overload.
	== (see page 3463)	This is the overview for the == operator overload.
	> (see page 3320)	This is the overview for the > operator overload.
	> (see page 3464)	This is the overview for the > operator overload.
	>= (see page 3321)	This is the overview for the >= operator overload.
	>= (see page 3465)	This is the overview for the >= operator overload.
	>> (see page 3467)	This is the overview for the >> operator overload.
	>>= (see page 3468)	This is >>=, a member of class Variant.
	AnsiString (see page 3468)	Converts the System::Variant::Variant (see page 3430) to an AnsiString object.
	bool (see page 3468)	Converts the System::Variant::Variant (see page 3430) to a boolean value.
	char (see page 3468)	This is char, a member of class Variant.
	Currency (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a Currency object.
	Currency* (see page 3469)	This is Currency*, a member of class Variant.
	double (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a double value.
	double* (see page 3470)	This is double*, a member of class Variant.
	float (see page 3470)	Converts the System::Variant::Variant (see page 3430) to a float value.
	float* (see page 3470)	This is float*, a member of class Variant.
	IDispatch* (see page 3470)	This is IDispatch*, a member of class Variant.
	IInterface* (see page 3471)	This is IInterface*, a member of class Variant.
	int (see page 3471)	Converts the System::Variant::Variant (see page 3430) to an integer value.
	int* (see page 3471)	This is int*, a member of class Variant.
	IUnknown* (see page 3471)	This is IUnknown*, a member of class Variant.
	long (see page 3472)	This is long, a member of class Variant.
	long* (see page 3472)	This is long*, a member of class Variant.
	short (see page 3472)	Converts the System::Variant::Variant (see page 3430) to a short value.
	short* (see page 3472)	This is short*, a member of class Variant.
	signed char (see page 3473)	This is signed char, a member of class Variant.
	signed char* (see page 3473)	This is signed char*, a member of class Variant.
	TDateTime (see page 3473)	Converts the System::Variant::Variant (see page 3430) to a TDateTime object.
	TDateTime* (see page 3473)	This is TDateTime*, a member of class Variant.

	UnicodeString (see page 3474)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3474)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 3474)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 3474)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3474)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3475)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3475)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3475)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 3475)	This is unsigned short*, a member of class Variant.
	WideString (see page 3476)	This is WideString, a member of class Variant.

38.1.10.2 OleVariant::~OleVariant Destructor

Frees an instance of System::OleVariant::OleVariant (see page 3281).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~OleVariant();
```

Description

~System::OleVariant::OleVariant (see page 3281) is the destructor for System::OleVariant::OleVariant (see page 3281).

38.1.10.3 OleVariant Constructor

This is the overview for the OleVariant constructor overload.

Overload List

	Name	Description
	OleVariant::OleVariant () (see page 3288)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (__int64) (see page 3291)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (__int64*) (see page 3291)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (AnsiString&) (see page 3288)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (bool) (see page 3291)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (char) (see page 3291)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (char*) (see page 3292)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Currency) (see page 3288)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Currency*) (see page 3288)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (double) (see page 3292)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (double*) (see page 3292)	This is OleVariant, a member of class OleVariant.

	OleVariant::OleVariant (float) (see page 3292)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (float*) (see page 3293)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IDispatch* const) (see page 3289)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IInterface* const) (see page 3289)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int) (see page 3293)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int*) (see page 3293)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int*, int, Word) (see page 3293)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IUnknown* const) (see page 3289)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (long*) (see page 3293)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (OleVariant&) (see page 3289)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (OleVariant*, int) (see page 3289)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (short) (see page 3294)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (short*) (see page 3294)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (signed char*) (see page 3294)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (TDateTime) (see page 3290)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (TDateTime*) (see page 3290)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (UnicodeString&) (see page 3290)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned char) (see page 3294)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned char*) (see page 3295)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned int) (see page 3295)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned int*) (see page 3295)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned long*) (see page 3295)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned short) (see page 3295)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned short*) (see page 3296)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Variant&) (see page 3290)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (wchar_t* const, bool) (see page 3296)	This is OleVariant, a member of class OleVariant.

	OleVariant::OleVariant (WideString&) (see page 3291)	This is OleVariant, a member of class OleVariant.

38.1.10.3.1 OleVariant::OleVariant Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall OleVariant();
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.2 OleVariant::OleVariant Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall OleVariant(const AnsiString& src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.3 OleVariant::OleVariant Constructor (Currency)

Pascal

```
constructor Create(const src: Currency);
```

C++

```
__fastcall OleVariant(const Currency src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.4 OleVariant::OleVariant Constructor (Currency*)

Pascal

```
constructor Create(var src: Currency);
```

C++

```
__fastcall OleVariant(Currency* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.5 OleVariant::OleVariant Constructor (IDispatch* const)

Pascal

```
constructor Create(src: IDispatch* const);
```

C++

```
__fastcall OleVariant(IDispatch* const src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.6 OleVariant::OleVariant Constructor (IInterface* const)

Pascal

```
constructor Create(src: IInterface* const);
```

C++

```
__fastcall OleVariant(IInterface* const src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.7 OleVariant::OleVariant Constructor (IUnknown* const)

Pascal

```
constructor Create(src: IUnknown* const);
```

C++

```
__fastcall OleVariant(IUnknown* const src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.8 OleVariant::OleVariant Constructor (OleVariant&)

Pascal

```
constructor Create(const src: OleVariant&);
```

C++

```
__fastcall OleVariant(const OleVariant& src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.9 OleVariant::OleVariant Constructor (OleVariant*, int)

Pascal

```
constructor Create(const values: OleVariant*; const valuesSize: Integer);
```

C++

```
__fastcall OleVariant(const OleVariant* values, const int valuesSize);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.10 OleVariant::OleVariant Constructor (TDateTime)

Pascal

```
constructor Create(const src: TDateTime);
```

C++

```
__fastcall OleVariant(const TDateTime src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.11 OleVariant::OleVariant Constructor (TDateTime*)

Pascal

```
constructor Create(var src: TDateTime);
```

C++

```
__fastcall OleVariant(TDateTime* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.12 OleVariant::OleVariant Constructor (UnicodeString&)

Pascal

```
constructor Create(const src: UnicodeString&);
```

C++

```
__fastcall OleVariant(const UnicodeString& src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.13 OleVariant::OleVariant Constructor (Variant&)

Pascal

```
constructor Create(const src: Variant&);
```

C++

```
__fastcall OleVariant(const Variant& src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.14 OleVariant::OleVariant Constructor (WideString&)

Pascal

```
constructor Create(const src: WideString&);
```

C++

```
__fastcall OleVariant(const WideString& src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.15 OleVariant::OleVariant Constructor (__int64)

Pascal

```
constructor Create(const src: __int64);
```

C++

```
__fastcall OleVariant(const __int64 src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.16 OleVariant::OleVariant Constructor (__int64*)

Pascal

```
constructor Create(var src: __int64);
```

C++

```
__fastcall OleVariant(__int64* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.17 OleVariant::OleVariant Constructor (bool)

Pascal

```
constructor Create(const src: bool);
```

C++

```
__fastcall OleVariant(const bool src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.18 OleVariant::OleVariant Constructor (char)

Pascal

```
constructor Create(const src: char);
```

C++

```
__fastcall OleVariant(const char src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.19 OleVariant::OleVariant Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall OleVariant(const char* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.20 OleVariant::OleVariant Constructor (double)

Pascal

```
constructor Create(const src: double);
```

C++

```
__fastcall OleVariant(const double src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.21 OleVariant::OleVariant Constructor (double*)

Pascal

```
constructor Create(var src: double);
```

C++

```
__fastcall OleVariant(double* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.22 OleVariant::OleVariant Constructor (float)

Pascal

```
constructor Create(const src: float);
```

C++

```
__fastcall OleVariant(const float src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.23 OleVariant::OleVariant Constructor (float*)

Pascal

```
constructor Create(var src: float);
```

C++

```
__fastcall OleVariant(float* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.24 OleVariant::OleVariant Constructor (int)

Pascal

```
constructor Create(const src: Integer);
```

C++

```
__fastcall OleVariant(const int src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.25 OleVariant::OleVariant Constructor (int*)

Pascal

```
constructor Create(var src: Integer);
```

C++

```
__fastcall OleVariant(int* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.26 OleVariant::OleVariant Constructor (int*, int, Word)

Pascal

```
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
```

C++

```
__fastcall OleVariant(const int* bounds, const int boundsSize, Word varType);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.27 OleVariant::OleVariant Constructor (long*)

Pascal

```
constructor Create(var src: long);
```

C++

```
__fastcall OleVariant(long* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.28 OleVariant::OleVariant Constructor (short)

Pascal

```
constructor Create(const src: short);
```

C++

```
__fastcall OleVariant(const short src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.29 OleVariant::OleVariant Constructor (short*)

Pascal

```
constructor Create(var src: short);
```

C++

```
__fastcall OleVariant(short* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.30 OleVariant::OleVariant Constructor (signed char*)

Pascal

```
constructor Create(var src: signed char);
```

C++

```
__fastcall OleVariant(signed char* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.31 OleVariant::OleVariant Constructor (unsigned char)

Pascal

```
constructor Create(const src: unsigned char);
```

C++

```
__fastcall OleVariant(const unsigned char src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.32 OleVariant::OleVariant Constructor (unsigned char*)

Pascal

```
constructor Create(var src: unsigned char);
```

C++

```
__fastcall OleVariant(unsigned char* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.33 OleVariant::OleVariant Constructor (unsigned int)

Pascal

```
constructor Create(const src: unsigned int);
```

C++

```
__fastcall OleVariant(const unsigned int src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.34 OleVariant::OleVariant Constructor (unsigned int*)

Pascal

```
constructor Create(var src: unsigned int);
```

C++

```
__fastcall OleVariant(unsigned int* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.35 OleVariant::OleVariant Constructor (unsigned long*)

Pascal

```
constructor Create(var src: unsigned long);
```

C++

```
__fastcall OleVariant(unsigned long* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.36 OleVariant::OleVariant Constructor (unsigned short)

Pascal

```
constructor Create(const src: unsigned short);
```

C++

```
__fastcall OleVariant(const unsigned short src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.37 OleVariant::OleVariant Constructor (unsigned short*)

Pascal

```
constructor Create(var src: unsigned short);
```

C++

```
__fastcall OleVariant(unsigned short* src);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.3.38 OleVariant::OleVariant Constructor (wchar_t* const, bool)

Pascal

```
constructor Create(src: wchar_t* const; isBSTR: bool = true);
```

C++

```
__fastcall OleVariant(wchar_t* const src, bool isBSTR = true);
```

Description

This is OleVariant, a member of class OleVariant.

38.1.10.4 OleVariant Data Members

The data members of the OleVariant class are listed here.

Public Data Members

	Name	Description
	TVarData::Reserved1 (see page 3298)	This is TVarData::Reserved1, a member of class OleVariant.
	TVarData::Reserved2 (see page 3298)	This is TVarData::Reserved2, a member of class OleVariant.
	TVarData::Reserved3 (see page 3298)	This is TVarData::Reserved3, a member of class OleVariant.
	TVarData::VArray (see page 3298)	This is TVarData::VArray, a member of class OleVariant.
	TVarData::VBoolean (see page 3299)	This is TVarData::VBoolean, a member of class OleVariant.
	TVarData::VByte (see page 3299)	This is TVarData::VByte, a member of class OleVariant.
	TVarData::VCurrency (see page 3299)	This is TVarData::VCurrency, a member of class OleVariant.
	TVarData::VDate (see page 3299)	This is TVarData::VDate, a member of class OleVariant.
	TVarData::VDispatch (see page 3299)	This is TVarData::VDispatch, a member of class OleVariant.
	TVarData::VDouble (see page 3300)	This is TVarData::VDouble, a member of class OleVariant.
	TVarData::VError (see page 3300)	This is TVarData::VError, a member of class OleVariant.
	TVarData::VInteger (see page 3300)	This is TVarData::VInteger, a member of class OleVariant.
	TVarData::VOleStr (see page 3300)	This is TVarData::VOleStr, a member of class OleVariant.
	TVarData::VPointer (see page 3301)	This is TVarData::VPointer, a member of class OleVariant.
	TVarData::VSingle (see page 3301)	This is TVarData::VSingle, a member of class OleVariant.
	TVarData::VSmallint (see page 3301)	This is TVarData::VSmallint, a member of class OleVariant.
	TVarData::VType (see page 3301)	This is TVarData::VType, a member of class OleVariant.
	TVarData::VUnknown (see page 3301)	This is TVarData::VUnknown, a member of class OleVariant.
	Variant::- (see page 3302)	This is Variant::-, a member of class OleVariant.
	Variant::! (see page 3302)	This is Variant::!, a member of class OleVariant.

	Variant::%= (see page 3302)	This is Variant::%=, a member of class OleVariant.
	Variant::&= (see page 3302)	This is Variant::&=, a member of class OleVariant.
	Variant::*= (see page 3303)	This is Variant::*=, a member of class OleVariant.
	Variant::/= (see page 3303)	This is Variant::/=, a member of class OleVariant.
	Variant::^= (see page 3303)	This is Variant::^=, a member of class OleVariant.
	Variant::__int64 (see page 3303)	This is Variant::__int64, a member of class OleVariant.
	Variant::|= (see page 3303)	This is Variant::|=, a member of class OleVariant.
	Variant::+= (see page 3304)	This is Variant::+=, a member of class OleVariant.
	Variant::<<= (see page 3304)	This is Variant::<<=, a member of class OleVariant.
	Variant::-= (see page 3304)	This is Variant::-=, a member of class OleVariant.
	Variant::>>= (see page 3304)	This is Variant::>>=, a member of class OleVariant.
	Variant::AnsiString (see page 3305)	This is Variant::AnsiString, a member of class OleVariant.
	Variant::ArrayDimCount (see page 3305)	This is Variant::ArrayDimCount, a member of class OleVariant.
	Variant::ArrayHighBound (see page 3305)	This is Variant::ArrayHighBound, a member of class OleVariant.
	Variant::ArrayLock (see page 3305)	This is Variant::ArrayLock, a member of class OleVariant.
	Variant::ArrayLowBound (see page 3305)	This is Variant::ArrayLowBound, a member of class OleVariant.
	Variant::ArrayRedim (see page 3306)	This is Variant::ArrayRedim, a member of class OleVariant.
	Variant::ArrayUnlock (see page 3306)	This is Variant::ArrayUnlock, a member of class OleVariant.
	Variant::AsType (see page 3306)	This is Variant::AsType, a member of class OleVariant.
	Variant::bool (see page 3306)	This is Variant::bool, a member of class OleVariant.
	Variant::Byte* (see page 3307)	This is Variant::Byte*, a member of class OleVariant.
	Variant::ChangeType (see page 3307)	This is Variant::ChangeType, a member of class OleVariant.
	Variant::char (see page 3307)	This is Variant::char, a member of class OleVariant.
	Variant::Clear (see page 3307)	This is Variant::Clear, a member of class OleVariant.
	Variant::Currency (see page 3307)	This is Variant::Currency, a member of class OleVariant.
	Variant::Currency* (see page 3308)	This is Variant::Currency*, a member of class OleVariant.
	Variant::double (see page 3308)	This is Variant::double, a member of class OleVariant.
	Variant::double* (see page 3308)	This is Variant::double*, a member of class OleVariant.
	Variant::float (see page 3308)	This is Variant::float, a member of class OleVariant.
	Variant::float* (see page 3309)	This is Variant::float*, a member of class OleVariant.
	Variant::GetElement (see page 3309)	This is Variant::GetElement, a member of class OleVariant.
	Variant::GetError (see page 3309)	This is Variant::GetError, a member of class OleVariant.
	Variant::IDispatch* (see page 3309)	This is Variant::IDispatch*, a member of class OleVariant.
	Variant::IInterface* (see page 3309)	This is Variant::IInterface*, a member of class OleVariant.
	Variant::int (see page 3310)	This is Variant::int, a member of class OleVariant.
	Variant::int* (see page 3310)	This is Variant::int*, a member of class OleVariant.
	Variant::IsArray (see page 3310)	This is Variant::IsArray, a member of class OleVariant.
	Variant::IsEmpty (see page 3310)	This is Variant::IsEmpty, a member of class OleVariant.
	Variant::IsNull (see page 3311)	This is Variant::IsNull, a member of class OleVariant.
	Variant::IUnknown* (see page 3311)	This is Variant::IUnknown*, a member of class OleVariant.
	Variant::long (see page 3311)	This is Variant::long, a member of class OleVariant.
	Variant::long* (see page 3311)	This is Variant::long*, a member of class OleVariant.
	Variant::PutElement (see page 3311)	This is Variant::PutElement, a member of class OleVariant.
	Variant::SetError (see page 3312)	This is Variant::SetError, a member of class OleVariant.
	Variant::short (see page 3312)	This is Variant::short, a member of class OleVariant.
	Variant::short* (see page 3312)	This is Variant::short*, a member of class OleVariant.
	Variant::signed char (see page 3312)	This is Variant::signed char, a member of class OleVariant.
	Variant::signed char* (see page 3313)	This is Variant::signed char*, a member of class OleVariant.
	Variant::TDateTime (see page 3313)	This is Variant::TDateTime, a member of class OleVariant.
	Variant::TDateTime* (see page 3313)	This is Variant::TDateTime*, a member of class OleVariant.
	Variant::Type (see page 3313)	This is Variant::Type, a member of class OleVariant.
	Variant::UnicodeString (see page 3313)	This is Variant::UnicodeString, a member of class OleVariant.
	Variant::unsigned char (see page 3314)	This is Variant::unsigned char, a member of class OleVariant.
	Variant::unsigned int (see page 3314)	This is Variant::unsigned int, a member of class OleVariant.

	Variant::unsigned int* (see page 3314)	This is Variant::unsigned int*, a member of class OleVariant.
	Variant::unsigned long (see page 3314)	This is Variant::unsigned long, a member of class OleVariant.
	Variant::unsigned short (see page 3315)	This is Variant::unsigned short, a member of class OleVariant.
	Variant::unsigned short* (see page 3315)	This is Variant::unsigned short*, a member of class OleVariant.
	Variant::WideString (see page 3315)	This is Variant::WideString, a member of class OleVariant.

38.1.10.4.1 OleVariant::TVarData::Reserved1 Data Member

Pascal

```
TVarData::Reserved1: __USING;
```

C++

```
__USING TVarData::Reserved1;
```

Description

This is TVarData::Reserved1, a member of class OleVariant.

38.1.10.4.2 OleVariant::TVarData::Reserved2 Data Member

Pascal

```
TVarData::Reserved2: __USING;
```

C++

```
__USING TVarData::Reserved2;
```

Description

This is TVarData::Reserved2, a member of class OleVariant.

38.1.10.4.3 OleVariant::TVarData::Reserved3 Data Member

Pascal

```
TVarData::Reserved3: __USING;
```

C++

```
__USING TVarData::Reserved3;
```

Description

This is TVarData::Reserved3, a member of class OleVariant.

38.1.10.4.4 OleVariant::TVarData::VArray Data Member

Pascal

```
TVarData::VArray: __USING;
```

C++

```
__USING TVarData::VArray;
```

Description

This is TVarData::VArray, a member of class OleVariant.

38.1.10.4.5 OleVariant::TVarData::VBoolean Data Member

Pascal

```
TVarData::VBoolean: __USING;
```

C++

```
__USING TVarData::VBoolean;
```

Description

This is TVarData::VBoolean, a member of class OleVariant.

38.1.10.4.6 OleVariant::TVarData::VByte Data Member

Pascal

```
TVarData::VByte: __USING;
```

C++

```
__USING TVarData::VByte;
```

Description

This is TVarData::VByte, a member of class OleVariant.

38.1.10.4.7 OleVariant::TVarData::VCurrency Data Member

Pascal

```
TVarData::VCurrency: __USING;
```

C++

```
__USING TVarData::VCurrency;
```

Description

This is TVarData::VCurrency, a member of class OleVariant.

38.1.10.4.8 OleVariant::TVarData::VDate Data Member

Pascal

```
TVarData::VDate: __USING;
```

C++

```
__USING TVarData::VDate;
```

Description

This is TVarData::VDate, a member of class OleVariant.

38.1.10.4.9 OleVariant::TVarData::VDispatch Data Member

Pascal

```
TVarData::VDispatch: __USING;
```

C++

```
__USING TVarData::VDispatch;
```

Description

This is TVarData::VDispatch, a member of class OleVariant.

38.1.10.4.10 OleVariant::TVarData::VDouble Data Member

Pascal

```
TVarData::VDouble: __USING;
```

C++

```
__USING TVarData::VDouble;
```

Description

This is TVarData::VDouble, a member of class OleVariant.

38.1.10.4.11 OleVariant::TVarData::VError Data Member

Pascal

```
TVarData::VError: __USING;
```

C++

```
__USING TVarData::VError;
```

Description

This is TVarData::VError, a member of class OleVariant.

38.1.10.4.12 OleVariant::TVarData::VInteger Data Member

Pascal

```
TVarData::VInteger: __USING;
```

C++

```
__USING TVarData::VInteger;
```

Description

This is TVarData::VInteger, a member of class OleVariant.

38.1.10.4.13 OleVariant::TVarData::VOleStr Data Member

Pascal

```
TVarData::VOleStr: __USING;
```

C++

```
__USING TVarData::VOleStr;
```

Description

This is TVarData::VOleStr, a member of class OleVariant.

38.1.10.4.14 OleVariant::TVarData::VPointer Data Member

Pascal

```
TVarData::VPointer: __USING;
```

C++

```
__USING TVarData::VPointer;
```

Description

This is TVarData::VPointer, a member of class OleVariant.

38.1.10.4.15 OleVariant::TVarData::VSingle Data Member

Pascal

```
TVarData::VSingle: __USING;
```

C++

```
__USING TVarData::VSingle;
```

Description

This is TVarData::VSingle, a member of class OleVariant.

38.1.10.4.16 OleVariant::TVarData::VSmallint Data Member

Pascal

```
TVarData::VSmallint: __USING;
```

C++

```
__USING TVarData::VSmallint;
```

Description

This is TVarData::VSmallint, a member of class OleVariant.

38.1.10.4.17 OleVariant::TVarData::VType Data Member

Pascal

```
TVarData::VType: __USING;
```

C++

```
__USING TVarData::VType;
```

Description

This is TVarData::VType, a member of class OleVariant.

38.1.10.4.18 OleVariant::TVarData::VUnknown Data Member

Pascal

```
TVarData::VUnknown: __USING;
```

C++

```
__USING TVarData::VUnknown;
```

Description

This is TVarData::VUnknown, a member of class OleVariant.

38.1.10.4.19 OleVariant::Variant::- Data Member

Pascal

```
Variant::-: __USING;
```

C++

```
__USING Variant::-;
```

Description

This is Variant::-, a member of class OleVariant.

38.1.10.4.20 OleVariant::Variant::! Data Member

Pascal

```
Variant::!: __USING;
```

C++

```
__USING Variant::!;
```

Description

This is Variant::!, a member of class OleVariant.

38.1.10.4.21 OleVariant::Variant::%= Data Member

Pascal

```
Variant::%=: __USING;
```

C++

```
__USING Variant::%=;
```

Description

This is Variant::%=, a member of class OleVariant.

38.1.10.4.22 OleVariant::Variant::&= Data Member

Pascal

```
Variant::&=: __USING;
```

C++

```
__USING Variant::&=;
```

Description

This is Variant::&=, a member of class OleVariant.

38.1.10.4.23 OleVariant::Variant::*= Data Member

Pascal

```
Variant::*=: __USING;
```

C++

```
__USING Variant::*=;
```

Description

This is Variant::*=, a member of class OleVariant.

38.1.10.4.24 OleVariant::Variant::/= Data Member

Pascal

```
Variant::/=: __USING;
```

C++

```
__USING Variant::/=;
```

Description

This is Variant::/=, a member of class OleVariant.

38.1.10.4.25 OleVariant::Variant::^= Data Member

Pascal

```
Variant::^=: __USING;
```

C++

```
__USING Variant::^=;
```

Description

This is Variant::^=, a member of class OleVariant.

38.1.10.4.26 OleVariant::Variant::__int64 Data Member

Pascal

```
Variant::__int64: __USING;
```

C++

```
__USING Variant::__int64;
```

Description

This is Variant::__int64, a member of class OleVariant.

38.1.10.4.27 OleVariant::Variant::|= Data Member

Pascal

```
Variant::|=: __USING;
```

C++

```
__USING Variant::|=;
```

Description

This is Variant::|=, a member of class OleVariant.

38.1.10.4.28 OleVariant::Variant::+= Data Member

Pascal

```
Variant::+=: __USING;
```

C++

```
__USING Variant::+=;
```

Description

This is Variant::+=, a member of class OleVariant.

38.1.10.4.29 OleVariant::Variant::<<= Data Member

Pascal

```
Variant::<<=: __USING;
```

C++

```
__USING Variant::<<=;
```

Description

This is Variant::<<=, a member of class OleVariant.

38.1.10.4.30 OleVariant::Variant::-= Data Member

Pascal

```
Variant::-=: __USING;
```

C++

```
__USING Variant::-=;
```

Description

This is Variant::-=, a member of class OleVariant.

38.1.10.4.31 OleVariant::Variant::>>= Data Member

Pascal

```
Variant::>>=: __USING;
```

C++

```
__USING Variant::>>=;
```

Description

This is Variant::>>=, a member of class OleVariant.

38.1.10.4.32 OleVariant::Variant::AnsiString Data Member

Pascal

```
Variant::AnsiString: __USING;
```

C++

```
__USING Variant::AnsiString;
```

Description

This is Variant::AnsiString, a member of class OleVariant.

38.1.10.4.33 OleVariant::Variant::ArrayDimCount Data Member

Pascal

```
Variant::ArrayDimCount: __USING;
```

C++

```
__USING Variant::ArrayDimCount;
```

Description

This is Variant::ArrayDimCount, a member of class OleVariant.

38.1.10.4.34 OleVariant::Variant::ArrayHighBound Data Member

Pascal

```
Variant::ArrayHighBound: __USING;
```

C++

```
__USING Variant::ArrayHighBound;
```

Description

This is Variant::ArrayHighBound, a member of class OleVariant.

38.1.10.4.35 OleVariant::Variant::ArrayLock Data Member

Pascal

```
Variant::ArrayLock: __USING;
```

C++

```
__USING Variant::ArrayLock;
```

Description

This is Variant::ArrayLock, a member of class OleVariant.

38.1.10.4.36 OleVariant::Variant::ArrayLowBound Data Member

Pascal

```
Variant::ArrayLowBound: __USING;
```

C++

```
__USING Variant::ArrayLowBound;
```

Description

This is Variant::ArrayLowBound, a member of class OleVariant.

38.1.10.4.37 OleVariant::Variant::ArrayRedim Data Member

Pascal

```
Variant::ArrayRedim: __USING;
```

C++

```
__USING Variant::ArrayRedim;
```

Description

This is Variant::ArrayRedim, a member of class OleVariant.

38.1.10.4.38 OleVariant::Variant::ArrayUnlock Data Member

Pascal

```
Variant::ArrayUnlock: __USING;
```

C++

```
__USING Variant::ArrayUnlock;
```

Description

This is Variant::ArrayUnlock, a member of class OleVariant.

38.1.10.4.39 OleVariant::Variant::AsType Data Member

Pascal

```
Variant::AsType: __USING;
```

C++

```
__USING Variant::AsType;
```

Description

This is Variant::AsType, a member of class OleVariant.

38.1.10.4.40 OleVariant::Variant::bool Data Member

Pascal

```
Variant::bool: __USING;
```

C++

```
__USING Variant::bool;
```

Description

This is Variant::bool, a member of class OleVariant.

38.1.10.4.41 OleVariant::Variant::Byte* Data Member

Pascal

```
Variant::Byte*: __USING;
```

C++

```
__USING Variant::Byte*;
```

Description

This is Variant::Byte*, a member of class OleVariant.

38.1.10.4.42 OleVariant::Variant::ChangeType Data Member

Pascal

```
Variant::ChangeType: __USING;
```

C++

```
__USING Variant::ChangeType;
```

Description

This is Variant::ChangeType, a member of class OleVariant.

38.1.10.4.43 OleVariant::Variant::char Data Member

Pascal

```
Variant::char: __USING;
```

C++

```
__USING Variant::char;
```

Description

This is Variant::char, a member of class OleVariant.

38.1.10.4.44 OleVariant::Variant::Clear Data Member

Pascal

```
Variant::Clear: __USING;
```

C++

```
__USING Variant::Clear;
```

Description

This is Variant::Clear, a member of class OleVariant.

38.1.10.4.45 OleVariant::Variant::Currency Data Member

Pascal

```
Variant::Currency: __USING;
```

C++

```
__USING Variant::Currency;
```

Description

This is Variant::Currency, a member of class OleVariant.

38.1.10.4.46 OleVariant::Variant::Currency* Data Member

Pascal

```
Variant::Currency*: __USING;
```

C++

```
__USING Variant::Currency*;
```

Description

This is Variant::Currency*, a member of class OleVariant.

38.1.10.4.47 OleVariant::Variant::double Data Member

Pascal

```
Variant::double: __USING;
```

C++

```
__USING Variant::double;
```

Description

This is Variant::double, a member of class OleVariant.

38.1.10.4.48 OleVariant::Variant::double* Data Member

Pascal

```
Variant::double*: __USING;
```

C++

```
__USING Variant::double*;
```

Description

This is Variant::double*, a member of class OleVariant.

38.1.10.4.49 OleVariant::Variant::float Data Member

Pascal

```
Variant::float: __USING;
```

C++

```
__USING Variant::float;
```

Description

This is Variant::float, a member of class OleVariant.

38.1.10.4.50 OleVariant::Variant::float* Data Member

Pascal

```
Variant::float*: __USING;
```

C++

```
__USING Variant::float*;
```

Description

This is Variant::float*, a member of class OleVariant.

38.1.10.4.51 OleVariant::Variant::GetElement Data Member

Pascal

```
Variant::GetElement: __USING;
```

C++

```
__USING Variant::GetElement;
```

Description

This is Variant::GetElement, a member of class OleVariant.

38.1.10.4.52 OleVariant::Variant::GetError Data Member

Pascal

```
Variant::GetError: __USING;
```

C++

```
__USING Variant::GetError;
```

Description

This is Variant::GetError, a member of class OleVariant.

38.1.10.4.53 OleVariant::Variant::IDispatch* Data Member

Pascal

```
Variant::IDispatch*: __USING;
```

C++

```
__USING Variant::IDispatch*;
```

Description

This is Variant::IDispatch*, a member of class OleVariant.

38.1.10.4.54 OleVariant::Variant::IInterface* Data Member

Pascal

```
Variant::IInterface*: __USING;
```

C++

```
__USING Variant::IInterface*;
```

Description

This is Variant::IInterface*, a member of class OleVariant.

38.1.10.4.55 OleVariant::Variant::int Data Member

Pascal

```
Variant::int: __USING;
```

C++

```
__USING Variant::int;
```

Description

This is Variant::int, a member of class OleVariant.

38.1.10.4.56 OleVariant::Variant::int* Data Member

Pascal

```
Variant::int*: __USING;
```

C++

```
__USING Variant::int*;
```

Description

This is Variant::int*, a member of class OleVariant.

38.1.10.4.57 OleVariant::Variant::IsArray Data Member

Pascal

```
Variant::IsArray: __USING;
```

C++

```
__USING Variant::IsArray;
```

Description

This is Variant::IsArray, a member of class OleVariant.

38.1.10.4.58 OleVariant::Variant::IsEmpty Data Member

Pascal

```
Variant::IsEmpty: __USING;
```

C++

```
__USING Variant::IsEmpty;
```

Description

This is Variant::IsEmpty, a member of class OleVariant.

38.1.10.4.59 OleVariant::Variant::IsNull Data Member

Pascal

```
Variant::IsNull: __USING;
```

C++

```
__USING Variant::IsNull;
```

Description

This is Variant::IsNull, a member of class OleVariant.

38.1.10.4.60 OleVariant::Variant::IUnknown* Data Member

Pascal

```
Variant::IUnknown*: __USING;
```

C++

```
__USING Variant::IUnknown*;
```

Description

This is Variant::IUnknown*, a member of class OleVariant.

38.1.10.4.61 OleVariant::Variant::long Data Member

Pascal

```
Variant::long: __USING;
```

C++

```
__USING Variant::long;
```

Description

This is Variant::long, a member of class OleVariant.

38.1.10.4.62 OleVariant::Variant::long* Data Member

Pascal

```
Variant::long*: __USING;
```

C++

```
__USING Variant::long*;
```

Description

This is Variant::long*, a member of class OleVariant.

38.1.10.4.63 OleVariant::Variant::PutElement Data Member

Pascal

```
Variant::PutElement: __USING;
```

C++

```
__USING Variant::PutElement;
```

Description

This is Variant::PutElement, a member of class OleVariant.

38.1.10.4.64 OleVariant::Variant::SetError Data Member

Pascal

```
Variant::SetError: __USING;
```

C++

```
__USING Variant::SetError;
```

Description

This is Variant::SetError, a member of class OleVariant.

38.1.10.4.65 OleVariant::Variant::short Data Member

Pascal

```
Variant::short: __USING;
```

C++

```
__USING Variant::short;
```

Description

This is Variant::short, a member of class OleVariant.

38.1.10.4.66 OleVariant::Variant::short* Data Member

Pascal

```
Variant::short*: __USING;
```

C++

```
__USING Variant::short*;
```

Description

This is Variant::short*, a member of class OleVariant.

38.1.10.4.67 OleVariant::Variant::signed char Data Member

Pascal

```
Variant::signed char: __USING;
```

C++

```
__USING Variant::signed char;
```

Description

This is Variant::signed char, a member of class OleVariant.

38.1.10.4.68 OleVariant::Variant::signed char* Data Member

Pascal

```
Variant::signed char*: __USING;
```

C++

```
__USING Variant::signed char*;
```

Description

This is Variant::signed char*, a member of class OleVariant.

38.1.10.4.69 OleVariant::Variant::TDateTime Data Member

Pascal

```
Variant::TDateTime: __USING;
```

C++

```
__USING Variant::TDateTime;
```

Description

This is Variant::TDateTime, a member of class OleVariant.

38.1.10.4.70 OleVariant::Variant::TDateTime* Data Member

Pascal

```
Variant::TDateTime*: __USING;
```

C++

```
__USING Variant::TDateTime*;
```

Description

This is Variant::TDateTime*, a member of class OleVariant.

38.1.10.4.71 OleVariant::Variant::Type Data Member

Pascal

```
Variant::Type: __USING;
```

C++

```
__USING Variant::Type;
```

Description

This is Variant::Type, a member of class OleVariant.

38.1.10.4.72 OleVariant::Variant::UnicodeString Data Member

Pascal

```
Variant::UnicodeString: __USING;
```

C++

```
__USING Variant::UnicodeString;
```

Description

This is Variant::UnicodeString, a member of class OleVariant.

38.1.10.4.73 OleVariant::Variant::unsigned char Data Member

Pascal

```
Variant::unsigned char: __USING;
```

C++

```
__USING Variant::unsigned char;
```

Description

This is Variant::unsigned char, a member of class OleVariant.

38.1.10.4.74 OleVariant::Variant::unsigned int Data Member

Pascal

```
Variant::unsigned int: __USING;
```

C++

```
__USING Variant::unsigned int;
```

Description

This is Variant::unsigned int, a member of class OleVariant.

38.1.10.4.75 OleVariant::Variant::unsigned int* Data Member

Pascal

```
Variant::unsigned int*: __USING;
```

C++

```
__USING Variant::unsigned int*;
```

Description

This is Variant::unsigned int*, a member of class OleVariant.

38.1.10.4.76 OleVariant::Variant::unsigned long Data Member

Pascal

```
Variant::unsigned long: __USING;
```

C++

```
__USING Variant::unsigned long;
```

Description

This is Variant::unsigned long, a member of class OleVariant.

38.1.10.4.77 OleVariant::Variant::unsigned short Data Member

Pascal

```
Variant::unsigned short: __USING;
```

C++

```
__USING Variant::unsigned short;
```

Description

This is Variant::unsigned short, a member of class OleVariant.

38.1.10.4.78 OleVariant::Variant::unsigned short* Data Member

Pascal

```
Variant::unsigned short*: __USING;
```

C++

```
__USING Variant::unsigned short*;
```

Description

This is Variant::unsigned short*, a member of class OleVariant.

38.1.10.4.79 OleVariant::Variant::WideString Data Member

Pascal

```
Variant::WideString: __USING;
```

C++

```
__USING Variant::WideString;
```

Description

This is Variant::WideString, a member of class OleVariant.

38.1.10.5 OleVariant Operators

The operators of the OleVariant class are listed here.

Public Operators

	Name	Description
	!= (see page 3315)	This is the overview for the != operator overload.
	< (see page 3316)	This is the overview for the < operator overload.
	<= (see page 3317)	This is the overview for the <= operator overload.
	= (see page 3317)	This is the overview for the = operator overload.
	== (see page 3320)	This is the overview for the == operator overload.
	> (see page 3320)	This is the overview for the > operator overload.
	>= (see page 3321)	This is the overview for the >= operator overload.

38.1.10.5.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	OleVariant::!= (char*) (see page 3316)	This is !=, a member of class OleVariant.
	OleVariant::!= (OleVariant&) (see page 3316)	This is !=, a member of class OleVariant.

38.1.10.5.1.1 OleVariant::!= Operator (OleVariant&)

Pascal

```
operator !=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator !=(const OleVariant& rhs) const;
```

Description

This is !=, a member of class OleVariant.

38.1.10.5.1.2 OleVariant::!= Operator (char*)

Pascal

```
operator !=(const rhs: char*);
```

C++

```
__fastcall bool operator !=(const char* rhs) const;
```

Description

This is !=, a member of class OleVariant.

38.1.10.5.2 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	OleVariant::< (char*) (see page 3317)	This is <, a member of class OleVariant.
	OleVariant::< (OleVariant&) (see page 3316)	This is <, a member of class OleVariant.

38.1.10.5.2.1 OleVariant::< Operator (OleVariant&)

Pascal

```
operator <(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <(const OleVariant& rhs) const;
```

Description

This is <, a member of class OleVariant.

38.1.10.5.2.2 OleVariant::< Operator (char*)

Pascal

```
operator <(const rhs: char*);
```

C++

```
__fastcall bool operator <(const char* rhs) const;
```

Description

This is <, a member of class OleVariant.

38.1.10.5.3 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	OleVariant::<= (char*) (see page 3317)	This is <=, a member of class OleVariant.
	OleVariant::<= (OleVariant&) (see page 3317)	This is <=, a member of class OleVariant.

38.1.10.5.3.1 OleVariant::<= Operator (OleVariant&)

Pascal

```
operator <=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <=(const OleVariant& rhs) const;
```

Description

This is <=, a member of class OleVariant.

38.1.10.5.3.2 OleVariant::<= Operator (char*)

Pascal

```
operator <=(const rhs: char*);
```

C++

```
__fastcall bool operator <=(const char* rhs) const;
```

Description

This is <=, a member of class OleVariant.

38.1.10.5.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	OleVariant::= (__int64) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).

	OleVariant::= (AnsiString&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (bool) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (char) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (char*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (Currency&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (Currency*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (double) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (double*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (float) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (IDispatch*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (int) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (IUnknown*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (long) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (OleVariant&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (short) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (signed char) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (TDateTime&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (TDateTime*) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (UnicodeString&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (unsigned char) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (unsigned int) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (unsigned long) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (unsigned short) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).

	OleVariant::= (Variant&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).
	OleVariant::= (WideString&) (see page 3319)	Copies a value into the System::OleVariant::OleVariant (see page 3281).

38.1.10.5.4.1 OleVariant::= Operator (AnsiString&)

Copies a value into the System::OleVariant::OleVariant (see page 3281).

Pascal

```
operator =(const rhs: OleVariant&);
operator =(const rhs: Variant&);
operator =(const rhs: AnsiString&);
operator =(const rhs: char*);
operator =(const rhs: WideString&);
operator =(const rhs: UnicodeString&);
operator =(const rhs: bool);
operator =(const rhs: char);
operator =(const rhs: signed char);
operator =(const rhs: unsigned char);
operator =(const rhs: short);
operator =(const rhs: unsigned short);
operator =(const rhs: Integer);
operator =(const rhs: unsigned int);
operator =(const rhs: long);
operator =(const rhs: unsigned long);
operator =(const rhs: float);
operator =(const rhs: double);
operator =(const rhs: __int64);
operator =(const rhs: TDateTime&);
operator =(const rhs: Currency&);
operator =(var rhs: double);
operator =(var rhs: TDateTime);
operator =(var rhs: Currency);
operator =(var rhs: IDispatch);
operator =(var rhs: IUnknown);
```

C++

```
__fastcall OleVariant& operator =(const OleVariant& rhs);
__fastcall OleVariant& operator =(const Variant& rhs);
__fastcall OleVariant& operator =(const AnsiString& rhs);
__fastcall OleVariant& operator =(const char* rhs);
__fastcall OleVariant& operator =(const WideString& rhs);
__fastcall OleVariant& operator =(const UnicodeString& rhs);
__fastcall OleVariant& operator =(const bool rhs);
__fastcall OleVariant& operator =(const char rhs);
__fastcall OleVariant& operator =(const signed char rhs);
__fastcall OleVariant& operator =(const unsigned char rhs);
__fastcall OleVariant& operator =(const short rhs);
__fastcall OleVariant& operator =(const unsigned short rhs);
__fastcall OleVariant& operator =(const int rhs);
__fastcall OleVariant& operator =(const unsigned int rhs);
__fastcall OleVariant& operator =(const long rhs);
__fastcall OleVariant& operator =(const unsigned long rhs);
__fastcall OleVariant& operator =(const float rhs);
__fastcall OleVariant& operator =(const double rhs);
__fastcall OleVariant& operator =(const __int64 rhs);
__fastcall OleVariant& operator =(const TDateTime& rhs);
__fastcall OleVariant& operator =(const Currency& rhs);
__fastcall OleVariant& operator =(double* rhs);
__fastcall OleVariant& operator =(TDateTime* rhs);
__fastcall OleVariant& operator =(Currency* rhs);
```

```
__fastcall OleVariant& operator =(IDispatch* rhs);
__fastcall OleVariant& operator =(IUnknown* rhs);
```

Description

The assignment operator converts the value of rhs, as appropriate, and assigns it to the System::OleVariant::OleVariant (see page 3281). If it is not possible to perform the conversion, an EVariantError exception is thrown.

38.1.10.5.5 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	OleVariant::== (char*) (see page 3320)	This is ==, a member of class OleVariant.
	OleVariant::== (OleVariant&) (see page 3320)	This is ==, a member of class OleVariant.

38.1.10.5.5.1 OleVariant::== Operator (OleVariant&)

Pascal

```
operator ==(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator ==(const OleVariant& rhs) const;
```

Description

This is ==, a member of class OleVariant.

38.1.10.5.5.2 OleVariant::== Operator (char*)

Pascal

```
operator ==(const rhs: char*);
```

C++

```
__fastcall bool operator ==(const char* rhs) const;
```

Description

This is ==, a member of class OleVariant.

38.1.10.5.6 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	OleVariant::> (char*) (see page 3321)	This is >, a member of class OleVariant.
	OleVariant::> (OleVariant&) (see page 3321)	This is >, a member of class OleVariant.

38.1.10.5.6.1 OleVariant::> Operator (OleVariant&)

Pascal

```
operator >(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >(const OleVariant& rhs) const;
```

Description

This is >, a member of class OleVariant.

38.1.10.5.6.2 OleVariant::> Operator (char*)

Pascal

```
operator >(const rhs: char*);
```

C++

```
__fastcall bool operator >(const char* rhs) const;
```

Description

This is >, a member of class OleVariant.

38.1.10.5.7 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	OleVariant::>= (char*) (see page 3321)	This is >=, a member of class OleVariant.
	OleVariant::>= (OleVariant&) (see page 3321)	This is >=, a member of class OleVariant.

38.1.10.5.7.1 OleVariant::>= Operator (OleVariant&)

Pascal

```
operator >=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >=(const OleVariant& rhs) const;
```

Description

This is >=, a member of class OleVariant.

38.1.10.5.7.2 OleVariant::>= Operator (char*)

Pascal

```
operator >=(const rhs: char*);
```

C++

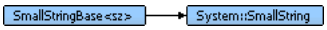
```
__fastcall bool operator >=(const char* rhs) const;
```

Description

This is >=, a member of class OleVariant.

38.1.11 SmallString Template

Class Hierarchy



Pascal

```
SmallString = class(SmallStringBase<sz>);
```

C++

```
template <unsigned char sz>
class SmallString : private SmallStringBase<sz>;
```

File

dstring.h

Description

This is class System::SmallString.

38.1.11.1 SmallString Members

The following tables list the members exposed by SmallString.

Public Methods

	Name	Description
	SmallString (see page 3322)	This is the overview for the SmallString constructor overload.

Public Operators

	Name	Description
	[] (see page 3324)	This is [], a member of class SmallString.
	= (see page 3324)	This is =, a member of class SmallString.
	AnsiString (see page 3324)	This is AnsiString, a member of class SmallString.

38.1.11.2 SmallString Constructor

This is the overview for the SmallString constructor overload.

Overload List

	Name	Description
	SmallString::SmallString () (see page 3323)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (AnsiString&) (see page 3323)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (char*) (see page 3323)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (SmallString&) (see page 3323)	This is SmallString, a member of class SmallString.

38.1.11.2.1 SmallString::SmallString Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall SmallString();
```

Description

This is SmallString, a member of class SmallString.

38.1.11.2.2 SmallString::SmallString Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall SmallString(const AnsiString& src);
```

Description

This is SmallString, a member of class SmallString.

38.1.11.2.3 SmallString::SmallString Constructor (SmallString&)

Pascal

```
constructor Create(const src: SmallString&);
```

C++

```
__fastcall SmallString(const SmallString& src);
```

Description

This is SmallString, a member of class SmallString.

38.1.11.2.4 SmallString::SmallString Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall SmallString(const char* src);
```

Description

This is SmallString, a member of class SmallString.

38.1.11.3 SmallString Operators

The operators of the SmallString class are listed here.

Public Operators

	Name	Description
	[] (see page 3324)	This is [], a member of class SmallString.
	= (see page 3324)	This is =, a member of class SmallString.
	AnsiString (see page 3324)	This is AnsiString, a member of class SmallString.

38.1.11.3.1 SmallString::[] Operator

Pascal

```
operator [](const idx: unsigned char);
```

C++

```
__fastcall char& operator [](const unsigned char idx);
```

Description

This is [], a member of class SmallString.

38.1.11.3.2 SmallString::= Operator

Pascal

```
operator =(const rhs: SmallString&);
```

C++

```
__fastcall SmallString& operator =(const SmallString& rhs);
```

Description

This is =, a member of class SmallString.

38.1.11.3.3 SmallString::AnsiString Operator

Pascal

```
operator AnsiString;
```

C++

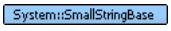
```
__fastcall operator AnsiString() const;
```

Description

This is AnsiString, a member of class SmallString.

38.1.12 SmallStringBase Template

Class Hierarchy



Pascal

```
SmallStringBase = class;
```

C++

```
template <unsigned char sz>
```

```
class SmallStringBase;
```

File

dstring.h

Description

This is class System::SmallStringBase.

38.1.12.1 SmallStringBase Members

The following tables list the members exposed by SmallStringBase.

Protected Data Members

	Name	Description
	Data (see page 3325)	This is Data, a member of class SmallStringBase.
	Len (see page 3325)	This is Len, a member of class SmallStringBase.

38.1.12.2 SmallStringBase Data Members

The data members of the SmallStringBase class are listed here.

Protected Data Members

	Name	Description
	Data (see page 3325)	This is Data, a member of class SmallStringBase.
	Len (see page 3325)	This is Len, a member of class SmallStringBase.

38.1.12.2.1 SmallStringBase::Data Data Member

Pascal

```
[sz] Data: char;
```

C++

```
char Data[sz];
```

Description

This is Data, a member of class SmallStringBase.

38.1.12.2.2 SmallStringBase::Len Data Member

Pascal

```
Len: unsigned char;
```

C++

```
unsigned char Len;
```

Description

This is Len, a member of class SmallStringBase.

38.1.13 sPointer Template

Class Hierarchy

Pascal

```
sPointer = class;
```

C++

```
template <typename T>
class sPointer;
```

File

sysdyn.h

Description

This is class System::sPointer.

38.1.13.1 sPointer Members

The following tables list the members exposed by sPointer.

Public Methods

	Name	Description
	~sPointer (see page 3326)	This is ~sPointer, a member of class sPointer.
	sPointer (see page 3326)	This is the overview for the sPointer constructor overload.

Public Operators

	Name	Description
	& (see page 3327)	This is &, a member of class sPointer.
	-> (see page 3328)	This is ->, a member of class sPointer.

38.1.13.2 sPointer::~sPointer Destructor

Pascal

```
destructor Destroy;
```

C++

```
~sPointer();
```

Description

This is ~sPointer, a member of class sPointer.

38.1.13.3 sPointer Constructor

This is the overview for the sPointer constructor overload.

Overload List

	Name	Description
	sPointer::sPointer () (see page 3327)	This is sPointer, a member of class sPointer.
	sPointer::sPointer (T*) (see page 3327)	This is sPointer, a member of class sPointer.

38.1.13.3.1 sPointer::sPointer Constructor ()

Pascal

```
constructor Create;
```

C++

```
sPointer();
```

Description

This is sPointer, a member of class sPointer.

38.1.13.3.2 sPointer::sPointer Constructor (T*)

Pascal

```
constructor Create(var p: T);
```

C++

```
sPointer(T* p);
```

Description

This is sPointer, a member of class sPointer.

38.1.13.4 sPointer Operators

The operators of the sPointer class are listed here.

Public Operators

	Name	Description
	& (see page 3327)	This is &, a member of class sPointer.
	-> (see page 3328)	This is ->, a member of class sPointer.

38.1.13.4.1 sPointer::& Operator

Pascal

```
operator &;
```

C++

```
T* operator &();
```

Description

This is &, a member of class sPointer.

38.1.13.4.2 sPointer::-> Operator

Pascal

```
operator ->;
```

C++

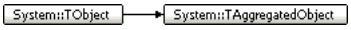
```
T* operator ->();
```

Description

This is ->, a member of class sPointer.

38.1.14 TAggregatedObject Class

Class Hierarchy



Pascal

```
TAggregatedObject = class(TObject);
```

C++

```
class TAggregatedObject : public TObject;
```

File

System.pas

Description

This is class System::TAggregatedObject.

38.1.14.1 TAggregatedObject Members

The following tables list the members exposed by TAggregatedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3330)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 3330)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 3330)	This is QueryInterface, a member of class TAggregatedObject.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 3329)	Instantiates an aggregated object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Controller (see page 3331)	Provides access to the controlling interface of the aggregate.

38.1.14.2 TAggregatedObject::Create Constructor

Instantiates an aggregated object.

Pascal

```
constructor Create(const Controller: IInterface);
```

C++

```
Create(const IInterface Controller);
```

Description

Call System::TAggregatedObject::Create to instantiate an aggregated object at runtime.

System::TAggregatedObject::Create allocates memory for an aggregated object and sets the System::TAggregatedObject::Controller (see page 3331) property to the System::TAggregatedObject::Controller (see page 3331) parameter passed to it. The System::TAggregatedObject::Controller (see page 3331) is the controlling interface of the aggregate. All other interfaces implemented in the aggregate delegate IInterface (see page 3503) method calls to this System::TAggregatedObject::Controller (see page 3331).

See Also

System::TAggregatedObject::Controller (see page 3331)

38.1.14.3 TAggregatedObject Methods

The methods of the TAggregatedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 3330)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 3330)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 3330)	This is QueryInterface, a member of class TAggregatedObject.

38.1.14.3.1 TAggregatedObject::_AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

This is _AddRef, a member of class TAggregatedObject.

38.1.14.3.2 TAggregatedObject::_Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

This is _Release, a member of class TAggregatedObject.

38.1.14.3.3 TAggregatedObject::QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

This is QueryInterface, a member of class TAggregatedObject.

38.1.14.4 TAggregatedObject Properties

The properties of the TAggregatedObject class are listed here.

Public Properties

	Name	Description
R	Controller (see page 3331)	Provides access to the controlling interface of the aggregate.

38.1.14.4.1 TAggregatedObject::Controller Property

Provides access to the controlling interface of the aggregate.

Pascal

```
property Controller: IInterface;
```

C++

```
__property IInterface Controller;
```

Description

System::TAggregatedObject::Controller is the interface reference of the aggregate. System::TAggregatedObject::TAggregatedObject (see page 3328) is used as part of an aggregate that has a single controlling IInterface (see page 3503). The IInterface (see page 3503) methods that are implemented in aggregated objects are delegated to the System::TAggregatedObject::Controller.

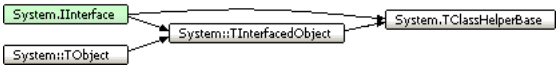
When the aggregated object is created, its constructor is passed a controlling IInterface (see page 3503). System::TAggregatedObject::Controller is initialized to this IInterface (see page 3503) in the constructor.

See Also

System::TAggregatedObject::Create (see page 3329)

38.1.15 TClassHelperBase Class

Class Hierarchy



Pascal

```
TClassHelperBase = class(TInterfacedObject, IInterface);
```

C++

```
class TClassHelperBase : public TInterfacedObject, public IInterface;
```

File

System

Description

This is class System.TClassHelperBase.

38.1.15.1 TClassHelperBase Members

The following tables list the members exposed by TClassHelperBase.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FInstance (see page 3333)	This is FInstance, a member of class TClassHelperBase.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Create (see page 3333)	This is _Create, a member of class TClassHelperBase.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.

	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

38.1.15.2 TClassHelperBase._Create Constructor

Pascal

```
constructor _Create(Instance: TObject);
```

C++

```
__fastcall TClassHelperBase(TObject * Instance);
```

Description

This is _Create, a member of class TClassHelperBase.

38.1.15.3 TClassHelperBase Fields

The fields of the TClassHelperBase class are listed here.

Protected Fields

	Name	Description
	FInstance (see page 3333)	This is FInstance, a member of class TClassHelperBase.

38.1.15.3.1 TClassHelperBase.FInstance Field

Pascal

```
FInstance: TObject;
```

C++

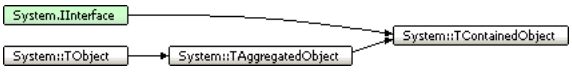
```
TObject * FInstance;
```

Description

This is FInstance, a member of class TClassHelperBase.

38.1.16 TContainedObject Class

Class Hierarchy



Pascal

```
TContainedObject = class(TAggregatedObject, IInterface);
```

C++

```
class TContainedObject : public TAggregatedObject, public IInterface;
```

File

System.pas

Description

This is class System::TContainedObject.

38.1.16.1 TContainedObject Members

The following tables list the members exposed by TContainedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3330)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 3330)	This is _Release, a member of class TAggregatedObject.
V	QueryInterface (see page 3335)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3329)	Instantiates an aggregated object.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Controller (see page 3331)	Provides access to the controlling interface of the aggregate.

38.1.16.2 TContainedObject Methods

The methods of the TContainedObject class are listed here.

Protected Methods

	Name	Description
	QueryInterface (see page 3335)	Returns a reference to a specified interface if the object supports that interface.

38.1.16.2.1 TContainedObject::QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

QueryInterface checks whether the object that implements this interface supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 3504)

38.1.17 TDateTime Class

System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.

Class Hierarchy

Pascal

```
TDateTime = class(TDateTimeBase);
```

C++

```
class TDateTime : public TDateTimeBase;
```

File

systdate.h

Description

System::TDateTime implements the Delphi System::TDateTime data type and the date/time runtime library routines that use the System::TDateTime data type.

The System::TDateTime class inherits a val data member declared as a double (see page 3353) that holds the date-time value. The integral part of a System::TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a System::TDateTime value is the time of day.

Following are some examples of System::TDateTime values and their corresponding dates and times:

0 12/30/1899 12:00 am

2.75 1/1/1900 6:00 pm

-1.25 12/29/1899 6:00 am

35065 1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the System::TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the System::TDateTime value is positive.

When working with negative System::TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the System::TDateTime value. For example, 6:00 am on 12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no System::TDateTime values between –1 and 0.

Note: Use only the operators declared by System::TDateTime. The compiler ignores any operators you overload yourself.

Warning: It is not possible to use System::TDateTime as a base class for inheriting. To add functionality to System::TDateTime, you must write a wrapper class.

Note: To utilize the C++ streaming operators (<< and >>) with Currency (see page 3242), you must use #include <iostream> before #include <systdate.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

ostream& operator << (ostream& os, const System::TDateTime& arg);

istream& operator >> (istream& is, System::TDateTime& arg);

38.1.17.1 TDateTime Members

The following tables list the members exposed by TDateTime.

Enumerations

	Name	Description
	TDateTimeFlag (see page 3340)	This is record System::TDateTime::TDateTimeFlag.

Public Data Members

	Name	Description
	Val (see page 3354)	This is Val, a member of class TDateTimeBase.

Public Methods

	Name	Description
	CurrentDate (see page 3340)	Returns the current date as a System::TDateTime::TDateTime (see page 3335) value.
	CurrentDateTime (see page 3341)	Returns the current date and time as a System::TDateTime::TDateTime (see page 3335) value.
	CurrentTime (see page 3341)	Returns the current time as a System::TDateTime::TDateTime (see page 3335) value.
	DateString (see page 3342)	Converts the date of the System::TDateTime::TDateTime (see page 3335) value to a string.
	DateTimeString (see page 3342)	Converts the System::TDateTime::TDateTime (see page 3335) value to a string.
	DayOfWeek (see page 3342)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3335)
	DecodeDate (see page 3343)	Separates the System::TDateTime::TDateTime (see page 3335) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 3343)	Separates the System::TDateTime::TDateTime (see page 3335) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 3343)	Converts the date/time value to an OS date/time stamp.
	FileDateToDateTime (see page 3344)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3335) value.
	FormatString (see page 3344)	Formats the System::TDateTime::TDateTime (see page 3335) object using the format given by format.
	TDateTime (see page 3338)	This is the overview for the TDateTime constructor overload.
	TimeString (see page 3345)	Converts the System::TDateTime::TDateTime (see page 3335) object to an AnsiString.

Public Operators

	Name	Description
	- (see page 3346)	This is the overview for the - operator overload.
	-- (see page 3347)	This is the overview for the -- operator overload.
	!= (see page 3348)	Compares two System::TDateTime::TDateTime (see page 3335) objects for inequality.
	+ (see page 3348)	This is the overview for the + operator overload.
	++ (see page 3349)	This is the overview for the ++ operator overload.
	+= (see page 3349)	This is the overview for the += operator overload.
	< (see page 3350)	Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than another specified System::TDateTime::TDateTime (see page 3335) value.
	<= (see page 3350)	Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3335) value.
	= (see page 3350)	This is the overview for the = operator overload.
	-= (see page 3351)	This is the overview for the -= operator overload.
	== (see page 3352)	Compares two System::TDateTime::TDateTime (see page 3335) objects for equality.
	> (see page 3352)	This is >, a member of class TDateTime.
	>= (see page 3353)	This is >=, a member of class TDateTime.
	double (see page 3353)	Converts the System::TDateTime::TDateTime (see page 3335) object to a double value.
	int (see page 3353)	Converts the System::TDateTime::TDateTime (see page 3335) object to an integer value.
	System::String (see page 3353)	This is System::String, a member of class TDateTime.

38.1.17.2 TDateTime Constructor

This is the overview for the TDateTime constructor overload.

Overload List

	Name	Description
	TDateTime::TDateTime () (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (double) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (int) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (System::String&, TDateTimeFlag) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (TDateTime&) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (TDateTimeBase&) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short) (see page 3338)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).

38.1.17.2.1 TDateTime::TDateTime Constructor ()

Creates and initializes an instance of System::TDateTime::TDateTime (see page 3335).

Pascal

```
constructor Create;
constructor Create(const src: TDateTimeBase&);
constructor Create(const src: TDateTime&);
constructor Create(const src: double);
constructor Create(const src: Integer);
constructor Create(const src: System::String&; flag: TDateTimeFlag = DateTime);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short);
constructor Create(hour: unsigned short; min: unsigned short; sec: unsigned short; msec:
unsigned short);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short; hour:
unsigned short; min: unsigned short; sec: unsigned short; msec: unsigned short);
```

C++

```
__fastcall TDateTime();
__fastcall TDateTime(const TDateTimeBase& src);
__fastcall TDateTime(const TDateTime& src);
__fastcall TDateTime(const double src);
__fastcall TDateTime(const int src);
```

```
__fastcall TDateTime(const System::String& src, TDateTimeFlag flag = DateTime);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day);
__fastcall TDateTime(unsigned short hour, unsigned short min, unsigned short sec, unsigned
short msec);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day, unsigned
short hour, unsigned short min, unsigned short sec, unsigned short msec);
```

Description

Use System::TDateTime::TDateTime (see page 3335) to instantiate a System::TDateTime::TDateTime (see page 3335) object.

When called with no arguments, the resulting System::TDateTime::TDateTime (see page 3335) object has a Val (see page 3785) data member of zero. This date and time corresponds to 12/30/1899 12:00 am.

When called with a System::TDateTime::TDateTime (see page 3335) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::TDateTime::TDateTimeBase (see page 3354) object as the src parameter, the resulting object has the same value as the src parameter. System::TDateTime::TDateTimeBase (see page 3354) is the base class for the System::TDateTime::TDateTime (see page 3335) type. It has a single member, a public Val (see page 3785) field that represents the value of the object.

When called with a double (see page 3353) or integer as the src parameter, the resulting object has the same value as the src parameter. The integral part of src is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

When called with an AnsiString as the src parameter and a System::TDateTime::TDateTimeFlag (see page 3340) as the flag parameter, the resulting object's value is converted from the AnsiString. The flag parameter indicates whether the string contains a date, a time, or both. The default for the flag is DateTime.

When called with three unsigned shorts (as the year, month, and day parameters), the resulting object's value is assembled from the specified year, month, and day, with a time portion of 0 (midnight). The year must be between 1 and 9999. Valid month values are 1 through 12. Valid day values are 1 through 28, 29, 30, or 31, depending on the month value. For example, the possible day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the year value specifies a leap year.

When called with four unsigned shorts (as the hour, min, sec, and msec parameters), the resulting object's value represents a time based on the specified number of hours, minutes, seconds, and milliseconds past midnight. The date portion of the resulting value is 0 (12/30/1899). The hour must be between 0 and 23, the minute must be between 0 and 59, the second must be between 0 and 59, and the millisecond must be between 0 and 999.

Note: When the constructor calls for a conversion that can't be performed (an AnsiString that does not represent a date or time, or invalid year, month, day, hour, min, sec, or msec value), an EConvertError exception is thrown.

38.1.17.3 TDateTime Enumerations

The enumerations of the TDateTime class are listed here.

Enumerations

	Name	Description
	TDateTimeFlag (see page 3340)	This is record System::TDateTime::TDateTimeFlag.

38.1.17.3.1 System::TDateTime::TDateTimeFlag Enumeration

Pascal

```
TDateTimeFlag = (
  Date,
  Time,
  DateTime
);
```

C++

```
enum TDateTimeFlag {
  Date,
  Time,
  DateTime
};
```

File

systdate.h

Description

This is record System::TDateTime::TDateTimeFlag.

38.1.17.4 TDateTime Methods

The methods of the TDateTime class are listed here.

Public Methods

	Name	Description
	CurrentDate (see page 3340)	Returns the current date as a System::TDateTime::TDateTime (see page 3335) value.
	CurrentDateTime (see page 3341)	Returns the current date and time as a System::TDateTime::TDateTime (see page 3335) value.
	CurrentTime (see page 3341)	Returns the current time as a System::TDateTime::TDateTime (see page 3335) value.
	DateString (see page 3342)	Converts the date of the System::TDateTime::TDateTime (see page 3335) value to a string.
	DateTimeString (see page 3342)	Converts the System::TDateTime::TDateTime (see page 3335) value to a string.
	DayOfWeek (see page 3342)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3335)
	DecodeDate (see page 3343)	Separates the System::TDateTime::TDateTime (see page 3335) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 3343)	Separates the System::TDateTime::TDateTime (see page 3335) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 3343)	Converts the date/time value to an OS date/time stamp.
	FileDateToDateTime (see page 3344)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3335) value.
	FormatString (see page 3344)	Formats the System::TDateTime::TDateTime (see page 3335) object using the format given by format.
	TimeString (see page 3345)	Converts the System::TDateTime::TDateTime (see page 3335) object to an AnsiString.

38.1.17.4.1 TDateTime::CurrentDate Method

Returns the current date as a System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
function CurrentDate: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDate();
```

Description

System::TDateTime::CurrentDate retrieves the current date and returns it as a new System::TDateTime::TDateTime (see page 3335) object with a 0 time value.

See Also

System::TDateTime::CurrentDateTime (see page 3341)

System::TDateTime::CurrentTime (see page 3341)

38.1.17.4.2 TDateTime::CurrentDateTime Method

Returns the current date and time as a System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
function CurrentDateTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDateTime();
```

Description

System::TDateTime::CurrentDateTime retrieves the current date and time, and returns it as a new System::TDateTime::TDateTime (see page 3335) object.

See Also

System::TDateTime::CurrentTime (see page 3341)

System::TDateTime::CurrentDate (see page 3340)

38.1.17.4.3 TDateTime::CurrentTime Method

Returns the current time as a System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
function CurrentTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentTime();
```

Description

System::TDateTime::CurrentTime retrieves the current time and returns it as a new System::TDateTime::TDateTime (see page 3335) object with a 0 date value.

See Also

System::TDateTime::CurrentDateTime (see page 3341)

System::TDateTime::CurrentDate (see page 3340)

38.1.17.4.4 TDateTime::DateString Method

Converts the date of the System::TDateTime::TDateTime (see page 3335) value to a string.

Pascal

```
function DateString: System::String;
```

C++

```
__fastcall System::String DateString() const;
```

Description

System::TDateTime::DateString returns the value of the System::TDateTime::TDateTime (see page 3335) object as an AnsiString. The conversion uses the format specified by the ShortDateFormat global variable.

See Also

System::TDateTime::DateTimeString (see page 3342)

System::TDateTime::TimeString (see page 3345)

variables

38.1.17.4.5 TDateTime::DateTimeString Method

Converts the System::TDateTime::TDateTime (see page 3335) value to a string.

Pascal

```
function DateTimeString: System::String;
```

C++

```
__fastcall System::String DateTimeString() const;
```

Description

System::TDateTime::DateTimeString returns an AnsiString object containing the date and time value (the Val (see page 3785) data member) of the System::TDateTime::TDateTime (see page 3335) object to a string. System::TDateTime::DateTimeString converts the System::TDateTime::TDateTime (see page 3335) value using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.

See Also

System::TDateTime::DateString (see page 3342)

System::TDateTime::TimeString (see page 3345)

variables

38.1.17.4.6 TDateTime::DayOfWeek Method

Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3335)

Pascal

```
function DayOfWeek: Integer;
```

C++

```
__fastcall int DayOfWeek() const;
```

Description

System::TDateTime::DayOfWeek returns an integer between 1 and 7. Sunday is treated as the first day of the week and Saturday as the seventh.

38.1.17.4.7 TDateTime::DecodeDate Method

Separates the System::TDateTime::TDateTime (see page 3335) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.

Pascal

```
procedure DecodeDate(var year: unsigned short; var month: unsigned short; var day: unsigned short);
```

C++

```
__fastcall void DecodeDate(unsigned short* year, unsigned short* month, unsigned short* day) const;
```

Description

Use System::TDateTime::DecodeDate() when you need to access the year, month, or day of a System::TDateTime::TDateTime (see page 3335) object's date portion.

38.1.17.4.8 TDateTime::DecodeTime Method

Separates the System::TDateTime::TDateTime (see page 3335) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.

Pascal

```
procedure DecodeTime(var hour: unsigned short; var min: unsigned short; var sec: unsigned short; var msec: unsigned short);
```

C++

```
__fastcall void DecodeTime(unsigned short* hour, unsigned short* min, unsigned short* sec, unsigned short* msec) const;
```

Description

Use System::TDateTime::DecodeTime() to access the hour, minute, second, or millisecond values of the System::TDateTime::TDateTime (see page 3335) object's time portion.

38.1.17.4.9 TDateTime::FileDate Method

Converts the date/time value to an OS date/time stamp.

Pascal

```
function FileDate: Integer;
```

C++

```
__fastcall int FileDate() const;
```

Description

Use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3335) value to an OS date/time stamp. For example, use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3335) value to a format that can be used as the Age parameter of FileSetDate. You may also need to use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3335) so that it is compatible with the Time field of the TSearchRec

structure used by the FindFirst and FindNext functions.

See Also

System::TDateTime::FileDateToDateTime (see page 3344)

FileSetDate (see page 4212)

TSearchRec (see page 4371)

38.1.17.4.10 TDateTime::FileDateToDateTime Method

Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
function FileDateToDateTime(fileDate: Integer): TDateTime; static;
```

C++

```
static __fastcall TDateTime FileDateToDateTime(int fileDate);
```

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use System::TDateTime::FileDateToDateTime to convert a timestamp to a System::TDateTime::TDateTime (see page 3335) value.

For example, use System::TDateTime::FileDateToDateTime to convert the file timestamp returned by FileGetDate or FileAge. You may also need to use System::TDateTime::FileDateToDateTime to convert the Time field of the TSearchRec struct used by the FindFirst and FindNext functions.

See Also

FileGetDate (see page 4199)

FileAge (see page 4183)

TSearchRec (see page 4371)

38.1.17.4.11 TDateTime::FormatString Method

Formats the System::TDateTime::TDateTime (see page 3335) object using the format given by format.

Pascal

```
function FormatString(const format: System::String&): System::String;
```

C++

```
__fastcall System::String FormatString(const System::String& format) const;
```

Description

The following format specifiers are supported:

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).

ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays the date using the format given by the ShortDateFormat global variable.
dddddd	Displays the date using the format given by the LongDateFormat global variable.
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3652)) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time using the format given by the ShortTimeFormat global variable.
tt	Displays the time using the format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator global variable.
:	Displays the time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double (see page 3353) quotes are displayed as is, and do not affect formatting.

If the string given by the format parameter is empty, the System::TDateTime::TDateTime (see page 3335) value is formatted as if a c format specifier had been given.

38.1.17.4.12 TDateTime::TimeString Method

Converts the System::TDateTime::TDateTime (see page 3335) object to an AnsiString.

Pascal

```
function TimeString: System::String;
```

C++

```
__fastcall System::String TimeString() const;
```

Description

System::TDateTime::TimeString returns an AnsiString object containing the time value (the fractional portion of the Val (see page 3785) data member) of the System::TDateTime::TDateTime (see page 3335) object to a string. The conversion uses the format specified by the LongTimeFormat variable. Change the format of how the string is displayed by changing the value of one or more of the date/time formatting variables.

See Also

System::TDateTime::DateString (see page 3342)

System::TDateTime::DateTimeString (see page 3342)

variables

38.1.17.5 TDateTime Operators

The operators of the TDateTime class are listed here.

Public Operators

	Name	Description
	- (see page 3346)	This is the overview for the - operator overload.
	-- (see page 3347)	This is the overview for the -- operator overload.
	!= (see page 3348)	Compares two System::TDateTime::TDateTime (see page 3335) objects for inequality.
	+ (see page 3348)	This is the overview for the + operator overload.
	++ (see page 3349)	This is the overview for the ++ operator overload.
	+= (see page 3349)	This is the overview for the += operator overload.
	< (see page 3350)	Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than another specified System::TDateTime::TDateTime (see page 3335) value.
	<= (see page 3350)	Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3335) value.
	= (see page 3350)	This is the overview for the = operator overload.
	-= (see page 3351)	This is the overview for the -= operator overload.
	== (see page 3352)	Compares two System::TDateTime::TDateTime (see page 3335) objects for equality.
	> (see page 3352)	This is >, a member of class TDateTime.
	>= (see page 3353)	This is >=, a member of class TDateTime.
	double (see page 3353)	Converts the System::TDateTime::TDateTime (see page 3335) object to a double value.
	int (see page 3353)	Converts the System::TDateTime::TDateTime (see page 3335) object to an integer value.
	System::String (see page 3353)	This is System::String, a member of class TDateTime.

38.1.17.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	TDateTime::- (double) (see page 3347)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::- (int) (see page 3347)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3335) object.

	TDateTime::- (TDateTime&) (see page 3347)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::- (TDateTimeBase&) (see page 3347)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3335) object.

38.1.17.5.1.1 TDateTime::- Operator (TDateTime&)

Subtracts a specified value from the System::TDateTime::TDateTime (see page 3335) object.

Pascal

```
operator -(const rhs: TDateTimeBase&);
operator -(const rhs: TDateTime&);
operator -(const rhs: double);
operator -(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator -(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator -(const TDateTime& rhs) const;
__fastcall TDateTime operator -(const double rhs) const;
__fastcall TDateTime operator -(const int rhs) const;
```

Description

Use the System::TDateTime (see page 3335)::- to subtract the value specified by the rhs operand from the System::TDateTime::TDateTime (see page 3335) object. The rhs operand represents the number of days (including fractional days) to subtract.

38.1.17.5.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	TDateTime::-- () (see page 3347)	Decrements the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.
	TDateTime::-- (int) (see page 3347)	Decrements the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.

38.1.17.5.2.1 TDateTime::-- Operator ()

Decrements the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
TDateTime& operator --();
TDateTime operator --(int);
```

Description

Use the unary System::TDateTime::operator_MinusMinus to decrement the value of the System::TDateTime::TDateTime (see page 3335) object by 1 day.

Use the binary System::TDateTime::operator_MinusMinus to decrement the value by the number of days specified by the integer operand.

38.1.17.5.3 TDateTime::!= Operator

Compares two System::TDateTime::TDateTime (see page 3335) objects for inequality.

Pascal

```
operator !=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator !=(const TDateTime& rhs) const;
```

Description

The != System::TDateTime (see page 3335)::!= returns true if the two objects do not have the same value, false otherwise.

38.1.17.5.4 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	TDateTime::+ (double) (see page 3348)	Adds a specified value to the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::+ (int) (see page 3348)	Adds a specified value to the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::+ (TDateTime&) (see page 3348)	Adds a specified value to the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::+ (TDateTimeBase&) (see page 3348)	Adds a specified value to the System::TDateTime::TDateTime (see page 3335) object.

38.1.17.5.4.1 TDateTime::+ Operator (TDateTime&)

Adds a specified value to the System::TDateTime::TDateTime (see page 3335) object.

Pascal

```
operator +(const rhs: TDateTimeBase&);
operator +(const rhs: TDateTime&);
operator +(const rhs: double);
operator +(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator +(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator +(const TDateTime& rhs) const;
__fastcall TDateTime operator +(const double rhs) const;
__fastcall TDateTime operator +(const int rhs) const;
```

Description

Use the + System::TDateTime (see page 3335)::+ to add the value specified by the rhs operand to the System::TDateTime::TDateTime (see page 3335) object. The rhs operand represents the number of days (including fractional days) to add.

38.1.17.5.5 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	TDateTime::++ () (see page 3349)	Increments the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.
	TDateTime::++ (int) (see page 3349)	Increments the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.

38.1.17.5.5.1 TDateTime::++ Operator ()

Increments the value of the System::TDateTime::TDateTime (see page 3335) object by a specified number of days.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
TDateTime& operator ++();
TDateTime operator ++(int);
```

Description

Use the unary ++ System::TDateTime (see page 3335)::+Plus to increment the value of the System::TDateTime::TDateTime (see page 3335) object by 1 day.

Use the binary ++ System::TDateTime (see page 3335)::+Plus to increment the value by the number of days specified by the integer operand.

38.1.17.5.6 += Operator

This is the overview for the += operator overload.

Overload List

	Name	Description
	TDateTime::+= (double) (see page 3349)	Increments the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.
	TDateTime::+= (int) (see page 3349)	Increments the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.
	TDateTime::+= (TDateTime&) (see page 3349)	Increments the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.
	TDateTime::+= (TDateTimeBase&) (see page 3349)	Increments the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.

38.1.17.5.6.1 TDateTime::+= Operator (TDateTime&)

Increments the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.

Pascal

```
operator +=(const rhs: TDateTimeBase&);
operator +=(const rhs: TDateTime&);
operator +=(const rhs: double);
operator +=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator +=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator +=(const TDateTime& rhs);
__fastcall TDateTime& operator +=(const double rhs);
__fastcall TDateTime& operator +=(const int rhs);
```

Description

Use the += System::TDateTime (see page 3335)::+= to add to the value specified by rhs to this System::TDateTime::TDateTime (see page 3335) value. The rhs operand represents the number of days (including fractional days) to add.

38.1.17.5.7 TDateTime::< Operator

Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than another specified System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
operator <(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <(const TDateTime& rhs) const;
```

Description

The < System::TDateTime (see page 3335)::< returns true if this System::TDateTime::TDateTime (see page 3335) object has a value (Val (see page 3785) data member) less than that of the object specified by rhs, and false otherwise.

38.1.17.5.8 TDateTime::<= Operator

Tests whether this System::TDateTime::TDateTime (see page 3335) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3335) value.

Pascal

```
operator <=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <=(const TDateTime& rhs) const;
```

Description

The <= System::TDateTime (see page 3335)::<= returns true if this System::TDateTime::TDateTime (see page 3335) object has a value (Val (see page 3785) data member) less than or equal to that of the object specified by rhs, and false otherwise.

38.1.17.5.9 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	TDateTime::= (double) (see page 3351)	Assigns the value of the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::= (int) (see page 3351)	Assigns the value of the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::= (TDateTime&) (see page 3351)	Assigns the value of the System::TDateTime::TDateTime (see page 3335) object.
	TDateTime::= (TDateTimeBase&) (see page 3351)	Assigns the value of the System::TDateTime::TDateTime (see page 3335) object.

38.1.17.5.9.1 TDateTime::= Operator (TDateTime&)

Assigns the value of the System::TDateTime::TDateTime (see page 3335) object.

Pascal

```
operator =(const rhs: TDateTimeBase&);
operator =(const rhs: TDateTime&);
operator =(const rhs: double);
operator =(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator =(const TDateTimeBase& rhs);
__fastcall TDateTime& operator =(const TDateTime& rhs);
__fastcall TDateTime& operator =(const double rhs);
__fastcall TDateTime& operator =(const int rhs);
```

Description

The assignment System::TDateTime (see page 3335)::= assigns the value of the rhs operand to the System::TDateTime::TDateTime (see page 3335) object. The assignment System::TDateTime (see page 3335)::= can take

a double (see page 3353) or integer value, which is assigned to the System::TDateTime::TDateTime (see page 3335) object such that the integral part of rhs is the number of days that have passed since 12/30/1899 and the fractional part is the time of day, represented as a fraction of 24 hours.

a System::TDateTime::TDateTimeBase (see page 3354) or System::TDateTime::TDateTime (see page 3335) object, whose value is assigned to the System::TDateTime::TDateTime (see page 3335) object. (Note that System::TDateTime::TDateTimeBase (see page 3354) is the base class for the System::TDateTime::TDateTime (see page 3335) class. It has a single member, a public Val (see page 3785) field that represents the value of the object).

38.1.17.5.10 -= Operator

This is the overview for the -= operator overload.

Overload List

	Name	Description
	TDateTime::-= (double) (see page 3352)	Decrements the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.
	TDateTime::-= (int) (see page 3352)	Decrements the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.

	TDateTime::-= (TDateTime&) (see page 3352)	Decrements the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.
	TDateTime::-= (TDateTimeBase&) (see page 3352)	Decrements the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.

38.1.17.5.10.1 TDateTime::-= Operator (TDateTime&)

Decrements the value of this System::TDateTime::TDateTime (see page 3335) object by a specified value.

Pascal

```
operator -=(const rhs: TDateTimeBase&);
operator -=(const rhs: TDateTime&);
operator -=(const rhs: double);
operator -=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator -=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator -=(const TDateTime& rhs);
__fastcall TDateTime& operator -=(const double rhs);
__fastcall TDateTime& operator -=(const int rhs);
```

Description

Use the -= System::TDateTime (see page 3335)::-= to subtract the value specified by rhs from this System::TDateTime::TDateTime (see page 3335) value. The rhs operand represents the number of days (including fractional days) to subtract.

38.1.17.5.11 TDateTime::== Operator

Compares two System::TDateTime::TDateTime (see page 3335) objects for equality.

Pascal

```
operator ==(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator ==(const TDateTime& rhs) const;
```

Description

The == System::TDateTime (see page 3335)::== returns true if the two objects have the same value, false otherwise.

38.1.17.5.12 TDateTime::> Operator

Pascal

```
operator >(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >(const TDateTime& rhs) const;
```

Description

This is >, a member of class TDateTime.

38.1.17.5.13 TDateTime::>= Operator

Pascal

```
operator >=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >=(const TDateTime& rhs) const;
```

Description

This is >=, a member of class TDateTime.

38.1.17.5.14 TDateTime::double Operator

Converts the System::TDateTime::TDateTime (see page 3335) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::TDateTime::double returns a double that represents this System::TDateTime::TDateTime (see page 3335) object. The integral part of the double is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

See Also

System::TDateTime::TDateTime (see page 3338)

38.1.17.5.15 TDateTime::int Operator

Converts the System::TDateTime::TDateTime (see page 3335) object to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::TDateTime::int returns an int that is the integer conversion of the date portion of this System::TDateTime::TDateTime (see page 3335) object. The resulting integer is the number of days that have passed since 12/30/1899.

See Also

System::TDateTime::TDateTime (see page 3338)

38.1.17.5.16 TDateTime::System::String Operator

Pascal

```
operator System::String;
```

C++

```
__fastcall operator System::String() const;
```

Description

This is System::String, a member of class TDateTime.

38.1.18 TDateTimeBase Class

Class Hierarchy

Pascal

```
TDateTimeBase = class;
```

C++

```
class TDateTimeBase;
```

File

systdate.h

Description

This is class System::TDateTimeBase.

38.1.18.1 TDateTimeBase Members

The following tables list the members exposed by TDateTimeBase.

Public Data Members

	Name	Description
	Val (see page 3354)	This is Val, a member of class TDateTimeBase.

38.1.18.2 TDateTimeBase Data Members

The data members of the TDateTimeBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 3354)	This is Val, a member of class TDateTimeBase.

38.1.18.2.1 TDateTimeBase::Val Data Member

Pascal

```
Val: double;
```

C++

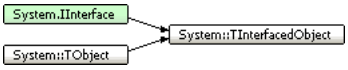
```
double Val;
```

Description

This is Val, a member of class TDateTimeBase.

38.1.19 TInterfacedObject Class

Class Hierarchy



Pascal

```
TInterfacedObject = class(TObject, IInterface);
```

C++

```
class TInterfacedObject : public TObject, public IInterface;
```

File

System.pas

Description

This is class System::TInterfacedObject.

38.1.19.1 TInterfacedObject Members

The following tables list the members exposed by TInterfacedObject.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

38.1.19.2 TInterfacedObject Data Members

The data members of the TInterfacedObject class are listed here.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

38.1.19.2.1 TInterfacedObject::FRefCount Data Member

Pascal

```
FRefCount: Integer;
```

C++

```
Integer FRefCount;
```

Description

This is FRefCount, a member of class TInterfacedObject.

38.1.19.3 TInterfacedObject Methods

The methods of the TInterfacedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.

38.1.19.3.1 TInterfacedObject::_AddRef Method

Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

Do not call the protected System::TInterfacedObject::_AddRef method directly. System::TInterfacedObject::_AddRef is called through the interface to indicate that another object is using the interfaced object's interface pointer. When the other object is through with the interface, it calls System::TInterfacedObject::_Release (see page 3357) through the same interface. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

System::TInterfacedObject::_AddRef increments the System::TInterfacedObject::RefCount (see page 3360) property.

See Also

System::TInterfacedObject::RefCount (see page 3360)

System::TInterfacedObject::_Release (see page 3357)

38.1.19.3.2 TInterfacedObject::_Release Method

Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

Do not call the protected _System::TInterfacedObject::_Release method directly. _System::TInterfacedObject::_Release is called through the IInterface (see page 3503) interface to indicate that another object is through with the interfaced object's interface pointer. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

_System::TInterfacedObject::_Release decrements the System::TInterfacedObject::RefCount (see page 3360) property.

See Also

System::TInterfacedObject::RefCount (see page 3360)

System::TInterfacedObject::_AddRef (see page 3357)

System::TInterfacedObject::BeforeDestruction (see page 3358)

38.1.19.3.3 TInterfacedObject::AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; virtual; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject (see page 3355) is allocated, System::TInterfacedObject::RefCount (see page 3360) is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount (see page 3360) to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

38.1.19.3.4 TInterfacedObject::BeforeDestruction Method

Checks that the reference count is zero before allowing the interfaced object to be destroyed.

Pascal

```
procedure BeforeDestruction; virtual; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TInterfacedObject::BeforeDestruction is called automatically immediately before the interfaced object's destructor. It checks that the reference count is zero, and, if it is not, raises an exception.

See Also

System::TInterfacedObject::RefCount (see page 3360)

System::TObject::free

38.1.19.3.5 TInterfacedObject::NewInstance Method

Allocates memory for each instance of an interfaced object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
TObject NewInstance();
```

Description

All constructors call System::TInterfacedObject::NewInstance automatically. System::TInterfacedObject::NewInstance calls System::TObject::InstanceSize (see page 3389) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TInterfacedObject::NewInstance directly.

System::TInterfacedObject::TInterfacedObject (see page 3355) overrides System::TInterfacedObject::NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a System::TInterfacedObject::RefCount (see page 3360) of zero. The System::TInterfacedObject::AfterConstruction (see page 3358) method decrements the value of System::TInterfacedObject::RefCount (see page 3360) after all constructors have executed.

See Also

System::TInterfacedObject::RefCount (see page 3360)

System::TObject::NewInstance (see page 3391)

System::TObject::InstanceSize (see page 3389)

System::TInterfacedObject::AfterConstruction (see page 3358)

38.1.19.3.6 TInterfacedObject::QueryInterface Method

Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

Do not call the protected System::TInterfacedObject::QueryInterface method directly. System::TInterfacedObject::QueryInterface is called through IInterface (see page 3503) to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and System::TInterfacedObject::QueryInterface returns S_OK (see page 3927). If the object does not support the interface, System::TInterfacedObject::QueryInterface returns E_NOINTERFACE (see page 3912).

If System::TInterfacedObject::QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

System::TInterfacedObject::_AddRef (see page 3357)

38.1.19.4 TInterfacedObject Properties

The properties of the TInterfacedObject class are listed here.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

38.1.19.4.1 TInterfacedObject::RefCount Property

Indicates the number of interface pointers currently dependent upon the object.

Pascal

```
property RefCount: Integer;
```

C++

```
__property Integer RefCount;
```

Description

System::TInterfacedObject::RefCount defines the lifetime of the object. When System::TInterfacedObject::RefCount is zero, the object is destroyed.

System::TInterfacedObject::RefCount is incremented by calls to the _AddRef (see page 3357) method and decremented by the _Release (see page 3357) method.

Note: If the destructor is called on a System::TInterfacedObject::TInterfacedObject (see page 3355) descendant that is not fully constructed (an exception was raised in the constructor), then System::TInterfacedObject::RefCount will have a value of 1. Otherwise, System::TInterfacedObject::RefCount will be zero.

38.1.20 TMetaClass Class

System::TMetaClass is the C++ representation of the Delphi class-reference type.

Class Hierarchy

Pascal

```
TMetaClass = class;
```

C++

```
class TMetaClass;
```

File

systobj.h

Description

System::TMetaClass allows operations to be performed directly on classes. This contrasts with class types, which can be instantiated as objects and which allow operations to be performed on those instances. The System::TMetaClass for a given class can be acquired by using the __classid keyword extension.

A variable of type System::TMetaClass can be NULL, which indicates that the variable does not currently reference a class.

Every class inherits a method function called ClassType, which returns a reference to the class of an object. The type of the value returned by ClassType is TClass (see page 3856) (System::TMetaClass*), which is a pointer to a System::TMetaClass.

The methods of System::TMetaClass correspond to the static methods of TObject (see page 3364). They have the same functionality but differ in parameters. Whereas the TObject (see page 3364) methods take a TClass (see page 3856)

(System::TMetaClass*) as the first parameter, this parameter is implicit in the System::TMetaClass methods.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
#include <memory>       //for STL auto_ptr class

TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TClassFinder> myClassFinder(new TClassFinder(MetaClass, False));
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
```

See Also

System::TObject (see page 3364)

38.1.21 TMonitor Record

Pascal

```
TMonitor = record
end;
```

C++

```
struct TMonitor {
public:
  struct TWaitingThread {
  };
};
```

File

System

Description

This is class System.TMonitor.

38.1.21.1 TMonitor Members

The following tables list the members exposed by TMonitor.

Public Methods

	Name	Description
	Enter (see page 3362)	This is the overview for the Enter method overload.
	Exit (see page 3363)	This is Exit, a member of class TMonitor.
	Pulse (see page 3363)	This is Pulse, a member of class TMonitor.
	PulseAll (see page 3363)	This is PulseAll, a member of class TMonitor.
	SetSpinCount (see page 3363)	This is SetSpinCount, a member of class TMonitor.
	TryEnter (see page 3364)	This is TryEnter, a member of class TMonitor.
	Wait (see page 3364)	This is Wait, a member of class TMonitor.

38.1.21.2 TMonitor Methods

The methods of the TMonitor class are listed here.

Public Methods

	Name	Description
	Enter (see page 3362)	This is the overview for the Enter method overload.
	Exit (see page 3363)	This is Exit, a member of class TMonitor.
	Pulse (see page 3363)	This is Pulse, a member of class TMonitor.
	PulseAll (see page 3363)	This is PulseAll, a member of class TMonitor.
	SetSpinCount (see page 3363)	This is SetSpinCount, a member of class TMonitor.
	TryEnter (see page 3364)	This is TryEnter, a member of class TMonitor.
	Wait (see page 3364)	This is Wait, a member of class TMonitor.

38.1.21.2.1 Enter Method

This is the overview for the Enter method overload.

Overload List

	Name	Description
	TMonitor.Enter (TObject) (see page 3362)	This is Enter, a member of class TMonitor.
	TMonitor.Enter (TObject, Cardinal) (see page 3363)	This is Enter, a member of class TMonitor.

38.1.21.2.1.1 TMonitor.Enter Method (TObject)

Pascal

```
class procedure Enter(AObject: TObject); overload; static;
```

C++

```
static __fastcall Enter(TObject * AObject);
```

Description

This is Enter, a member of class TMonitor.

38.1.21.2.1.2 TMonitor.Enter Method (TObject, Cardinal)

Pascal

```
class function Enter(AObject: TObject; Timeout: Cardinal): Boolean; overload; static;
```

C++

```
static __fastcall Boolean Enter(TObject * AObject, unsigned Timeout);
```

Description

This is Enter, a member of class TMonitor.

38.1.21.2.2 TMonitor.Exit Method

Pascal

```
class procedure Exit(AObject: TObject); overload; static;
```

C++

```
static __fastcall Exit(TObject * AObject);
```

Description

This is Exit, a member of class TMonitor.

38.1.21.2.3 TMonitor.Pulse Method

Pascal

```
class procedure Pulse(AObject: TObject); overload; static;
```

C++

```
static __fastcall Pulse(TObject * AObject);
```

Description

This is Pulse, a member of class TMonitor.

38.1.21.2.4 TMonitor.PulseAll Method

Pascal

```
class procedure PulseAll(AObject: TObject); overload; static;
```

C++

```
static __fastcall PulseAll(TObject * AObject);
```

Description

This is PulseAll, a member of class TMonitor.

38.1.21.2.5 TMonitor.SetSpinCount Method

Pascal

```
class procedure SetSpinCount(AObject: TObject; ASpinCount: Integer); static;
```

C++

```
static __fastcall SetSpinCount(TObject * AObject, int ASpinCount);
```

Description

This is SetSpinCount, a member of class TMonitor.

38.1.21.2.6 TMonitor.TryEnter Method

Pascal

```
class function TryEnter(AObject: TObject): Boolean; overload; static;
```

C++

```
static __fastcall Boolean TryEnter(TObject * AObject);
```

Description

This is TryEnter, a member of class TMonitor.

38.1.21.2.7 TMonitor.Wait Method

Pascal

```
class function Wait(AObject: TObject; Timeout: Cardinal): Boolean; overload; static;
```

C++

```
static __fastcall Boolean Wait(TObject * AObject, unsigned Timeout);
```

Description

This is Wait, a member of class TMonitor.

38.1.22 TObject Class

System::TObject is the ultimate ancestor of all objects and components.

Class Hierarchy

Pascal

```
TObject = class;
```

C++

```
class TObject;
```

File

System.pas

Description

System::TObject encapsulates fundamental behavior common to objects by introducing methods that

create, maintain and destroy instances of the object by allocating, initializing, and freeing required memory.

respond when object instances are created or destroyed.

return class-type and instance information on an object and runtime type information (RTTI) about its published properties.

support message handling.

support interfaces that the object implements.

Use System::TObject as an immediate base class when declaring simple objects that do not need to persist (are not saved and reloaded) and that do not need to be assigned to other objects.

Much of the capability of objects is established by methods that System::TObject introduces. Many of these methods are used internally by IDEs and are not intended for users to call directly. Others are overridden in descendant objects that have more complex behavior.

Although System::TObject is the based object of a component framework, not all objects are components. All component classes are descended from TComponent.

Note: System::TObject is never directly instantiated. Although it does not use programming language features that prevent instantiation, System::TObject is an abstract class.

See Also

TComponent

TPersistent

TObject Branch

TWinControl/TWidgetControl Branch

VCL Overview

38.1.22.1 TObject Members

The following tables list the members exposed by TObject.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.

	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

38.1.22.2 TObject::Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
Create();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject (see page 3364) allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

38.1.22.3 TObject::Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; virtual;
```

C++

```
virtual Destroy();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free (see page 3382) instead. Free (see page 3382) verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject (see page 3364) deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free (see page 3382) on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free (see page 3382) provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

38.1.22.4 TObject Methods

The methods of the TObject class are listed here.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

38.1.22.4.1 TObject::AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; virtual;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject::TObject (see page 3364) does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 3368)

OnCreate

System::TObject::Create (see page 3366)

38.1.22.4.2 TObject::BeforeDestruction Method

Responds before the first destructor executes.

Pascal

```
procedure BeforeDestruction; virtual;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TObject::BeforeDestruction is called automatically before the object's first destructor executes. Do not call it explicitly in your applications.

The System::TObject::BeforeDestruction method implemented in System::TObject::TObject (see page 3364) does nothing. Override this method when creating a class that takes some action before the object is destroyed. For example, TCustomForm overrides System::TObject::BeforeDestruction to generate an OnDestroy event.

Note: System::TObject::BeforeDestruction is not called when the object is destroyed before it is fully constructed. That is, if the object's constructor raises an exception, the destructor is called to dispose of the object, but System::TObject::BeforeDestruction is not called.

See Also

System::TObject::AfterConstruction (see page 3367)

OnDestroy

System::TObject::Destroy (see page 3366)

38.1.22.4.3 TObject::ClassInfo Method

Returns a pointer to the runtime type information (RTTI) table for the object type.

Pascal

```
class function ClassInfo: Pointer;
```

C++

```
Pointer ClassInfo();
```

Description

System::TObject::ClassInfo provides access to the RTTI table for a given object type.

Note: The format of the RTTI table is subject to change. Applications should use the other RTTI methods provided by System::TObject::TObject (see page 3364); these provide easy, consistent access to RTTI data.

Note: Some classes do not provide runtime type information. For these classes, System::TObject::ClassInfo returns nil (Delphi) or NULL (C++). All classes descended from TPersistent do provide runtime type information.

Note: The Delphi compiler does not provide RTTI by default. Most classes implemented in Delphi that need to provide RTTI are descended from TPersistent. If your Delphi class needs RTTI but not the full functionality of TPersistent, compile the code with the $M switch.

Note: The C++ compiler provides RTTI by default. This feature can be disabled as project option and selectively re-enabled using the __rtti keyword.

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

_typeinfo_Macro

38.1.22.4.4 TObject::ClassName Method

Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

Pascal

```
class function ClassName: string;
```

C++

```
string ClassName();
```

Description

Use System::TObject::ClassName to obtain the class name from an object instance or class reference. This is useful for differentiating object instances that are assigned to a variable that has the type of an ancestor class.

Note: In C++ code, call System::TObject::ClassName as a method to obtain an object's class name. Use the global static function to obtain the class name from a metaclass object.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = reinterpret_cast<TControl *>(Objects->Items[i]);
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName();
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
    }
  }
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TCoolBar *CoolBar = new TCoolBar(this); // The Ownder (Form1) will clean this up.
  CoolBar->Parent = this;
  CoolBar->Align = alTop;
  std::auto_ptr<TList> List(new TList);
```

```
    try
    {
      TControl *Control;
      Control = new TButton(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      Control = new TCheckBox(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      Control = new TEdit(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      AddBands(CoolBar, List.get());
    }
    catch (...)
    {
      ShowMessage("Some objects could not be added to Coolband");
    }
  }
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassNameIs (see page 3371)

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

System::TObject::ClassType (see page 3374)

38.1.22.4.5 TObject::ClassNameIs Method

Determines whether an object is of a specific type.

Pascal

```
class function ClassNameIs(const Name: string): Boolean;
```

C++

```
Boolean ClassNameIs(const string Name);
```

Description

System::TObject::ClassNameIs determines whether an object instance or class reference has a class name that matches a specified string. This is useful to query objects across modules or shared libraries.

Note: In C++ code, call System::TObject::ClassNameIs as a method to compare an object's class name. Use the global static function to compare the class name from a metaclass object.

See Also

System::TObject::ClassName (see page 3369)

System::TObject::InheritsFrom (see page 3388)

38.1.22.4.6 TObject::ClassParent Method

Returns the type of the immediate ancestor of a class.

Pascal

```
class function ClassParent: TClass;
```

C++

```
TClass ClassParent();
```

Description

System::TObject::ClassParent returns the name of the parent class for an object instance or class reference. For System::TObject::TObject (see page 3364), System::TObject::ClassParent returns nil (Delphi) or NULL (C++).

Avoid using System::TObject::ClassParent in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassParent.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 3388) method instead of System::TObject::ClassParent.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
```

```
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 3368)

System::TObject::ClassType (see page 3374)

System::TObject::InheritsFrom (see page 3388)

38.1.22.4.7 TObject::ClassType Method

Returns the class reference for the object's class.

Pascal

```
function ClassType: TClass;
```

C++

```
__fastcall TClass ClassType();
```

Description

Note: System::TObject::ClassType dynamically determines the type of an object and returns its class reference, or metaclass.

Avoid using System::TObject::ClassType in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassType.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 3388) method instead of System::TObject::ClassType.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
```

```
      if (ClassRef == NULL) // if it wasn't a control
        MessageBeep(0);
    }
  }
  /*
  To run this example, add the example code to a new project.
  The example code dynamically creates a TCoolbar and three
  TCoolBand objects populated with a windowed control in each
  TCoolBand.
  */
  void AddBands(TCoolBar *CoolBar, TList *Objects)
  {
    TControl *pCurrent;
    for (int i = 0; i < Objects->Count; i++)
    {
      pCurrent = reinterpret_cast<TControl *>(Objects->Items[i]);
      // only add windowed controls to the coolbar
      if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
      {
        // Placing the control onto the CoolBar
        // causes the TCoolBar object to create a TCoolBand
        // and place the control within the band.
        pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
        // Get the reference of the TCoolBand just created and assign text.
        String S = pCurrent->ClassName();
        CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
      }
    }
  }

  #include <memory>       //for STL auto_ptr class

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    TCoolBar *CoolBar = new TCoolBar(this); // The Ownder (Form1) will clean this up.
    CoolBar->Parent = this;
    CoolBar->Align = alTop;
    std::auto_ptr<TList> List(new TList);
    try
    {
      TControl *Control;
      Control = new TButton(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      Control = new TCheckBox(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      Control = new TEdit(CoolBar); // The Ownder (CoolBar) will clean this up.
      List->Add(Control);
      AddBands(CoolBar, List.get());
    }
    catch (...)
    {
      ShowMessage("Some objects could not be added to Coolband");
    }
  }
```

Delphi Examples:

```
  {
  This example shows how to obtain the ancestry of a component
  using the ClassType and ClassParent properties. It uses a
  button and a list box on a form. When the user clicks the
  button, the name of the button's class and the names of its
  parent classes are added to the list box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    ClassRef: TClass;
```

```
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 3368)

System::TObject::ClassName (see page 3369)

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

_classid

38.1.22.4.8 TObject::CleanupInstance Method

Performs finalization on long strings, variants, and interface variables within a class.

Pascal

```
procedure CleanupInstance;
```

C++

```
__fastcall CleanupInstance();
```

Description

Do not call System::TObject::CleanupInstance directly. System::TObject::CleanupInstance is called automatically when the object instance is destroyed.

System::TObject::CleanupInstance releases all long strings and variants. It sets long strings to empty and variants to Unassigned.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

delete

38.1.22.4.9 TObject::DefaultHandler Method

Provides the interface for a method that processes message records.

Pascal

```
procedure DefaultHandler(var Message); virtual;
```

C++

```
virtual DefaultHandler(var  Message);
```

Description

System::TObject::DefaultHandler is called by System::TObject::Dispatch (see page 3379) when it cannot find a method for a particular message. System::TObject::DefaultHandler provides message handling for all messages for which an object does not have specific handlers. Descendant classes that process messages override System::TObject::DefaultHandler according to the types of messages they handle.

Note: In a Delphi message-handling method, calling inherited results in a call to the ancestor's System::TObject::DefaultHandler method only if that ancestor does not specify a message method for the particular message being handled. Otherwise, calling inherited results in a call to the specific handler for that type of message.

Delphi Examples:

```
{
This example demonstrates the use of message dispatching
on TObject and any descendants. Using the same techniques
a form of message based OOP can be acheived where an object
"implements" a certain interface based on messages.
}

{ Define the object interface messages }
const
  INTF_MESSAGE_LOWER_STR = 1;
  INTF_MESSAGE_UPPER_STR = 2;

type
  { String container message that will be passed }
  TStrCntrMessage = packed record
    MessageId : Word;
    MsgText   : String;
  end;

  { String handler object that will serve requests }
  TStringHandler = class
  private
    procedure StrUpCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_UPPER_STR;

    procedure StrLowCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_LOWER_STR;

  public
    procedure DefaultHandler(var Message); override;

  end;

{ TStringHandler }

procedure TStringHandler.DefaultHandler(var Message);
var
  MessageId : Word absolute Message;
begin
  { Display out message then pass control to default handler }
  MessageDlg('Unhandled message ' + IntToStr(MessageId) +
    ' received!', mtInformation, [mbOK], 0);

  inherited;
end;
```

```
procedure TStringHandler.StrLowCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := LowerCase(AMessage.MsgText);
end;

procedure TStringHandler.StrUpCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := UpperCase(AMessage.MsgText);
end;

{ TForm2}

procedure TForm2.FormCreate(Sender: TObject);
var
  Hndlr: TStringHandler;
  Msg: TStrCntrMessage;
begin
  { Create object instance, dispatch a message and free it }
  Hndlr := TStringHandler.Create();

  { Lower-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_LOWER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Lower-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  { Upper-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_UPPER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Upper-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  Hndlr.Free;
end;
```

See Also

System::TObject::Dispatch (see page 3379)

38.1.22.4.10 TObject::Dispatch Method

Calls message-handling methods for the object, based on the contents of the Message parameter.

Pascal

```
procedure Dispatch(var Message); virtual;
```

C++

```
virtual Dispatch(var  Message);
```

Description

Call System::TObject::Dispatch to automatically pass messages to the appropriate message handler.

System::TObject::Dispatch determines whether a message is in the list of message handlers declared for the object. If the object does not handle the message, System::TObject::Dispatch then examines the message-handler list of the ancestor class, and continues checking ancestors until it either finds a specific handler or runs out of ancestors, in which case it calls

System::TObject::DefaultHandler (see page 3377).

The only assumption System::TObject::Dispatch makes about the data in Message is that the first two bytes contain a message ID—that is, an integer that determines which message handler System::TObject::Dispatch calls. Although any kind of data can be passed to System::TObject::Dispatch, most System::TObject::TObject (see page 3364) descendants expect a message record such as TMessage or a specific data structure type.

Delphi Examples:

```
{
This example demonstrates the use of message dispatching
on TObject and any descendants. Using the same techniques
a form of message based OOP can be acheived where an object
"implements" a certain interface based on messages.
}

{ Define the object interface messages }
const
  INTF_MESSAGE_LOWER_STR = 1;
  INTF_MESSAGE_UPPER_STR = 2;

type
  { String container message that will be passed }
  TStrCntrMessage = packed record
    MessageId : Word;
    MsgText   : String;
  end;

  { String handler object that will serve requests }
  TStringHandler = class
  private
    procedure StrUpCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_UPPER_STR;

    procedure StrLowCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_LOWER_STR;

  public
    procedure DefaultHandler(var Message); override;

  end;

{ TStringHandler }

procedure TStringHandler.DefaultHandler(var Message);
var
  MessageId : Word absolute Message;
begin
  { Display out message then pass control to default handler }
  MessageDlg('Unhandled message ' + IntToStr(MessageId) +
    ' received!', mtInformation, [mbOK], 0);

  inherited;
end;

procedure TStringHandler.StrLowCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := LowerCase(AMessage.MsgText);
end;

procedure TStringHandler.StrUpCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := UpperCase(AMessage.MsgText);
end;
```

```
{ TForm2}

procedure TForm2.FormCreate(Sender: TObject);
var
  Hndlr: TStringHandler;
  Msg: TStrCntrMessage;
begin
  { Create object instance, dispatch a message and free it }
  Hndlr := TStringHandler.Create();

  { Lower-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_LOWER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Lower-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  { Upper-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_UPPER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Upper-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  Hndlr.Free;
end;
```

See Also

System::TObject::DefaultHandler (see page 3377)

TMessage

38.1.22.4.11 TObject::Equals Method

Checks if the current instance and the parameter are equal.

Pascal

```
function Equals(Obj: TObject): Boolean; virtual;
```

C++

```
virtual Boolean Equals(TObject Obj);
```

Description

The function has one parameter, **Obj** of type TObject (see page 3364).

By default, the System::TObject::Equals method shows if the addresses corresponding to the object and the parameter are identical. Returns a boolean value that represents the equality between the two addresses.

Note: System::TObject::Equals is supposed to be overridden in user derived classes, in order to provide consumer objects with an equality determining function.

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

38.1.22.4.12 FieldAddress Method

This is the overview for the FieldAddress method overload.

Overload List

	Name	Description
	TObject::FieldAddress (ShortString) (see page 3382)	Returns the address of a published object field.
	TObject::FieldAddress (string) (see page 3382)	Returns the address of a published object field.

38.1.22.4.12.1 TObject::FieldAddress Method (ShortString)

Returns the address of a published object field.

Pascal

```
function FieldAddress(const Name: ShortString): Pointer; overload;
function FieldAddress(const Name: string): Pointer; overload;
```

C++

```
Pointer FieldAddress(const ShortString Name);
Pointer FieldAddress(const string Name);
```

Description

System::TObject::FieldAddress is used internally by the component streaming system to access a specified published field of an object. System::TObject::FieldAddress returns a pointer to the field if it exists. If the object has no published field by that name, System::TObject::FieldAddress returns nil (Delphi) or NULL (C++).

Programs should access and manipulate fields by using properties instead of System::TObject::FieldAddress.

38.1.22.4.13 TObject::Free Method

Destroys an object and frees its associated memory, if necessary.

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

Use System::TObject::Free to destroy an object. System::TObject::Free automatically calls the destructor if the object reference is not nil. Any object instantiated at runtime that does not have an owner should be destroyed by a call to System::TObject::Free so that it can be properly disposed of and its memory released. Unlike System::TObject::Destroy (see page 3366), System::TObject::Free is successful even if the object is nil; so if the object was never initialized, System::TObject::Free won't

result in an error.

When you call System::TObject::Free for a component, it calls System::TObject::Free for all components that it owns—that is, all components in its component list. Since a form owns all the controls and other components that are created on it in design mode, those components are automatically freed when the form is freed. By default, all forms are owned by the Application object; when the application terminates, it frees the Application object, which frees all forms. For objects that are not components, or for components created with a nil owner, be sure to call System::TObject::Free after you are finished with them; otherwise the allocated memory will not be usable until after the application terminates.

Warning: Never explicitly free a component within one of its own event handlers or the event handler of a component it owns or contains. For example, don't free a button, or the form that owns the button, in its OnClick event handler.

To free a form, call its Release (see page 3732) method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains are through executing.

Note: In C++ code, do not use System::TObject::Free to destroy an object. Use the delete keyword.

C++ Examples:

```
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
```

```
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

System::TObject::Destroy (see page 3366)

Release

Owner

FreeAndNil (see page 4245)

38.1.22.4.14 TObject::FreeInstance Method

Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.

Pascal

```
procedure FreeInstance; virtual;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

All destructors call System::TObject::FreeInstance automatically to deallocate memory that was allocated by overriding System::TObject::NewInstance (see page 3391).

Do not call System::TObject::FreeInstance directly. System::TObject::FreeInstance should be overridden if System::TObject::NewInstance (see page 3391) was overridden to change the way the object's instance data was allocated.

Like System::TObject::NewInstance (see page 3391), System::TObject::FreeInstance uses the value returned from System::TObject::InstanceSize (see page 3389) to deallocate the object's memory.

See Also

System::TObject::InstanceSize (see page 3389)

System::TObject::NewInstance (see page 3391)

38.1.22.4.15 TObject::GetHashCode Method

Returns an integer containing the hash code.

Pascal

```
function GetHashCode: Integer; virtual;
```

C++

```
virtual __fastcall Integer GetHashCode();
```

Description

By default, calling System::TObject::GetHashCode on an object, returns an integer representing the virtual address at which the

object is stored.

Note: System::TObject::GetHashCode is supposed to be overridden in user derived classes, in order to provide consumer objects with an integer hash code representation.

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

38.1.22.4.16 TObject::GetInterface Method

Retrieves a specified interface.

Pascal

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

C++

```
Boolean GetInterface(const TGUID IID, out  Obj);
```

Description

System::TObject::GetInterface retrieves the interface designated by a GUID or type name. The basic implementation of System::TObject::GetInterface uses the GUID specified in the IID parameter. If the specified interface is supported by the class, it is returned in the Obj parameter, and System::TObject::GetInterface has a return value of true. Otherwise Obj contains nil (Delphi) or NULL (C++), and System::TObject::GetInterface returns false.

Note: In Delphi code, IID can be an interface name. The compiler automatically translates this name into the corresponding GUID.

Note: In C++ code, use the templated version of System::TObject::GetInterface to obtain an interface from a DelphiInterface (see page 3263) object.

System::TObject::GetInterface is equivalent to the as operator (Delphi) and dynamic casts (C++), except that System::TObject::GetInterface does not raise an exception if the interface is not supported.

See Also

System::TInterfacedObject::QueryInterface (see page 3359)

38.1.22.4.17 TObject::GetInterfaceEntry Method

Returns the entry for a specific interface implemented in a class.

Pascal

```
class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
```

C++

```
PInterfaceEntry GetInterfaceEntry(const TGUID IID);
```

Description

System::TObject::GetInterfaceEntry returns the class entry for the interface specified by the IID parameter.

Note: In Delphi Code, IID can be an interface name. The compiler replaces this name with the actual GUID.

Note: COM objects can use System::TObject::GetInterfaceEntry to automate dispatch calls to a dual-IDispatch (see page 3498) interface.

38.1.22.4.18 TObject::GetInterfaceTable Method

Returns a pointer to a structure containing all of the interfaces implemented by a given class.

Pascal

```
class function GetInterfaceTable: PInterfaceTable;
```

C++

```
PInterfaceTable GetInterfaceTable();
```

Description

System::TObject::GetInterfaceTable returns the interface entries for the class. This list contains only interfaces implemented by this class, not its ancestors. To find the ancestor list, iteratively call System::TObject::ClassParent (see page 3372) and then call System::TObject::GetInterfaceTable on the value it returns. To find the entry for a specific interface, use the System::TObject::GetInterfaceEntry (see page 3387) method instead.

See Also

System::TObject::GetInterfaceEntry (see page 3387)

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

38.1.22.4.19 TObject::InheritsFrom Method

Determines the relationship of two object types.

Pascal

```
class function InheritsFrom(AClass: TClass): Boolean;
```

C++

```
Boolean InheritsFrom(TClass AClass);
```

Description

Use System::TObject::InheritsFrom to determine if a particular class type or object is an instance of a class or one of its descendants. System::TObject::InheritsFrom returns true if the object type specified in the aClass parameter is an ancestor of the object type or the type of the object itself. Otherwise, it returns false.

Note: System::TObject::InheritsFrom is similar to the Delphi is operator, but applies to class references.

Note: In C++ code, a non-static version of System::TObject::InheritsFrom is provided. This call is useful in determining whether a descendant class method or property can be used, given a variable of a base class. For example, use System::TObject::InheritsFrom to determine whether the Sender parameter in an event handler is of a particular class type or

one of its descendants.

See Also

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

System::TObject::ClassNameIs (see page 3371)

38.1.22.4.20 TObject::InitInstance Method

Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

Pascal

```
class function InitInstance(Instance: Pointer): TObject;
```

C++

```
TObject InitInstance(Pointer Instance);
```

Description

There should be no reason to call System::TObject::InitInstance directly. System::TObject::InitInstance is called by System::TObject::NewInstance (see page 3391) when an object is created. When overriding System::TObject::NewInstance (see page 3391), be sure to call System::TObject::InitInstance as the last statement.

System::TObject::InitInstance is not virtual, so you cannot override it: Instead, initialize any data for an object in the constructor.

See Also

System::TObject::Create (see page 3366)

System::TObject::NewInstance (see page 3391)

38.1.22.4.21 TObject::InstanceSize Method

Returns the size in bytes of each instance of the object type.

Pascal

```
class function InstanceSize: Longint;
```

C++

```
Longint InstanceSize();
```

Description

System::TObject::InstanceSize indicates how many bytes of memory are required for a class's instance data. System::TObject::InstanceSize is called from methods that allocate and deallocate memory. System::TObject::InstanceSize is not a virtual method, so it cannot be overridden. System::TObject::InstanceSize should be called only when implementing a custom version of System::TObject::NewInstance (see page 3391).

See Also

System::TObject::FreeInstance (see page 3386)

System::TObject::NewInstance (see page 3391)

38.1.22.4.22 MethodAddress Method

This is the overview for the MethodAddress method overload.

Overload List

	Name	Description
	TObject::MethodAddress (ShortString) (see page 3390)	Returns the address of a class method by name.
	TObject::MethodAddress (string) (see page 3390)	Returns the address of a class method by name.

38.1.22.4.22.1 TObject::MethodAddress Method (ShortString)

Returns the address of a class method by name.

Pascal

```
class function MethodAddress(const Name: ShortString): Pointer; overload;
class function MethodAddress(const Name: string): Pointer; overload;
```

C++

```
Pointer MethodAddress(const ShortString Name);
Pointer MethodAddress(const string Name);
```

Description

There are situations where it is useful to be able to invoke an object method without the need to hard code the method name in advance. Call System::TObject::MethodAddress to dynamically retrieve the address of such a method by specifying the method **Name** as a string.

An easy way to invoke the method is to define a procedure or function data type, such as :

type TProc = procedure of object;

Assign the object name, and the System::TObject::MethodAddress to a TMethod (see page 3806) variable, such as :

MethodVar.Data := Pointer(ObjectInstanceName);

MethodVar.Code := ObjectInstanceName.System::TObject::MethodAddress('MethodNameString');

And pass this in a call to a variable of the procedure or function type :

Proc := TProc(MethodVar);

Proc;

See Also

TMethod (see page 3806)

38.1.22.4.23 TObject::MethodName Method

Returns the name of a class method by address.

Pascal

```
class function MethodName(Address: Pointer): string;
```

C++

```
string MethodName(Pointer Address);
```

Description

There are situations where it is useful to be able to invoke an object method without the need to hard code the method name in advance. Call MethodAddress (see page 3390) to dynamically retrieve the address of such a method by specifying the method

name as a string.

MethodName is the converse to this process - by supplying a method **Address**, the name of the method is returned as a string.

38.1.22.4.24 TObject::NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; virtual;
```

C++

```
virtual TObject NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize (see page 3389) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance (see page 3386) to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance (see page 3389).

See Also

System::TObject::FreeInstance (see page 3386)

System::TObject::InstanceSize (see page 3389)

System::TObject::InitInstance (see page 3389)

38.1.22.4.25 TObject::SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; virtual;
```

C++

```
virtual HResult SafeCallException(TObject ExceptObject, Pointer ExceptAddr);
```

Description

System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject (see page 3364), System::TObject::SafeCallException simply returns E_UNEXPECTED (see page 3913). This is the appropriate response for classes that do no support interfaces

38.1.22.4.26 TObject::ToString Method

Returns a string containing the class name.

Pascal

```
function ToString: string; virtual;
```

C++

```
virtual __fastcall string ToString();
```

Description

By default, the System::TObject::ToString returns a string containing the class name of the instance that is being called.

Example

Calling System::TObject::ToString on a TButton instance returns a string containing "TButton".

Note: System::TObject::ToString is intended to be overridden in user derived classes, in order to provide consumer objects with a string representation.

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

38.1.22.4.27 TObject::UnitName Method

Returns a string containing the unit name.

Pascal

```
class function UnitName: string;
```

C++

```
string UnitName();
```

Description

By default, calling System::TObject::UnitName on an object, returns a string representing the unit name where it is contained.

Example

Calling System::TObject::UnitName on a TButton instance returns a string containing "StdCtrls (see page 1959)".

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

38.1.23 UnicodeString Class

System::UnicodeString is the C++ analog for the Delphi UnicodeString type.

Class Hierarchy

Pascal

```
UnicodeString = class;
```

C++

```
class UnicodeString;
```

File

ustring.h

Description

Delphi utilizes several string types. System::UnicodeString can contain both Unicode and ANSI strings. Support for this type includes the following features:

- strings as large as available memory;
- efficient use of memory through shared references;
- routines and operators that evaluate strings based on the current locale.

Despite its name, System::UnicodeString can represent both ANSI character set strings and Unicode strings.

System::UnicodeString variables that have not been assigned an initial value contain a zero-length string.

Note: Delphi also supports System::UnicodeString, but implements it as a primitive type rather than a class. By default, variables declared as type String are System::UnicodeString.

Note: To be compatible with Delphi, System::UnicodeString indexing is 1-based.

See Also

System::AnsiStringBase (see page 3185)

System::WideString (see page 3476)

System::UnicodeString::[] (see page 3424)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

System::UnicodeString::+= (see page 3425)

38.1.23.1 UnicodeString Members

The following tables list the members exposed by UnicodeString.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3399)	This is record System::UnicodeString::TStringFloatFormat.
	TStringLeadCharType (see page 3399)	This is record System::UnicodeString::TStringLeadCharType.

Structures

	Name	Description
	StrRec (see page 3398)	This is record System::UnicodeString::StrRec.

Protected Methods

	Name	Description
	GetRec (see page 3411)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3419)	Throws exception if index is out of range in System::UnicodeString (see page 3393).

Public Friends

Name	Description
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) (see page 3429)	This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) (see page 3430)	This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

Public Methods

	Name	Description
	~UnicodeString (see page 3396)	Frees an instance of System::UnicodeString (see page 3393).
	BytesOf (see page 3401)	This is BytesOf, a member of class UnicodeString.
	ByteType (see page 3401)	This is ByteType, a member of class UnicodeString.
	c_str (see page 3402)	Returns a pointer to the underlying string System::UnicodeString::data (see page 3406).
	cat_printf (see page 3402)	Appends to value of System::UnicodeString (see page 3393), given format string and its arguments.
	cat_sprintf (see page 3403)	Appends result of given format string and its arguments to end of System::UnicodeString (see page 3393).
	cat_vprintf (see page 3403)	Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3393).
	CodePage (see page 3404)	Gets code page of a System::UnicodeString (see page 3393).
	Compare (see page 3404)	Compares the System::UnicodeString (see page 3393) to another specified string case-sensitively.
	CompareIC (see page 3405)	Compares System::UnicodeString (see page 3393) to another specified string case insensitively.
	CurrToStr (see page 3405)	Converts a currency value to its string representation.
	CurrToStrF (see page 3405)	Converts currency value to its string representation using indicated float format convention.
	data (see page 3406)	Returns a pointer to the underlying string System::UnicodeString::data.
	Delete (see page 3406)	Removes specified number of characters from string.

	ElementSize (see page 3407)	Gets element size of data (see page 3406) in a System::UnicodeString (see page 3393).
	EnsureUnicode (see page 3407)	Ensure that string contains Unicode data (see page 3406).
	FloatToStrF (see page 3407)	Returns string representation of floating-point value.
	FmtLoadStr (see page 3408)	Formats output using a resourced format string.
	Format (see page 3409)	Formats string assembled from System::UnicodeString (see page 3393), format, and array arguments.
	FormatFloat (see page 3409)	Formats floating-point value given by `value` using format string given by `format`.
	Insert (see page 3411)	Inserts specified string into System::UnicodeString (see page 3393).
	IntToHex (see page 3412)	Converts number into string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3412)	Returns true if the specified character matches any character in `delimiters` string.
	IsEmpty (see page 3412)	Indicates if System::UnicodeString (see page 3393) is empty.
	IsLeadSurrogate (see page 3412)	This is IsLeadSurrogate, a member of class UnicodeString.
	IsPathDelimiter (see page 3413)	Indicates if specified character is a path delimiter.
	IsTrailSurrogate (see page 3413)	This is IsTrailSurrogate, a member of class UnicodeString.
	LastChar (see page 3413)	Returns pointer to last character in System::UnicodeString (see page 3393).
	LastDelimiter (see page 3413)	Returns index of rightmost character that matches any character in `delimiters` string.
	Length (see page 3414)	Returns length in characters of System::UnicodeString (see page 3393).
	LoadStr (see page 3414)	This is the overview for the LoadStr method overload.
	LoadString (see page 3415)	Loads string from compiled resources of specified module.
	LowerCase (see page 3415)	Returns lower-case version of System::UnicodeString (see page 3393).
	Pos (see page 3416)	Returns character index at which specified substring begins.
	printf (see page 3416)	Sets value of System::UnicodeString (see page 3393) instance, given format string and its arguments.
	RefCount (see page 3416)	Gets reference count of System::UnicodeString (see page 3393).
	SetLength (see page 3417)	Changes character length of System::UnicodeString (see page 3393).
	sprintf (see page 3417)	Sets value of System::UnicodeString (see page 3393), given format string and its arguments.
	StringOfChar (see page 3417)	Returns string with specified number of characters.
	SubString (see page 3418)	Returns specified substring of System::UnicodeString (see page 3393).
	swap (see page 3418)	Swaps System::UnicodeString (see page 3393) with another string.
	t_str (see page 3418)	Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).
	ToDouble (see page 3419)	Converts string to floating-point value.
	ToInt (see page 3419)	Converts string to integer, throwing exception if string is invalid.
	ToIntDef (see page 3420)	Converts string to integer, returning default value if string is invalid.
	Trim (see page 3420)	Returns new System::UnicodeString (see page 3393) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3420)	Returns new System::UnicodeString (see page 3393) without leading spaces or control characters.
	TrimRight (see page 3421)	Returns new System::UnicodeString (see page 3393) without trailing spaces or control characters.
	UnicodeString (see page 3396)	This is the overview for the UnicodeString constructor overload.
	Unique (see page 3421)	Makes System::UnicodeString (see page 3393) instance unique.
	UpperCase (see page 3421)	Returns upper-case version of System::UnicodeString (see page 3393).
	vprintf (see page 3422)	Sets value of System::UnicodeString (see page 3393) given format string and its arguments.
	w_str (see page 3422)	Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).

Public Operators

	Name	Description
	!= (see page 3423)	Compares two strings.
	[] (see page 3423)	This is the overview for the [] operator overload.
	+ (see page 3424)	Concatenates two System::UnicodeString (see page 3393) values.

	+= (see page 3425)	Append (see page 3624) System::UnicodeString (see page 3393).
	< (see page 3426)	Compares two strings.
	<= (see page 3426)	Compares two strings.
	= (see page 3427)	Assigns string `rhs` to System::UnicodeString (see page 3393).
	== (see page 3427)	Compares System::UnicodeString (see page 3393) with specified string.
	> (see page 3428)	Compares two strings.
	>= (see page 3428)	Compares two strings.

38.1.23.2 UnicodeString::~UnicodeString Destructor

Frees an instance of System::UnicodeString (see page 3393).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~UnicodeString();
```

Description

System::UnicodeString (see page 3393)::~UnicodeString (see page 3393) is the destructor for System::UnicodeString (see page 3393).

38.1.23.3 UnicodeString Constructor

This is the overview for the UnicodeString constructor overload.

Overload List

	Name	Description
	UnicodeString::UnicodeString () (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (__int64) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (AnsiStringT<codePage> &) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (char) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (char*) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (char*, int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (char16_t*, int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (char32_t*, int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (double) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).

	UnicodeString::UnicodeString (long) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (short) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (UnicodeString&) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (unsigned __int64) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (unsigned int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (unsigned long) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (unsigned short) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (wchar_t) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (wchar_t*) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (wchar_t*, int) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).
	UnicodeString::UnicodeString (WideString &) (see page 3397)	Creates instance of System::UnicodeString (see page 3393).

38.1.23.3.1 UnicodeString::UnicodeString Constructor ()

Creates instance of System::UnicodeString (see page 3393).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: UnicodeString&);
constructor Create(const src: wchar_t*; len: Integer);
constructor Create(const src: char*; len: Integer);
constructor Create(const src: wchar_t*);
constructor Create(const src: char16_t*; numChar16: Integer = -1);
constructor Create(const src: char32_t*; numChar32: Integer = -1);
constructor Create(src: char);
constructor Create(src: wchar_t);
constructor Create(src: short);
constructor Create(src: unsigned short);
constructor Create(src: Integer);
constructor Create(src: unsigned int);
constructor Create(src: long);
constructor Create(src: unsigned long);
constructor Create(src: __int64);
constructor Create(src: unsigned __int64);
constructor Create(src: double);
constructor Create(const src: WideString &);
constructor Create(const src: AnsiStringT<codePage> &);
```

C++

```
__fastcall UnicodeString();
__fastcall UnicodeString(const char* src);
__fastcall UnicodeString(const UnicodeString& src);
```

```
__fastcall UnicodeString(const wchar_t* src, int len);
__fastcall UnicodeString(const char* src, int len);
__fastcall UnicodeString(const wchar_t* src);
__fastcall UnicodeString(const char16_t* src, int numChar16 = -1);
__fastcall UnicodeString(const char32_t* src, int numChar32 = -1);
__fastcall UnicodeString(char src);
__fastcall UnicodeString(wchar_t src);
__fastcall UnicodeString(short src);
__fastcall UnicodeString(unsigned short src);
__fastcall UnicodeString(int src);
__fastcall UnicodeString(unsigned int src);
__fastcall UnicodeString(long src);
__fastcall UnicodeString(unsigned long src);
__fastcall UnicodeString(__int64 src);
__fastcall UnicodeString(unsigned __int64 src);
__fastcall UnicodeString(double src);
__fastcall UnicodeString(const WideString & src);
template <unsigned short codePage> __fastcall UnicodeString(const AnsiStringT<codePage> & src);
```

Description

Call System::UnicodeString::UnicodeString (see page 3393) to create and initialize a System::UnicodeString (see page 3393) instance. If the constructor is given no arguments, the resulting System::UnicodeString (see page 3393) instance is an empty string. If the constructor is passed a single character, the resulting System::UnicodeString (see page 3393) instance has character length one, and contains that character. All other versions of the constructor copy the string passed as an argument, converting characters to `wchar_t` if necessary. When constructing a System::UnicodeString (see page 3393) instance from an array of `char` or `wchar_t`, the `src` string should be null terminated, unless a second argument is given that indicates the number of characters in the array.

38.1.23.4 UnicodeString Structures

The structures of the UnicodeString class are listed here.

Structures

	Name	Description
	StrRec (see page 3398)	This is record System::UnicodeString::StrRec.

38.1.23.4.1 System::UnicodeString::StrRec Structure

Pascal

```
StrRec = record
  codePage: unsigned short;
  elemSize: unsigned short;
  refCnt: Integer;
  length: Integer;
end;
```

C++

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

File

ustring.h

Description

This is record System::UnicodeString::StrRec.

38.1.23.5 UnicodeString Enumerations

The enumerations of the UnicodeString class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3399)	This is record System::UnicodeString::TStringFloatFormat.
	TStringLeadCharType (see page 3399)	This is record System::UnicodeString::TStringLeadCharType.

38.1.23.5.1 System::UnicodeString::TStringFloatFormat Enumeration

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

ustring.h

Description

This is record System::UnicodeString::TStringFloatFormat.

38.1.23.5.2 System::UnicodeString::TStringLeadCharType Enumeration

Pascal

```
TStringLeadCharType = (
  ctNotLeadChar,
  ctbLeadSurrogate,
  ctTrailSurrogate
);
```

C++

```
enum TStringLeadCharType {
  ctNotLeadChar,
  ctbLeadSurrogate,
  ctTrailSurrogate
};
```

File

ustring.h

Description

This is record System::UnicodeString::TStringLeadCharType.

38.1.23.6 UnicodeString Methods

The methods of the UnicodeString class are listed here.

Protected Methods

	Name	Description
	GetRec (see page 3411)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3419)	Throws exception if index is out of range in System::UnicodeString (see page 3393).

Public Methods

	Name	Description
	BytesOf (see page 3401)	This is BytesOf, a member of class UnicodeString.
	ByteType (see page 3401)	This is ByteType, a member of class UnicodeString.
	c_str (see page 3402)	Returns a pointer to the underlying string System::UnicodeString::data (see page 3406).
	cat_printf (see page 3402)	Appends to value of System::UnicodeString (see page 3393), given format string and its arguments.
	cat_sprintf (see page 3403)	Appends result of given format string and its arguments to end of System::UnicodeString (see page 3393).
	cat_vprintf (see page 3403)	Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3393).
	CodePage (see page 3404)	Gets code page of a System::UnicodeString (see page 3393).
	Compare (see page 3404)	Compares the System::UnicodeString (see page 3393) to another specified string case-sensitively.
	CompareIC (see page 3405)	Compares System::UnicodeString (see page 3393) to another specified string case insensitively.
	CurrToStr (see page 3405)	Converts a currency value to its string representation.
	CurrToStrF (see page 3405)	Converts currency value to its string representation using indicated float format convention.
	data (see page 3406)	Returns a pointer to the underlying string System::UnicodeString::data.
	Delete (see page 3406)	Removes specified number of characters from string.
	ElementSize (see page 3407)	Gets element size of data (see page 3406) in a System::UnicodeString (see page 3393).
	EnsureUnicode (see page 3407)	Ensure that string contains Unicode data (see page 3406).
	FloatToStrF (see page 3407)	Returns string representation of floating-point value.
	FmtLoadStr (see page 3408)	Formats output using a resourced format string.
	Format (see page 3409)	Formats string assembled from System::UnicodeString (see page 3393), format, and array arguments.
	FormatFloat (see page 3409)	Formats floating-point value given by `value` using format string given by `format`.
	Insert (see page 3411)	Inserts specified string into System::UnicodeString (see page 3393).
	IntToHex (see page 3412)	Converts number into string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3412)	Returns true if the specified character matches any character in `delimiters` string.
	IsEmpty (see page 3412)	Indicates if System::UnicodeString (see page 3393) is empty.
	IsLeadSurrogate (see page 3412)	This is IsLeadSurrogate, a member of class UnicodeString.
	IsPathDelimiter (see page 3413)	Indicates if specified character is a path delimiter.
	IsTrailSurrogate (see page 3413)	This is IsTrailSurrogate, a member of class UnicodeString.
	LastChar (see page 3413)	Returns pointer to last character in System::UnicodeString (see page 3393).

	LastDelimiter (see page 3413)	Returns index of rightmost character that matches any character in `delimiters` string.
	Length (see page 3414)	Returns length in characters of System::UnicodeString (see page 3393).
	LoadStr (see page 3414)	This is the overview for the LoadStr method overload.
	LoadString (see page 3415)	Loads string from compiled resources of specified module.
	LowerCase (see page 3415)	Returns lower-case version of System::UnicodeString (see page 3393).
	Pos (see page 3416)	Returns character index at which specified substring begins.
	printf (see page 3416)	Sets value of System::UnicodeString (see page 3393) instance, given format string and its arguments.
	RefCount (see page 3416)	Gets reference count of System::UnicodeString (see page 3393).
	SetLength (see page 3417)	Changes character length of System::UnicodeString (see page 3393).
	sprintf (see page 3417)	Sets value of System::UnicodeString (see page 3393), given format string and its arguments.
	StringOfChar (see page 3417)	Returns string with specified number of characters.
	SubString (see page 3418)	Returns specified substring of System::UnicodeString (see page 3393).
	swap (see page 3418)	Swaps System::UnicodeString (see page 3393) with another string.
	t_str (see page 3418)	Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).
	ToDouble (see page 3419)	Converts string to floating-point value.
	ToInt (see page 3419)	Converts string to integer, throwing exception if string is invalid.
	ToIntDef (see page 3420)	Converts string to integer, returning default value if string is invalid.
	Trim (see page 3420)	Returns new System::UnicodeString (see page 3393) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3420)	Returns new System::UnicodeString (see page 3393) without leading spaces or control characters.
	TrimRight (see page 3421)	Returns new System::UnicodeString (see page 3393) without trailing spaces or control characters.
	Unique (see page 3421)	Makes System::UnicodeString (see page 3393) instance unique.
	UpperCase (see page 3421)	Returns upper-case version of System::UnicodeString (see page 3393).
	vprintf (see page 3422)	Sets value of System::UnicodeString (see page 3393) given format string and its arguments.
	w_str (see page 3422)	Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).

38.1.23.6.1 UnicodeString::BytesOf Method

Pascal

```
function BytesOf: DynamicArray<System::Byte>;
```

C++

```
__fastcall DynamicArray<System::Byte> BytesOf() const;
```

Description

This is BytesOf, a member of class UnicodeString.

38.1.23.6.2 UnicodeString::ByteType Method

Pascal

```
function ByteType(index: Integer): TStringLeadCharType;
```

C++

```
__fastcall TStringLeadCharType ByteType(int index) const;
```

Description

This is ByteType, a member of class UnicodeString.

38.1.23.6.3 UnicodeString::c_str Method

Returns a pointer to the underlying string System::UnicodeString::data (see page 3406).

Pascal

```
function c_str: wchar_t*;
```

C++

```
__fastcall wchar_t* c_str() const;
```

Description

System::UnicodeString::c_str returns a **wchar_t** pointer to the location in memory where the value of the System::UnicodeString (see page 3393) object is stored. If the System::UnicodeString (see page 3393) is unassigned, System::UnicodeString::c_str returns a **wchar_t** pointer to the empty string ("").

Usually, the value returned by System::UnicodeString::c_str points to the internal character array referenced by the System::UnicodeString::data (see page 3406) function. This pointer is valid until the System::UnicodeString (see page 3393) is next modified (for example, when the System::UnicodeString::SetLength (see page 3417) method is called or the System::UnicodeString (see page 3393) goes out of scope). However, if the internal array is NULL, System::UnicodeString::c_str returns a **wchar_t** pointer to the empty string ("").

The System::UnicodeString::c_str method is intended primarily for reading the value of the System::UnicodeString (see page 3393). To modify the System::UnicodeString (see page 3393)'s value, use the [] operator or System::UnicodeString (see page 3393) methods such as System::UnicodeString::Insert (see page 3411) and System::UnicodeString::Delete (see page 3406).

See Also

System::UnicodeString::data (see page 3406)

System::UnicodeString::Delete (see page 3406)

System::UnicodeString::Insert (see page 3411)

System::UnicodeString::SetLength (see page 3417)

38.1.23.6.4 UnicodeString::cat_printf Method

Appends to value of System::UnicodeString (see page 3393), given format string and its arguments.

Pascal

```
function cat_printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::cat_printf to modify the value of the System::UnicodeString (see page 3393) by appending the result of using a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the number of characters appended to the System::UnicodeString (see page 3393).

See Also

System::UnicodeString::cat_sprintf (see page 3403)

System::UnicodeString::cat_vprintf (see page 3403)

System::UnicodeString::Format (see page 3409)

System::UnicodeString::printf (see page 3416)

38.1.23.6.5 UnicodeString::cat_sprintf Method

Appends result of given format string and its arguments to end of System::UnicodeString (see page 3393).

Pascal

```
function cat_sprintf(const format: wchar_t*; : ...): UnicodeString&; cdecl;
```

C++

```
__cdecl UnicodeString& cat_sprintf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::cat_sprintf to append to the value of the System::UnicodeString (see page 3393) object, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

For example, the following code

```
UnicodeString Salutation = "Mr."
UnicodeString s = "";
s.cat_sprintf("Dear %s ", Salutation);
s.cat_sprintf("%s,", "Smith");
```

results in an System::UnicodeString (see page 3393) object with the value

```
Dear Mr. Smith,
```

This method returns a reference to the System::UnicodeString (see page 3393) object (*this) with the resulting value.

See Also

System::UnicodeString::cat_printf (see page 3402)

System::UnicodeString::cat_vprintf (see page 3403)

System::UnicodeString::Format (see page 3409)

System::UnicodeString::sprintf (see page 3417)

38.1.23.6.6 UnicodeString::cat_vprintf Method

Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3393).

Pascal

```
function cat_vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const wchar_t* format, va_list);
```

Description

Use System::UnicodeString::cat_vprintf to append to the value of the System::UnicodeString (see page 3393) object, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using `va_start`.

This method returns the number of characters appended to the System::UnicodeString (see page 3393) object.

See Also

System::UnicodeString::Format (see page 3409)

System::UnicodeString::cat_printf (see page 3402)

System::UnicodeString::cat_sprintf (see page 3403)

System::UnicodeString::vprintf (see page 3422)

arg

start

38.1.23.6.7 UnicodeString::CodePage Method

Gets code page of a System::UnicodeString (see page 3393).

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::UnicodeString::CodePage gets the code page of a System::UnicodeString (see page 3393) instance. If the instance is empty, it returns the system default Unicode code page.

See Also

System::UnicodeString::GetRec (see page 3411)

38.1.23.6.8 UnicodeString::Compare Method

Compares the System::UnicodeString (see page 3393) to another specified string case-sensitively.

Pascal

```
function Compare(const rhs: UnicodeString&): Integer;
```

C++

```
__fastcall int Compare(const UnicodeString& rhs) const;
```

Description

Compares a System::UnicodeString (see page 3393) instance to `rhs`, with case sensitivity. Use System::UnicodeString::CompareIC (see page 3405) to compare without case sensitivity. The compare operation should be the same as using the System::UnicodeString (see page 3393) comparison operators. System::UnicodeString::Compare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

System::UnicodeString::CompareIC (see page 3405)

38.1.23.6.9 UnicodeString::CompareIC Method

Compares System::UnicodeString (see page 3393) to another specified string case insensitively.

Pascal

```
function CompareIC(const rhs: UnicodeString&): Integer;
```

C++

```
__fastcall int CompareIC(const UnicodeString& rhs) const;
```

Description

System::UnicodeString::CompareIC compares a System::UnicodeString (see page 3393) instance to `rhs`, without case sensitivity. Use System::UnicodeString::Compare (see page 3404) to compare with case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

System::UnicodeString::Compare (see page 3404)

38.1.23.6.10 UnicodeString::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency): String; static;
```

C++

```
static __fastcall UnicodeString CurrToStr(Currency value);
```

Description

System::UnicodeString::CurrToStr converts the currency value given by `value` to its string representation. The conversion uses the sffGeneral floating-point conversion rules (see System::UnicodeString::FloatToStrF (see page 3407)). For further details, see System::UnicodeString::CurrToStrF (see page 3405).

See Also

System::UnicodeString::CurrToStrF (see page 3405)

System::UnicodeString::FloatToStrF (see page 3407)

38.1.23.6.11 UnicodeString::CurrToStrF Method

Converts currency value to its string representation using indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer): String;
static;
```

C++

```
static __fastcall UnicodeString CurrToStrF(Currency value, TStringFloatFormat format, int
digits);
```

Description

System::UnicodeString::CurrToStrF converts the currency value given by `value` to its string representation using the indicated float format convention. A call to System::UnicodeString::CurrToStrF corresponds to a call to System::UnicodeString::FloatToStrF (see page 3407) with an implied precision of 19 digits.

See Also

System::UnicodeString::FloatToStrF (see page 3407)

38.1.23.6.12 UnicodeString::data Method

Returns a pointer to the underlying string System::UnicodeString::data.

Pascal

```
function data: const void*;
```

C++

```
__fastcall const void* data() const;
```

Description

System::UnicodeString::data returns a **wchar_t** pointer to the System::UnicodeString (see page 3393) object that it uses to store its value. This is the same value as the one returned by the System::UnicodeString::c_str (see page 3402) method, except that System::UnicodeString::c_str (see page 3402) returns a pointer to an empty string when the string is unassigned.

See Also

System::UnicodeString::c_str (see page 3402)

38.1.23.6.13 UnicodeString::Delete Method

Removes specified number of characters from string.

Pascal

```
function Delete(index: Integer; count: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& Delete(int index, int count);
```

Description

System::UnicodeString::Delete modifies the System::UnicodeString (see page 3393) object to remove `count` characters from the string beginning with the character at `index`, where 1 is the index of the first character. It returns the resulting modified string (*this).

If `index` is larger than the character length of the System::UnicodeString (see page 3393) object or less than 1, no characters are deleted.

If `count` specifies more characters than remain starting at the `index`, System::UnicodeString::Delete removes the rest of the string. If `count` is less than 0, no characters are deleted.

38.1.23.6.14 UnicodeString::ElementSize Method

Gets element size of data (see page 3406) in a System::UnicodeString (see page 3393).

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::UnicodeString::ElementSize gets the element size in bytes of the data (see page 3406) in a System::UnicodeString (see page 3393) instance. If the instance is empty, this returns 2.

See Also

System::UnicodeString::data (see page 3406)

System::UnicodeString::GetRec (see page 3411)

38.1.23.6.15 UnicodeString::EnsureUnicode Method

Ensure that string contains Unicode data (see page 3406).

Pascal

```
function EnsureUnicode: UnicodeString&;
```

C++

```
__fastcall UnicodeString& EnsureUnicode();
```

Description

If the string data (see page 3406) in the System::UnicodeString (see page 3393) instance is not Unicode, System::UnicodeString::EnsureUnicode converts it to Unicode.

38.1.23.6.16 UnicodeString::FloatToStrF Method

Returns string representation of floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): String; static;
```

C++

```
static __fastcall UnicodeString FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

The `format` parameter controls the format of the resulting string. The `precision` parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the `digits` parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the `digits` parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3242) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the `CurrencyString`, `CurrencyFormat`, `NegCurrFormat`, `ThousandSeparator`, and `DecimalSeparator` global variables, all of which are initialized from the Currency (see page 3242) Format (see page 3409) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the `Digits` parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the `DecimalSeparator` and `ThousandSeparator` global variables.

If the given value is NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

38.1.23.6.17 UnicodeString::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer): String; static;
```

C++

```
static __fastcall UnicodeString FmtLoadStr(int ident, const TVarRec * args, int size);
```

Description

System::UnicodeString::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the `args` array, as a parameter to the System::UnicodeString::Format (see page 3409) function. `Ident` is the string resource ID of the desired format string.

See Also

System::UnicodeString::Format (see page 3409)

System::UnicodeString::LoadStr (see page 3414)

System::UnicodeString::LoadString (see page 3415)

38.1.23.6.18 UnicodeString::Format Method

Formats string assembled from System::UnicodeString (see page 3393), format, and array arguments.

Pascal

```
function Format(const format: UnicodeString&; const args: TVarRec *; size: Integer): String;
static;
```

C++

```
static __fastcall UnicodeString Format(const UnicodeString& format, const TVarRec * args, int
size);
```

Description

Use System::UnicodeString::Format to obtain a System::UnicodeString (see page 3393) object whose value is the result of applying an array of arguments (`args`) to a given standard C++ format specifier string (`format`). The last parameter (`size`) is the 1-based index of the last element in `args` (one less than the number of arguments).

Note that System::UnicodeString::Format is a static method. The returned value is a newly allocated System::UnicodeString (see page 3393), even if the method is called with an existing System::UnicodeString (see page 3393) instance.

See Also

System::UnicodeString::printf (see page 3416)

System::UnicodeString::sprintf (see page 3417)

38.1.23.6.19 UnicodeString::FormatFloat Method

Formats floating-point value given by `value` using format string given by `format`.

Pascal

```
function FormatFloat(const format: UnicodeString&; const value: long double&): String; static;
```

C++

```
static __fastcall UnicodeString FormatFloat(const UnicodeString& format, const long double&
value);
```

Description

The following format specifiers are supported in the `format` string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the `DecimalSeparator` global variable, whose default value is specified in the Number Format (see page 3409) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output has thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the `ThousandSeparator` global variable, whose default value is specified in the Number Format (see page 3409) of the International section in the Windows Control Panel.

E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to System::UnicodeString::FloatToStrF (see page 3407) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3409) string-1234-12340.50

Format (see page 3409) string-	**1234**	**-1234**	**0.5**	**0**
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

See Also

System::UnicodeString::FloatToStrF (see page 3407)

38.1.23.6.20 GetRec Method

This is the overview for the GetRec method overload.

Overload List

	Name	Description
	UnicodeString::GetRec () (see page 3411)	Gets a struct describing a System::UnicodeString (see page 3393).
	UnicodeString::GetRec () (see page 3411)	Gets a struct describing a System::UnicodeString (see page 3393).

38.1.23.6.20.1 UnicodeString::GetRec Method ()

Gets a struct describing a System::UnicodeString (see page 3393).

Pascal

```
function GetRec: const StrRec&;
function GetRec: StrRec&;
```

C++

```
const StrRec& GetRec() const;
StrRec& GetRec();
```

Description

System::UnicodeString::GetRec gets a struct that describes the contents of a System::UnicodeString (see page 3393) instance. The struct has this format:

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

38.1.23.6.21 UnicodeString::Insert Method

Inserts specified string into System::UnicodeString (see page 3393).

Pascal

```
function Insert(const str: UnicodeString&; index: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& Insert(const UnicodeString& str, int index);
```

Description

System::UnicodeString::Insert inserts the string `str` into this System::UnicodeString (see page 3393) object beginning at the position `index`, where 1 is the first position in the string. It returns the resulting modified string (*this).

If `index` is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to System::UnicodeString::Length (see page 3414), turning the operation into an append.

If the `str` parameter is an empty string, System::UnicodeString::Insert does nothing.

System::UnicodeString::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer string.

Note: The `index` parameter always specifies the character, not the byte, index.

See Also

System::UnicodeString::Length (see page 3414)

38.1.23.6.22 UnicodeString::IntToHex Method

Converts number into string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer): String; static;
```

C++

```
static __fastcall UnicodeString IntToHex(int value, int digits);
```

Description

`Value` is the number to convert. `Digits` indicates the minimum number of hexadecimal digits.

38.1.23.6.23 UnicodeString::IsDelimiter Method

Returns true if the specified character matches any character in `delimiters` string.

Pascal

```
function IsDelimiter(const delimiters: UnicodeString&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const UnicodeString& delimiters, int index) const;
```

Description

System::UnicodeString::IsDelimiter returns true if the character indicated by `index` (counting from 1) is a character contained in `delimiters`. If the character is not in `delimiters`, System::UnicodeString::IsDelimiter returns false.

See Also

System::UnicodeString::LastDelimiter (see page 3413)

38.1.23.6.24 UnicodeString::IsEmpty Method

Indicates if System::UnicodeString (see page 3393) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::UnicodeString::IsEmpty returns true if a System::UnicodeString (see page 3393) object is empty, false otherwise.

38.1.23.6.25 UnicodeString::IsLeadSurrogate Method

Pascal

```
function IsLeadSurrogate(index: Integer): bool;
```

C++

```
__fastcall bool IsLeadSurrogate(int index) const;
```

Description

This is IsLeadSurrogate, a member of class UnicodeString.

38.1.23.6.26 UnicodeString::IsPathDelimiter Method

Indicates if specified character is a path delimiter.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::UnicodeString::IsPathDelimiter returns true if the character indicated by `index` (counting from 1) is the path delimiter ('\' on Windows, '/' on Linux).

38.1.23.6.27 UnicodeString::IsTrailSurrogate Method

Pascal

```
function IsTrailSurrogate(index: Integer): bool;
```

C++

```
__fastcall bool IsTrailSurrogate(int index) const;
```

Description

This is IsTrailSurrogate, a member of class UnicodeString.

38.1.23.6.28 UnicodeString::LastChar Method

Returns pointer to last character in System::UnicodeString (see page 3393).

Pascal

```
function LastChar: wchar_t*;
```

C++

```
__fastcall wchar_t* LastChar() const;
```

Description

Returns a pointer to the last character in a System::UnicodeString (see page 3393) instance. This function supports multibyte characters, returning a multibyte character when appropriate.

38.1.23.6.29 UnicodeString::LastDelimiter Method

Returns index of rightmost character that matches any character in `delimiters` string.

Pascal

```
function LastDelimiter(const delimiters: UnicodeString&): Integer;
```

C++

```
__fastcall int LastDelimiter(const UnicodeString& delimiters) const;
```

Description

System::UnicodeString::LastDelimiter returns the 1-based character index in the System::UnicodeString (see page 3393) object of the rightmost character that matches any character in `delimiters` except NULL The System::UnicodeString (see page 3393) object may contain multibyte characters; `delimiters` must contain non-NULL characters.

See Also

System::UnicodeString::IsDelimiter (see page 3412)

38.1.23.6.30 UnicodeString::Length Method

Returns length in characters of System::UnicodeString (see page 3393).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::UnicodeString::Length returns the number of characters (not necessarily bytes) in a System::UnicodeString (see page 3393) object.

38.1.23.6.31 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	UnicodeString::LoadStr (HINSTANCE, int) (see page 3414)	Loads string from compiled resources of specified module.
	UnicodeString::LoadStr (int) (see page 3414)	Loads string from compiled resources of specified module.

38.1.23.6.31.1 UnicodeString::LoadStr Method (HINSTANCE, int)

Loads string from compiled resources of specified module.

Pascal

```
function LoadStr(ident: Integer): String; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer): String; static;
```

C++

```
static __fastcall UnicodeString LoadStr(int ident);
static __fastcall UnicodeString LoadStr(HINSTANCE hInstance, int ident);
```

Description

System::UnicodeString::LoadStr loads a specified string resource.

`ident` is the resource identifier of the string.

`hInstance` is the instance handle of the module that contains the string resource. If `hInstance` is not given,

System::UnicodeString::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::UnicodeString::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::UnicodeString::LoadString (see page 3415)

System::UnicodeString::FmtLoadStr (see page 3408)

38.1.23.6.32 UnicodeString::LoadString Method

Loads string from compiled resources of specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& LoadString(HINSTANCE hInstance, int ident);
```

Description

System::UnicodeString::LoadString loads a specified string resource.

`ident` is the resource identifier of the string.

`hInstance` is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::UnicodeString::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::UnicodeString::LoadStr (see page 3414)

System::UnicodeString::FmtLoadStr (see page 3408)

38.1.23.6.33 UnicodeString::LowerCase Method

Returns lower-case version of System::UnicodeString (see page 3393).

Pascal

```
function LowerCase: String;
```

C++

```
__fastcall UnicodeString LowerCase() const;
```

Description

System::UnicodeString::LowerCase returns a new System::UnicodeString (see page 3393) instance that contains all characters in this string, converted to lower case. The conversion uses the current locale when appropriate. The original System::UnicodeString (see page 3393) instance remains unchanged.

See Also

System::UnicodeString::UpperCase (see page 3421)

38.1.23.6.34 UnicodeString::Pos Method

Returns character index at which specified substring begins.

Pascal

```
function Pos(const subStr: UnicodeString&): Integer;
```

C++

```
__fastcall int Pos(const UnicodeString& subStr) const;
```

Description

System::UnicodeString::Pos returns the character index in the System::UnicodeString (see page 3393) instance at which the substring `subStr` begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::UnicodeString (see page 3393), System::UnicodeString::Pos returns 0.

See Also

System::UnicodeString::SubString (see page 3418)

38.1.23.6.35 UnicodeString::printf Method

Sets value of System::UnicodeString (see page 3393) instance, given format string and its arguments.

Pascal

```
function printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::printf to set the value of a System::UnicodeString (see page 3393) instance, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the length in characters of the final formatted string.

See Also

System::UnicodeString::cat_printf (see page 3402)

System::UnicodeString::Format (see page 3409)

System::UnicodeString::sprintf (see page 3417)

System::UnicodeString::vprintf (see page 3422)

38.1.23.6.36 UnicodeString::RefCount Method

Gets reference count of System::UnicodeString (see page 3393).

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::UnicodeString::RefCount gets the reference count of a System::UnicodeString (see page 3393) instance. If the

instance is empty, it returns 0.

See Also

System::UnicodeString::GetRec (see page 3411)

38.1.23.6.37 UnicodeString::SetLength Method

Changes character length of System::UnicodeString (see page 3393).

Pascal

```
function SetLength(newLength: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& SetLength(int newLength);
```

Description

System::UnicodeString::SetLength changes the string's character length to `newLength`, truncating if necessary.

System::UnicodeString::SetLength reallocates the underlying array of characters that stores the value of the System::UnicodeString (see page 3393) instance. Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::UnicodeString (see page 3393) instance, an EOutOfMemory exception is thrown. Following a call to System::UnicodeString::SetLength, the System::UnicodeString (see page 3393) instance is guaranteed not to be a shared reference.

System::UnicodeString::SetLength returns the modified System::UnicodeString (see page 3393) object (*this)

38.1.23.6.38 UnicodeString::sprintf Method

Sets value of System::UnicodeString (see page 3393), given format string and its arguments.

Pascal

```
function sprintf(const format: wchar_t*; : ...): UnicodeString&; cdecl;
```

C++

```
__cdecl UnicodeString& sprintf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::sprintf to set to the value of the System::UnicodeString (see page 3393) instance, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

This method returns a reference to the System::UnicodeString (see page 3393) instance (*this) with the resulting value.

See Also

System::UnicodeString::Format (see page 3409)

System::UnicodeString::cat_sprintf (see page 3403)

System::UnicodeString::printf (see page 3416)

System::UnicodeString::vprintf (see page 3422)

38.1.23.6.39 UnicodeString::StringOfChar Method

Returns string with specified number of characters.

Pascal

```
function StringOfChar(ch: wchar_t; count: Integer): String; static;
```

C++

```
static __fastcall UnicodeString StringOfChar(wchar_t ch, int count);
```

Description

System::UnicodeString::StringOfChar returns a string containing `count` characters with the character value given by `ch`. For example,

```
UnicodeString (see page 3393) s = System::UnicodeString::UnicodeString (see page 3393)().StringOfChar('A', 4);
```

sets `s` to the string "AAAA".

38.1.23.6.40 UnicodeString::SubString Method

Returns specified substring of System::UnicodeString (see page 3393).

Pascal

```
function SubString(index: Integer; count: Integer): String;
```

C++

```
__fastcall UnicodeString SubString(int index, int count) const;
```

Description

System::UnicodeString::SubString returns a new System::UnicodeString (see page 3393) instance that is a substring of this System::UnicodeString (see page 3393) instance. The substring contains `count` characters beginning at `index`, which is a character index.

See Also

System::UnicodeString::Pos (see page 3416)

38.1.23.6.41 UnicodeString::swap Method

Swaps System::UnicodeString (see page 3393) with another string.

Pascal

```
function swap(var other: String): UnicodeString&;
```

C++

```
UnicodeString& swap(UnicodeString& other);
```

Description

System::UnicodeString::swap swaps a System::UnicodeString (see page 3393) with a given string. Swapping is not done by copying, but by swapping references. This is a very efficient operation, well suited to sorting strings, for example.

38.1.23.6.42 UnicodeString::t_str Method

Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).

Pascal

```
function t_str: char*;
```

C++

```
__fastcall char* t_str();
```

Description

System::UnicodeString::t_str behaves the same as System::UnicodeString::c_str (see page 3402).

See Also

System::UnicodeString::c_str (see page 3402)

38.1.23.6.43 UnicodeString::ThrowIfOutOfRange Method

Throws exception if index is out of range in System::UnicodeString (see page 3393).

Pascal

```
procedure ThrowIfOutOfRange(idx: Integer); cdecl;
```

C++

```
__cdecl void ThrowIfOutOfRange(int idx) const;
```

Description

System::UnicodeString::ThrowIfOutOfRange throws an `ERangeError` exception if the given 1-based character index is out of range for a System::UnicodeString (see page 3393) instance. Specifically, it throws an exception if the index is less than 1 or greater than the character length of the string.

38.1.23.6.44 UnicodeString::ToDouble Method

Converts string to floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::UnicodeString::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The `DecimalSeparator` global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

38.1.23.6.45 UnicodeString::ToInt Method

Converts string to integer, throwing exception if string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::UnicodeString::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a

valid value, an EConvertError exception is thrown.

See Also

System::UnicodeString::ToIntDef (see page 3420)

38.1.23.6.46 UnicodeString::ToIntDef Method

Converts string to integer, returning default value if string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::UnicodeString::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by `defaultValue` is returned.

See Also

System::UnicodeString::ToInt (see page 3419)

38.1.23.6.47 UnicodeString::Trim Method

Returns new System::UnicodeString (see page 3393) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim: String;
```

C++

```
__fastcall UnicodeString Trim() const;
```

Description

System::UnicodeString::Trim returns a new System::UnicodeString (see page 3393) instance, removing leading and trailing spaces and control characters. Use System::UnicodeString::Trim to remove blank space before and after the first printing character. The original System::UnicodeString (see page 3393) instance remains unchanged.

See Also

System::UnicodeString::TrimLeft (see page 3420)

System::UnicodeString::TrimRight (see page 3421)

38.1.23.6.48 UnicodeString::TrimLeft Method

Returns new System::UnicodeString (see page 3393) without leading spaces or control characters.

Pascal

```
function TrimLeft: String;
```

C++

```
__fastcall UnicodeString TrimLeft() const;
```

Description

System::UnicodeString::TrimLeft returns a new System::UnicodeString (see page 3393) instance, removing leading spaces

and control characters. Use System::UnicodeString::TrimLeft to remove blank space before the first printing character. The original System::UnicodeString (see page 3393) instance remains unchanged.

See Also

System::UnicodeString::Trim (see page 3420)

System::UnicodeString::TrimRight (see page 3421)

38.1.23.6.49 UnicodeString::TrimRight Method

Returns new System::UnicodeString (see page 3393) without trailing spaces or control characters.

Pascal

```
function TrimRight: String;
```

C++

```
__fastcall UnicodeString TrimRight() const;
```

Description

System::UnicodeString::TrimRight returns a new System::UnicodeString (see page 3393) instance, removing trailing spaces and control characters. Use the System::UnicodeString::TrimRight function to remove blank space after the last printing character. The original System::UnicodeString (see page 3393) instance remains unchanged.

See Also

System::UnicodeString::Trim (see page 3420)

System::UnicodeString::TrimLeft (see page 3420)

38.1.23.6.50 UnicodeString::Unique Method

Makes System::UnicodeString (see page 3393) instance unique.

Pascal

```
function Unique: UnicodeString&;
```

C++

```
__fastcall UnicodeString& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::UnicodeString::Unique sets the reference count on the string for the System::UnicodeString (see page 3393) object to 1 so that it is the only object referencing that string.

See Also

System::UnicodeString::RefCount (see page 3416)

38.1.23.6.51 UnicodeString::UpperCase Method

Returns upper-case version of System::UnicodeString (see page 3393).

Pascal

```
function UpperCase: String;
```

C++

```
__fastcall UnicodeString UpperCase() const;
```

Description

System::UnicodeString::UpperCase returns a new System::UnicodeString (see page 3393) instance that contains all characters in this string, converted to upper case. The conversion uses the current locale where appropriate. The original System::UnicodeString (see page 3393) instance remains unchanged.

See Also

System::UnicodeString::LowerCase (see page 3415)

38.1.23.6.52 UnicodeString::vprintf Method

Sets value of System::UnicodeString (see page 3393) given format string and its arguments.

Pascal

```
function vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const wchar_t* format, va_list);
```

Description

Use System::UnicodeString::vprintf to set to the value of the System::UnicodeString (see page 3393) instance, given a standard C++ format specifier. Pass values to any arguments in the format specifier as a variable argument list that was created using `va_start`.

This method returns the length in characters of the final formatted string.

See Also

System::UnicodeString::Format (see page 3409)

System::UnicodeString::cat_vprintf (see page 3403)

System::UnicodeString::sprintf (see page 3417)

System::UnicodeString::printf (see page 3416)

arg

start

38.1.23.6.53 UnicodeString::w_str Method

Returns System::UnicodeString::UnicodeString (see page 3393) data (see page 3406).

Pascal

```
function w_str: wchar_t*;
```

C++

```
__fastcall wchar_t* w_str() const;
```

Description

System::UnicodeString::w_str behaves the same as System::UnicodeString::c_str (see page 3402).

See Also

System::UnicodeString::c_str (see page 3402)

38.1.23.7 UnicodeString Operators

The operators of the UnicodeString class are listed here.

Public Operators

	Name	Description
	!= (see page 3423)	Compares two strings.
	[] (see page 3423)	This is the overview for the [] operator overload.
	+ (see page 3424)	Concatenates two System::UnicodeString (see page 3393) values.
	+= (see page 3425)	Append (see page 3624) System::UnicodeString (see page 3393).
	< (see page 3426)	Compares two strings.
	<= (see page 3426)	Compares two strings.
	= (see page 3427)	Assigns string `rhs` to System::UnicodeString (see page 3393).
	== (see page 3427)	Compares System::UnicodeString (see page 3393) with specified string.
	> (see page 3428)	Compares two strings.
	>= (see page 3428)	Compares two strings.

38.1.23.7.1 UnicodeString::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator !=(const UnicodeString& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::!= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	UnicodeString::[] (int) (see page 3424)	Returns character at specified index in string.
	UnicodeString::[] (int) (see page 3424)	Returns character at specified index in string.

38.1.23.7.2.1 UnicodeString::[] Operator (int)

Returns character at specified index in string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall wchar_t operator [](const int idx) const;
__fastcall wchar_t& operator [](const int idx);
```

Description

System::UnicodeString (see page 3393)::[] returns the character in the string at the character index value `idx`. The System::UnicodeString (see page 3393)::[] assumes a base index of 1.

See Also

System::UnicodeString::Substring

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

System::UnicodeString::+= (see page 3425)

38.1.23.7.3 UnicodeString::+ Operator

Concatenates two System::UnicodeString (see page 3393) values.

Pascal

```
operator +(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString operator +(const UnicodeString& rhs) const;
```

Description

The friend function returns a System::UnicodeString (see page 3393) instance that is the concatenation of strings `lhs` and `rhs`.

The member function returns a System::UnicodeString (see page 3393) instance that is the concatenation of this

System::UnicodeString (see page 3393) instance and `rhs`.

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3424)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+= (see page 3425)

38.1.23.7.4 UnicodeString::+= Operator

Append (see page 3624) System::UnicodeString (see page 3393).

Pascal

```
operator +=(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString& operator +=(const UnicodeString& rhs);
```

Description

System::UnicodeString (see page 3393)::+= concatenates this System::UnicodeString (see page 3393) instance and `rhs`, assigning the result to this System::UnicodeString (see page 3393) instance.

System::UnicodeString (see page 3393)::+= is not supported on a property with getter/setter functions.

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3424)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.5 UnicodeString::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator <(const UnicodeString& rhs) const;
```

Description

Returns true if a System::UnicodeString (see page 3393) instance is less than `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::< performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.6 UnicodeString::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator <=(const UnicodeString& rhs) const;
```

Description

Returns true if the System::UnicodeString (see page 3393) instance is less than or equal to `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::<= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.7 UnicodeString::= Operator

Assigns string `rhs` to System::UnicodeString (see page 3393).

Pascal

```
operator =(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString& operator =(const UnicodeString& rhs);
```

Description

Use System::UnicodeString (see page 3393)::= to assign the value of the System::UnicodeString (see page 3393) instance from another instance of System::UnicodeString (see page 3393).

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3424)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

System::UnicodeString::+= (see page 3425)

38.1.23.7.8 UnicodeString::== Operator

Compares System::UnicodeString (see page 3393) with specified string.

Pascal

```
operator ==(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator ==(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3393) instance is the same as `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::== performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.9 UnicodeString::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator >(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3393) instance is greater than `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::> performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::>= (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.7.10 UnicodeString::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator >=(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3393) instance is greater than or equal to `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3393)::>= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3405) method.

See Also

System::UnicodeString::CompareIC (see page 3405)

System::UnicodeString::= (see page 3427)

System::UnicodeString::== (see page 3427)

System::UnicodeString::!= (see page 3423)

System::UnicodeString::> (see page 3428)

System::UnicodeString::< (see page 3426)

System::UnicodeString::<= (see page 3426)

System::UnicodeString::+ (see page 3424)

38.1.23.8 UnicodeString Friends

The friends of the UnicodeString class are listed here.

Public Friends

Name	Description
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) (see page 3429)	This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) (see page 3430)	This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

38.1.23.8.1 friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) Friend

Pascal

```
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs)
```

C++

```
friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs);
```

Description

This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).

38.1.23.8.2 friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) Friend

Pascal

```
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs)
```

C++

```
friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs);
```

Description

This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

38.1.24 Variant Class

System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.

Class Hierarchy

Pascal

```
Variant = class(TVarData);
```

C++

```
class Variant : public TVarData;
```

File

sysvari.h

Description

The System::Variant type is capable of representing values that change type dynamically. Whereas a variable of any other type is statically bound to that type, a variable of the System::Variant type can assume values of differing types at run-time. The System::Variant type is most commonly used in situations where the actual type to be operated upon varies or is unknown at compile-time. While variants offer great flexibility, they also consume more memory than regular variables, and operations on variants are substantially slower than operations on statically typed values.

System::Variants (see page 4997) can contain various numeric, string, Boolean, and date-time values. If COM is available (Windows only) variants can also contain OLE Automation objects. Finally, variants can contain arrays of any of the above types. The special System::Variant value Unassigned is used to indicate that a variant has not yet been assigned a value, and the special variant value Null is used to indicate unknown or missing data.

A variant can be constructed from the following data types:

short (see page 3472)

int (see page 3471)

float (see page 3470)

double (see page 3469)

Currency (see page 3469)

TDateTime (see page 3473)

bool (see page 3468)

WordBool

Byte

AnsiString (see page 3468)&

char (see page 3468) *

wchar_t * const

Ole2::IDispatch (see page 3498)* const (Windows only)

Ole2::IUnknown (see page 3828)* const (Windows only)

Note: When a System::Variant contains an OLE Automation object, the variant can be used to get and set properties of the object, and to invoke methods on the object.

System::Variant variables are always initialized to be Unassigned when they are first created. This is true whether a variant variable is global, local, or part of a structure such as an array.

In C++ code, variant arrays are usually created using a System::Variant constructor in conjunction with the OPENARRAY macro:

```
Variant V(OPENARRAY(int,(0,HighVal,0,HighVal)),varInteger);
```

You can also use the VarArrayCreate function, but that function is designed for use in Delphi code, and makes for complex C++ code.

Note: Use only the operators declared within System::Variant. The compiler ignores any operators you overload yourself.

Note: For binary operators, if one operand is of type System::Variant, the other operand is automatically converted to type System::Variant.

See Also

Autocmd

TVarRec (see page 3819)

TVarData (see page 3813)

Openarray

38.1.24.1 Variant Members

The following tables list the members exposed by Variant.

Public Methods

	Name	Description
	~Variant (see page 3433)	Frees a System::Variant::Variant (see page 3430) object.
	ArrayDimCount (see page 3437)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3438)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3438)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3438)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3439)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3439)	Unlocks a variant array that was previously locked by ArrayLock (see page 3438).

	AsType (see page 3439)	Casts a System::Variant::Variant (see page 3430) object to the type specified by the argument VarType.
	ChangeType (see page 3439)	Changes the data type of a System::Variant::Variant (see page 3430) object once it has been instantiated.
	Clear (see page 3440)	System::Variant::Clears the value of the System::Variant::Variant (see page 3430).
	Empty (see page 3440)	This is Empty, a member of class Variant.
	GetBaseVariant (see page 3440)	Returns the Variant (see page 3430) that is assigned as the value of this Variant (see page 3430) when Type (see page 3444) is varVariant (see page 3941).
	GetElement (see page 3441)	This is the overview for the GetElement method overload.
	GetError (see page 3441)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and returns the integer value.
	IsArray (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varArray (see page 3929) bit.
	IsEmpty (see page 3442)	Queries the VType of the base class TVarData (see page 3813).
	IsNull (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varNull (see page 3936) bit.
	NoParam (see page 3443)	This is NoParam, a member of class Variant.
	PutElement (see page 3443)	This is the overview for the PutElement method overload.
	Sanitized (see page 3444)	This is Sanitized, a member of class Variant.
	SetError (see page 3444)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and assigns the integer err as the value of the varError (see page 3934).
	Type (see page 3444)	This is Type, a member of class Variant.
	Variant (see page 3434)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 3446)	This is the overview for the - operator overload.
	! (see page 3447)	Returns a new System::Variant::Variant (see page 3430) that is the logical negation of this System::Variant::Variant (see page 3430).
	!= (see page 3448)	This is the overview for the != operator overload.
	% (see page 3448)	This is the overview for the % operator overload.
	%= (see page 3449)	Returns this System::Variant::Variant (see page 3430) as the remainder from dividing its original value and the System::Variant::Variant (see page 3430) specified by rhs.
	& (see page 3449)	This is the overview for the & operator overload.
	&= (see page 3450)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise and operation.
	* (see page 3450)	This is the overview for the * operator overload.
	*= (see page 3451)	Multiplies this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) specified by rhs.
	/ (see page 3452)	This is the overview for the / operator overload.
	/= (see page 3452)	Divides this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) rhs.
	^ (see page 3453)	This is the overview for the ^ operator overload.
	^= (see page 3453)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise xor operation.
	__int64 (see page 3454)	This is __int64, a member of class Variant.
	| (see page 3454)	This is the overview for the | operator overload.
	|= (see page 3454)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise or operation.
	+ (see page 3455)	This is the overview for the + operator overload.
	+= (see page 3455)	Adds the System::Variant::Variant (see page 3430) specified by rhs to this System::Variant::Variant (see page 3430) object.
	< (see page 3456)	This is the overview for the < operator overload.
	<< (see page 3456)	This is the overview for the << operator overload.
	<<= (see page 3458)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3430).

	<= (see page 3458)	This is the overview for the <= operator overload.
	-= (see page 3463)	Subtracts the System::Variant::Variant (see page 3430) specified by rhs from this System::Variant::Variant (see page 3430) object.
	= (see page 3459)	This is the overview for the = operator overload.
	== (see page 3463)	This is the overview for the == operator overload.
	> (see page 3464)	This is the overview for the > operator overload.
	>= (see page 3465)	This is the overview for the >= operator overload.
	>> (see page 3467)	This is the overview for the >> operator overload.
	>>= (see page 3468)	This is >>=, a member of class Variant.
	AnsiString (see page 3468)	Converts the System::Variant::Variant (see page 3430) to an AnsiString object.
	bool (see page 3468)	Converts the System::Variant::Variant (see page 3430) to a boolean value.
	char (see page 3468)	This is char, a member of class Variant.
	Currency (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a Currency object.
	Currency* (see page 3469)	This is Currency*, a member of class Variant.
	double (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a double value.
	double* (see page 3470)	This is double*, a member of class Variant.
	float (see page 3470)	Converts the System::Variant::Variant (see page 3430) to a float value.
	float* (see page 3470)	This is float*, a member of class Variant.
	IDispatch* (see page 3470)	This is IDispatch*, a member of class Variant.
	IInterface* (see page 3471)	This is IInterface*, a member of class Variant.
	int (see page 3471)	Converts the System::Variant::Variant (see page 3430) to an integer value.
	int* (see page 3471)	This is int*, a member of class Variant.
	IUnknown* (see page 3471)	This is IUnknown*, a member of class Variant.
	long (see page 3472)	This is long, a member of class Variant.
	long* (see page 3472)	This is long*, a member of class Variant.
	short (see page 3472)	Converts the System::Variant::Variant (see page 3430) to a short value.
	short* (see page 3472)	This is short*, a member of class Variant.
	signed char (see page 3473)	This is signed char, a member of class Variant.
	signed char* (see page 3473)	This is signed char*, a member of class Variant.
	TDateTime (see page 3473)	Converts the System::Variant::Variant (see page 3430) to a TDateTime object.
	TDateTime* (see page 3473)	This is TDateTime*, a member of class Variant.
	UnicodeString (see page 3474)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3474)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 3474)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 3474)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3474)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3475)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3475)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3475)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 3475)	This is unsigned short*, a member of class Variant.
	WideString (see page 3476)	This is WideString, a member of class Variant.

38.1.24.2 Variant::~Variant Destructor

Frees a System::Variant::Variant (see page 3430) object.

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~Variant();
```

Description

System::Variant::Destroy is the destructor for System::Variant::Variant (see page 3430).

38.1.24.3 Variant Constructor

This is the overview for the Variant constructor overload.

Overload List

	Name	Description
	Variant::Variant () (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (__int64) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (__int64*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (AnsiString&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (bool) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (char) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (char*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (Currency&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (Currency*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (double) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (double*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (float) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (float*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (int) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (int*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (int*, int, Word) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (long) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (long double) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (long*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (short) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (short*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (signed char) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (signed char*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (System::DelphiInterface<IDispatch>&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (TDateTime&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (TDateTime*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (TVarData&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (UnicodeString&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned char) (see page 3435)	Creates an instance of System::Variant::Create.

	Variant::Variant (unsigned char*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned int) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned int*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned long) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned long*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned short) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned short*) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (Variant&) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (Variant*, int) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (wchar_t*, bool) (see page 3435)	Creates an instance of System::Variant::Create.
	Variant::Variant (WideString&) (see page 3435)	Creates an instance of System::Variant::Create.

38.1.24.3.1 Variant::Variant Constructor ()

Creates an instance of System::Variant::Create.

Pascal

```
constructor Create;
constructor Create(const src: Variant&);
constructor Create(const src: TVarData&);
constructor Create(const src: bool);
constructor Create(const src: char);
constructor Create(const src: signed char);
constructor Create(const src: unsigned char);
constructor Create(const src: short);
constructor Create(const src: unsigned short);
constructor Create(const src: Integer);
constructor Create(const src: unsigned int);
constructor Create(const src: long);
constructor Create(const src: unsigned long);
constructor Create(const src: float);
constructor Create(const src: double);
constructor Create(const src: long double);
constructor Create(const src: __int64);
constructor Create(const src: AnsiString&);
constructor Create(const src: Currency&);
constructor Create(const src: TDateTime&);
constructor Create(const src: WideString&);
constructor Create(const src: UnicodeString&);
constructor Create(const src: char*);
constructor Create(var src: wchar_t; makeCopy: bool = true);
constructor Create(const src: System::DelphiInterface<IDispatch>&);
constructor Create(var src: signed char);
constructor Create(var src: unsigned char);
constructor Create(var src: short);
constructor Create(var src: unsigned short);
constructor Create(var src: Integer);
constructor Create(var src: unsigned int);
constructor Create(var src: long);
```

```
constructor Create(var src: unsigned long);
constructor Create(var src: float);
constructor Create(var src: double);
constructor Create(var src: __int64);
constructor Create(var src: Currency);
constructor Create(var src: TDateTime);
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
constructor Create(const values: Variant*; const valuesSize: Integer);
```

C++

```
__fastcall Variant();
__fastcall Variant(const Variant& src);
__fastcall Variant(const TVarData& src);
__fastcall Variant(const bool src);
__fastcall Variant(const char src);
__fastcall Variant(const signed char src);
__fastcall Variant(const unsigned char src);
__fastcall Variant(const short src);
__fastcall Variant(const unsigned short src);
__fastcall Variant(const int src);
__fastcall Variant(const unsigned int src);
__fastcall Variant(const long src);
__fastcall Variant(const unsigned long src);
__fastcall Variant(const float src);
__fastcall Variant(const double src);
__fastcall Variant(const long double src);
__fastcall Variant(const __int64 src);
__fastcall Variant(const AnsiString& src);
__fastcall Variant(const Currency& src);
__fastcall Variant(const TDateTime& src);
__fastcall Variant(const WideString& src);
__fastcall Variant(const UnicodeString& src);
__fastcall Variant(const char* src);
__fastcall Variant(wchar_t* src, bool makeCopy = true);
__fastcall Variant(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant(signed char* src);
__fastcall Variant(unsigned char* src);
__fastcall Variant(short* src);
__fastcall Variant(unsigned short* src);
__fastcall Variant(int* src);
__fastcall Variant(unsigned int* src);
__fastcall Variant(long* src);
__fastcall Variant(unsigned long* src);
__fastcall Variant(float* src);
__fastcall Variant(double* src);
__fastcall Variant(__int64* src);
__fastcall Variant(Currency* src);
__fastcall Variant(TDateTime* src);
__fastcall Variant(const int* bounds, const int boundsSize, Word varType);
__fastcall Variant(const Variant* values, const int valuesSize);
```

Description

The first System::Variant::Create constructor above creates a variant instance with the value Unassigned.

The second System::Variant::Create constructor creates a System::Variant::Create array with the bounds given by bounds and the element type given by varType. The bounds parameter must contain an even number of values, and each pair of values must specify the upper and lower bounds of one dimension of the array. The boundsSize parameter is the index of the last element in bounds. The element type of the array, given by the varType parameter, must be one of the TVarData (see page 3813) type codes, and cannot include the varArray (see page 3929) or varByRef (see page 3930) bits. The element type cannot be varString (see page 3938). To create an array of strings, use the varOleStr (see page 3936) type code. If the element type is varSystem::Variant::Create, the elements of the array are themselves System::Variant::Creates and can, in turn, contain System::Variant::Create arrays.

The third System::Variant::Create constructor creates a one-dimensional System::Variant::Create array with the elements given

by the values parameter. The low bound of the returned array is zero, the high bound is given by valuesSize (the index of the last value). The element type of the instantiated array is System::Variant::Create.

The remaining System::Variant::Create constructors create a System::Variant::Create with the same value as the src argument, converting src as appropriate. Constructors that declare src as a const parameter copy src into the variant's internal representation. Constructors that do not declare src as a const create a reference to src and set the variant's varByRef (see page 3930) bit.

See Also

Unassigned (see page 5076)

TVarData (see page 3813)

38.1.24.4 Variant Methods

The methods of the Variant class are listed here.

Public Methods

	Name	Description
	ArrayDimCount (see page 3437)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3438)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3438)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3438)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3439)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3439)	Unlocks a variant array that was previously locked by ArrayLock (see page 3438).
	AsType (see page 3439)	Casts a System::Variant::Variant (see page 3430) object to the type specified by the argument VarType.
	ChangeType (see page 3439)	Changes the data type of a System::Variant::Variant (see page 3430) object once it has been instantiated.
	Clear (see page 3440)	System::Variant::Clears the value of the System::Variant::Variant (see page 3430).
	Empty (see page 3440)	This is Empty, a member of class Variant.
	GetBaseVariant (see page 3440)	Returns the Variant (see page 3430) that is assigned as the value of this Variant (see page 3430) when Type (see page 3444) is varVariant (see page 3941).
	GetElement (see page 3441)	This is the overview for the GetElement method overload.
	GetError (see page 3441)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and returns the integer value.
	IsArray (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varArray (see page 3929) bit.
	IsEmpty (see page 3442)	Queries the VType of the base class TVarData (see page 3813).
	IsNull (see page 3442)	Queries the VType of the base class TVarData (see page 3813) for the varNull (see page 3936) bit.
	NoParam (see page 3443)	This is NoParam, a member of class Variant.
	PutElement (see page 3443)	This is the overview for the PutElement method overload.
	Sanitized (see page 3444)	This is Sanitized, a member of class Variant.
	SetError (see page 3444)	Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and assigns the integer err as the value of the varError (see page 3934).
	Type (see page 3444)	This is Type, a member of class Variant.

38.1.24.4.1 Variant::ArrayDimCount Method

Returns the number of dimensions of the given variant.

Pascal

```
function ArrayDimCount: Integer;
```

C++

```
__fastcall int ArrayDimCount() const;
```

Description

The resulting value is zero if the given variant is not a System::Variant::Variant (see page 3430) array.

38.1.24.4.2 Variant::ArrayHighBound Method

Returns the high bound of the given dimension in the given variant array.

Pascal

```
function ArrayHighBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayHighBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

38.1.24.4.3 Variant::ArrayLock Method

Locks the given variant array and returns a pointer to the data stored in the array.

Pascal

```
function ArrayLock: Pointer;
```

C++

```
__fastcall Pointer ArrayLock();
```

Description

While locked, a variant array can not be resized and any calls to ArrayRedim (see page 3439) will fail. Once locked, a variant array must later be unlocked using ArrayUnlock (see page 3439). An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array.

The pointer returned by System::Variant::ArrayLock points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, System::Variant::ArrayLock can be used to gain direct access to the array data which results in much improved performance.

38.1.24.4.4 Variant::ArrayLowBound Method

Returns the low bound of the given dimension in the given variant array.

Pascal

```
function ArrayLowBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayLowBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

38.1.24.4.5 Variant::ArrayRedim Method

Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.

Pascal

```
procedure ArrayRedim(highBound: Integer);
```

C++

```
__fastcall void ArrayRedim(int highBound);
```

Description

Existing elements of the array are preserved, and new elements are set to zero or empty. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if there is not enough memory to resize the array.

38.1.24.4.6 Variant::ArrayUnlock Method

Unlocks a variant array that was previously locked by ArrayLock (see page 3438).

Pascal

```
procedure ArrayUnlock;
```

C++

```
__fastcall void ArrayUnlock();
```

Description

System::Variant::ArrayUnlock invalidates the pointer returned by a call to the ArrayLock (see page 3438) method. An ESystem::Variant::VariantError exception is raised if the System::Variant::Variant (see page 3430) is not an array.

38.1.24.4.7 Variant::AsType Method

Casts a System::Variant::Variant (see page 3430) object to the type specified by the argument VarType.

Pascal

```
function AsType(VarType: Integer): Variant;
```

C++

```
__fastcall Variant AsType(int VarType) const;
```

Description

System::Variant::AsType returns a new System::Variant::Variant (see page 3430) with the same value as this System::Variant::Variant (see page 3430) but with the type specified by VarType. After the call to System::Variant::AsType there are two System::Variant::Variant (see page 3430) objects of the same type.

38.1.24.4.8 Variant::ChangeType Method

Changes the data type of a System::Variant::Variant (see page 3430) object once it has been instantiated.

Pascal

```
function ChangeType(VarType: Integer): Variant&;
```

C++

```
__fastcall Variant& ChangeType(int VarType);
```

Description

System::Variant::ChangeType converts the System::Variant::Variant (see page 3430) to the type specified by VarType.

See Also

System::Variant::Type2

38.1.24.4.9 Variant::Clear Method

System::Variant::Clears the value of the System::Variant::Variant (see page 3430).

Pascal

```
procedure Clear;
```

C++

```
__fastcall void Clear();
```

Description

System::Variant::Clear assigns the value Unassigned to this System::Variant::Variant (see page 3430).

See Also

Unassigned (see page 5076)

38.1.24.4.10 Variant::Empty Method

Pascal

```
function Empty: Variant&; static;
```

C++

```
static Variant& Empty();
```

Description

This is Empty, a member of class Variant.

38.1.24.4.11 Variant::GetBaseVariant Method

Returns the Variant (see page 3430) that is assigned as the value of this Variant (see page 3430) when Type (see page 3444) is varVariant (see page 3941).

Pascal

```
function GetBaseVariant: Variant*;
```

C++

```
__fastcall Variant* GetBaseVariant();
```

Description

Call GetBaseVariant to remove a level of indirection by accessing a Variant (see page 3430) that has been assigned as the value of this Variant (see page 3430). Check the Type (see page 3444) method to ensure that the variant is of type var

Variant (see page 3430) before calling GetBaseVariant.

See Also

System::Variant (see page 3430)

38.1.24.4.12 GetElement Method

This is the overview for the GetElement method overload.

Overload List

	Name	Description
	Variant::GetElement (int) (see page 3441)	Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.
	Variant::GetElement (int, int) (see page 3441)	Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.
	Variant::GetElement (int, int, int) (see page 3441)	Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.
	Variant::GetElement (int, int, int, int) (see page 3441)	Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.
	Variant::GetElement (int, int, int, int, int) (see page 3441)	Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.

38.1.24.4.12.1 Variant::GetElement Method (int)

Returns the value of a specified element in a System::Variant::Variant (see page 3430) array.

Pascal

```
function GetElement(const i1: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
Integer; const i5: Integer): Variant;
```

C++

```
__fastcall Variant GetElement(const int i1) const;
__fastcall Variant GetElement(const int i1, const int i2) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4, const
int i5) const;
```

Description

Use System::Variant::GetElement to access one-dimensional or multi-dimensional System::Variant::Variant (see page 3430) arrays of up to 5 dimensions. The parameters specify the indexes of the desired element.

See Also

System::Variant::PutElement (see page 3443)

38.1.24.4.13 Variant::GetError Method

Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and returns the integer value.

Pascal

```
function GetError: HResult;
```

C++

```
__fastcall HResult GetError() const;
```

Description

See the base class TVarData (see page 3813) for more information about varError (see page 3934).

38.1.24.4.14 Variant::IsArray Method

Queries the VType of the base class TVarData (see page 3813) for the varArray (see page 3929) bit.

Pascal

```
function IsArray: bool;
```

C++

```
__fastcall bool IsArray() const;
```

Description

System::Variant::IsArray returns true if the given variant is an array, false otherwise.

See Also

TVarData (see page 3813)

38.1.24.4.15 Variant::IsEmpty Method

Queries the VType of the base class TVarData (see page 3813).

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::Variant::IsEmpty returns true if the System::Variant::Variant (see page 3430) type code is varEmpty (see page 3933), false otherwise.

See Also

TVarData (see page 3813)

38.1.24.4.16 Variant::IsNull Method

Queries the VType of the base class TVarData (see page 3813) for the varNull (see page 3936) bit.

Pascal

```
function IsNull: bool;
```

C++

```
__fastcall bool IsNull() const;
```

Description

System::Variant::IsNull returns true if the System::Variant::Variant (see page 3430) type code is varNull (see page 3936), false otherwise.

See Also

TVarData (see page 3813)

38.1.24.4.17 Variant::NoParam Method

Pascal

```
function NoParam: Variant&; static;
```

C++

```
static Variant& NoParam();
```

Description

This is NoParam, a member of class Variant.

38.1.24.4.18 PutElement Method

This is the overview for the PutElement method overload.

Overload List

	Name	Description
	Variant::PutElement (Variant&, int) (see page 3443)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.
	Variant::PutElement (Variant&, int, int) (see page 3443)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.
	Variant::PutElement (Variant&, int, int, int) (see page 3443)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.
	Variant::PutElement (Variant&, int, int, int, int) (see page 3443)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.
	Variant::PutElement (Variant&, int, int, int, int, int) (see page 3443)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.

38.1.24.4.18.1 Variant::PutElement Method (Variant&, int)

Assigns an element as the value of an entry in a System::Variant::Variant (see page 3430) array.

Pascal

```
procedure PutElement(const data: Variant&; const i1: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer; const i5: Integer);
```

C++

```
__fastcall void PutElement(const Variant& data, const int i1);
```

```
__fastcall void PutElement(const Variant& data, const int i1, const int i2);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4, const int i5);
```

Description

Use System::Variant::PutElement to access one-dimensional and multi-dimensional System::Variant::Variant (see page 3430) arrays of up to 5 dimensions. The data parameter is the value to assign. The remaining parameters provide indexes into the dimensions of the array.

See Also

System::Variant::GetElement (see page 3441)

38.1.24.4.19 Variant::Sanitized Method

Pascal

```
function Sanitized: Variant;
```

C++

```
__fastcall Variant Sanitized() const;
```

Description

This is Sanitized, a member of class Variant.

38.1.24.4.20 Variant::SetError Method

Converts the System::Variant::Variant (see page 3430) to a varError (see page 3934) type and assigns the integer err as the value of the varError (see page 3934).

Pascal

```
procedure SetError(const err: HResult);
```

C++

```
__fastcall void SetError(const HResult err);
```

Description

See the base class TVarData (see page 3813) for more information about varError (see page 3934).

See Also

TVarData (see page 3813)

38.1.24.4.21 Variant::Type Method

Pascal

```
function Type: Integer;
```

C++

```
__fastcall int Type() const;
```

Description

This is Type, a member of class Variant.

38.1.24.5 Variant Operators

The operators of the Variant class are listed here.

Public Operators

	Name	Description
	- (see page 3446)	This is the overview for the - operator overload.
	! (see page 3447)	Returns a new System::Variant::Variant (see page 3430) that is the logical negation of this System::Variant::Variant (see page 3430).
	!= (see page 3448)	This is the overview for the != operator overload.
	% (see page 3448)	This is the overview for the % operator overload.
	%= (see page 3449)	Returns this System::Variant::Variant (see page 3430) as the remainder from dividing its original value and the System::Variant::Variant (see page 3430) specified by rhs.
	& (see page 3449)	This is the overview for the & operator overload.
	&= (see page 3450)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise and operation.
	* (see page 3450)	This is the overview for the * operator overload.
	*= (see page 3451)	Multiplies this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) specified by rhs.
	/ (see page 3452)	This is the overview for the / operator overload.
	/= (see page 3452)	Divides this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) rhs.
	^ (see page 3453)	This is the overview for the ^ operator overload.
	^= (see page 3453)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise xor operation.
	__int64 (see page 3454)	This is __int64, a member of class Variant.
	| (see page 3454)	This is the overview for the | operator overload.
	|= (see page 3454)	Returns this System::Variant::Variant (see page 3430) after performing a bitwise or operation.
	+ (see page 3455)	This is the overview for the + operator overload.
	+= (see page 3455)	Adds the System::Variant::Variant (see page 3430) specified by rhs to this System::Variant::Variant (see page 3430) object.
	< (see page 3456)	This is the overview for the < operator overload.
	<< (see page 3456)	This is the overview for the << operator overload.
	<<= (see page 3458)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3430).
	<= (see page 3458)	This is the overview for the <= operator overload.
	= (see page 3459)	This is the overview for the = operator overload.
	-= (see page 3463)	Subtracts the System::Variant::Variant (see page 3430) specified by rhs from this System::Variant::Variant (see page 3430) object.
	== (see page 3463)	This is the overview for the == operator overload.
	> (see page 3464)	This is the overview for the > operator overload.
	>= (see page 3465)	This is the overview for the >= operator overload.
	>> (see page 3467)	This is the overview for the >> operator overload.
	>>= (see page 3468)	This is >>=, a member of class Variant.
	AnsiString (see page 3468)	Converts the System::Variant::Variant (see page 3430) to an AnsiString object.
	bool (see page 3468)	Converts the System::Variant::Variant (see page 3430) to a boolean value.
	char (see page 3468)	This is char, a member of class Variant.
	Currency (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a Currency object.
	Currency* (see page 3469)	This is Currency*, a member of class Variant.
	double (see page 3469)	Converts the System::Variant::Variant (see page 3430) to a double value.
	double* (see page 3470)	This is double*, a member of class Variant.
	float (see page 3470)	Converts the System::Variant::Variant (see page 3430) to a float value.
	float* (see page 3470)	This is float*, a member of class Variant.
	IDispatch* (see page 3470)	This is IDispatch*, a member of class Variant.

	IInterface* (see page 3471)	This is IInterface*, a member of class Variant.
	int (see page 3471)	Converts the System::Variant::Variant (see page 3430) to an integer value.
	int* (see page 3471)	This is int*, a member of class Variant.
	IUnknown* (see page 3471)	This is IUnknown*, a member of class Variant.
	long (see page 3472)	This is long, a member of class Variant.
	long* (see page 3472)	This is long*, a member of class Variant.
	short (see page 3472)	Converts the System::Variant::Variant (see page 3430) to a short value.
	short* (see page 3472)	This is short*, a member of class Variant.
	signed char (see page 3473)	This is signed char, a member of class Variant.
	signed char* (see page 3473)	This is signed char*, a member of class Variant.
	TDateTime (see page 3473)	Converts the System::Variant::Variant (see page 3430) to a TDateTime object.
	TDateTime* (see page 3473)	This is TDateTime*, a member of class Variant.
	UnicodeString (see page 3474)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3474)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 3474)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 3474)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3474)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3475)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3475)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3475)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 3475)	This is unsigned short*, a member of class Variant.
	WideString (see page 3476)	This is WideString, a member of class Variant.

38.1.24.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Variant::- (AnsiString&) (see page 3446)	This is -, a member of class Variant.
	Variant::- () (see page 3446)	This is -, a member of class Variant.
	Variant::- (double) (see page 3447)	This is -, a member of class Variant.
	Variant::- (int) (see page 3447)	This is -, a member of class Variant.
	Variant::- (Variant&) (see page 3447)	This is -, a member of class Variant.

38.1.24.5.1.1 Variant::- Operator ()

Pascal

```
operator -;
```

C++

```
__fastcall Variant operator -() const;
```

Description

This is -, a member of class Variant.

38.1.24.5.1.2 Variant::- Operator (AnsiString&)

Pascal

```
operator -(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator -(const AnsiString& rhs) const;
```

Description

This is -, a member of class Variant.

38.1.24.5.1.3 Variant::- Operator (Variant&)

Pascal

```
operator -(const rhs: Variant&);
```

C++

```
__fastcall Variant operator -(const Variant& rhs) const;
```

Description

This is -, a member of class Variant.

38.1.24.5.1.4 Variant::- Operator (double)

Pascal

```
operator -(rhs: double);
```

C++

```
__fastcall Variant operator -(double rhs) const;
```

Description

This is -, a member of class Variant.

38.1.24.5.1.5 Variant::- Operator (int)

Pascal

```
operator -(rhs: Integer);
```

C++

```
__fastcall Variant operator -(int rhs) const;
```

Description

This is -, a member of class Variant.

38.1.24.5.2 Variant::! Operator

Returns a new System::Variant::Variant (see page 3430) that is the logical negation of this System::Variant::Variant (see page 3430).

Pascal

```
operator !;
```

C++

```
__fastcall Variant operator !() const;
```

Description

The System::Variant (see page 3430)::! returns 0 if the System::Variant::Variant (see page 3430) has a nonzero value and 1 if the value of the System::Variant::Variant (see page 3430) is 0 or NULL.

38.1.24.5.3 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Variant::!= (char*) (see page 3448)	Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.
	Variant::!= (double) (see page 3448)	Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.
	Variant::!= (int) (see page 3448)	Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.
	Variant::!= (TDateTime) (see page 3448)	Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.
	Variant::!= (Variant&) (see page 3448)	Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.

38.1.24.5.3.1 Variant::!= Operator (TDateTime)

Compares the System::Variant::Variant (see page 3430) with another value, testing for inequality.

Pascal

```
operator !=(const rhs: Variant&);
operator !=(const rhs: char*);
operator !=(rhs: Integer);
operator !=(rhs: double);
operator !=(rhs: TDateTime);
```

C++

```
__fastcall bool operator !=(const Variant& rhs) const;
__fastcall bool operator !=(const char* rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
__fastcall bool operator !=(TDateTime rhs) const;
```

Description

The != System::Variant (see page 3430)::!= returns true if the System::Variant::Variant (see page 3430)'s value is not the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.4 % Operator

This is the overview for the % operator overload.

Overload List

	Name	Description
	Variant::% (AnsiString&) (see page 3449)	Returns the remainder from dividing the System::Variant::Variant (see page 3430)'s value by a specified value.

	Variant::% (double) (see page 3449)	Returns the remainder from dividing the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::% (int) (see page 3449)	Returns the remainder from dividing the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::% (Variant&) (see page 3449)	Returns the remainder from dividing the System::Variant::Variant (see page 3430)'s value by a specified value.

38.1.24.5.4.1 Variant::% Operator (AnsiString&)

Returns the remainder from dividing the System::Variant::Variant (see page 3430)'s value by a specified value.

Pascal

```
operator %(const rhs: Variant&);
operator %(const rhs: AnsiString&);
operator %(rhs: Integer);
operator %(rhs: double);
```

C++

```
__fastcall Variant operator %(const Variant& rhs) const;
__fastcall Variant operator %(const AnsiString& rhs) const;
__fastcall Variant operator %(int rhs) const;
__fastcall Variant operator %(double rhs) const;
```

Description

The System::Variant (see page 3430)::% returns a new System::Variant::Variant (see page 3430) that is the modulus of this System::Variant::Variant (see page 3430) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.5 Variant::%= Operator

Returns this System::Variant::Variant (see page 3430) as the remainder from dividing its original value and the System::Variant::Variant (see page 3430) specified by rhs.

Pascal

```
operator %=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator %=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.6 & Operator

This is the overview for the & operator overload.

Overload List

	Name	Description
	Variant::& (AnsiString&) (see page 3450)	Performs a bitwise and operation with a specified value.
	Variant::& (double) (see page 3450)	Performs a bitwise and operation with a specified value.
	Variant::& (int) (see page 3450)	Performs a bitwise and operation with a specified value.
	Variant::& (Variant&) (see page 3450)	Performs a bitwise and operation with a specified value.

38.1.24.5.6.1 Variant::& Operator (AnsiString&)

Performs a bitwise and operation with a specified value.

Pascal

```
operator &(const rhs: Variant&);
operator &(const rhs: AnsiString&);
operator &(rhs: Integer);
operator &(rhs: double);
```

C++

```
__fastcall Variant operator &(const Variant& rhs) const;
__fastcall Variant operator &(const AnsiString& rhs) const;
__fastcall Variant operator &(int rhs) const;
__fastcall Variant operator &(double rhs) const;
```

Description

The System::Variant (see page 3430)::& returns a new System::Variant::Variant (see page 3430) that is the result of a bitwise and operation on this System::Variant::Variant (see page 3430) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.7 Variant::&= Operator

Returns this System::Variant::Variant (see page 3430) after performing a bitwise and operation.

Pascal

```
operator &=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator &=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.8 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Variant::* (AnsiString&) (see page 3451)	Multiplies the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::* (double) (see page 3451)	Multiplies the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::* (int) (see page 3451)	Multiplies the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::* (Variant&) (see page 3451)	Multiplies the System::Variant::Variant (see page 3430)'s value by a specified value.

38.1.24.5.8.1 Variant::* Operator (AnsiString&)

Multiplies the System::Variant::Variant (see page 3430)'s value by a specified value.

Pascal

```
operator *(const rhs: Variant&);
operator *(const rhs: AnsiString&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Variant operator *(const Variant& rhs) const;
__fastcall Variant operator *(const AnsiString& rhs) const;
__fastcall Variant operator *(int rhs) const;
__fastcall Variant operator *(double rhs) const;
```

Description

The System::Variant (see page 3430)::* returns a new System::Variant::Variant (see page 3430) that is the result of multiplying this System::Variant::Variant (see page 3430) with rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.9 Variant::*= Operator

Multiplies this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) specified by rhs.

Pascal

```
operator *=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator *=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.10 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Variant::/ (AnsiString&) (see page 3452)	Divides the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::/ (double) (see page 3452)	Divides the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::/ (int) (see page 3452)	Divides the System::Variant::Variant (see page 3430)'s value by a specified value.
	Variant::/ (Variant&) (see page 3452)	Divides the System::Variant::Variant (see page 3430)'s value by a specified value.

38.1.24.5.10.1 Variant::/ Operator (AnsiString&)

Divides the System::Variant::Variant (see page 3430)'s value by a specified value.

Pascal

```
operator /(const rhs: Variant&);
operator /(const rhs: AnsiString&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Variant operator /(const Variant& rhs) const;
__fastcall Variant operator /(const AnsiString& rhs) const;
__fastcall Variant operator /(int rhs) const;
__fastcall Variant operator /(double rhs) const;
```

Description

The System::Variant (see page 3430)::/ returns a new System::Variant::Variant (see page 3430) that is the result of dividing this System::Variant::Variant (see page 3430) by rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.11 Variant::/= Operator

Divides this System::Variant::Variant (see page 3430) by the System::Variant::Variant (see page 3430) rhs.

Pascal

```
operator /=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator /=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.12 ^ Operator

This is the overview for the ^ operator overload.

Overload List

	Name	Description
	Variant::^ (AnsiString&) (see page 3453)	Performs a bitwise xor operation with a specified value.
	Variant::^ (double) (see page 3453)	Performs a bitwise xor operation with a specified value.
	Variant::^ (int) (see page 3453)	Performs a bitwise xor operation with a specified value.
	Variant::^ (Variant&) (see page 3453)	Performs a bitwise xor operation with a specified value.

38.1.24.5.12.1 Variant::^ Operator (AnsiString&)

Performs a bitwise xor operation with a specified value.

Pascal

```
operator ^(const rhs: Variant&);
operator ^(const rhs: AnsiString&);
operator ^(rhs: Integer);
operator ^(rhs: double);
```

C++

```
__fastcall Variant operator ^(const Variant& rhs) const;
__fastcall Variant operator ^(const AnsiString& rhs) const;
__fastcall Variant operator ^(int rhs) const;
__fastcall Variant operator ^(double rhs) const;
```

Description

The ^ System::Variant (see page 3430)::^ returns a new System::Variant::Variant (see page 3430) that is the result of a bitwise xor operation on this System::Variant::Variant (see page 3430) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.13 Variant::^= Operator

Returns this System::Variant::Variant (see page 3430) after performing a bitwise xor operation.

Pascal

```
operator ^=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator ^=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.14 Variant::__int64 Operator

Pascal

```
operator __int64;
```

C++

```
__fastcall operator __int64() const;
```

Description

This is __int64, a member of class Variant.

38.1.24.5.15 | Operator

This is the overview for the | operator overload.

Overload List

	Name	Description
	Variant::| (AnsiString&) (see page 3454)	Performs a bitwise or operation with a specified value.
	Variant::| (double) (see page 3454)	Performs a bitwise or operation with a specified value.
	Variant::| (int) (see page 3454)	Performs a bitwise or operation with a specified value.
	Variant::| (Variant&) (see page 3454)	Performs a bitwise or operation with a specified value.

38.1.24.5.15.1 Variant::| Operator (AnsiString&)

Performs a bitwise or operation with a specified value.

Pascal

```
operator |(const rhs: Variant&);
operator |(const rhs: AnsiString&);
operator |(rhs: Integer);
operator |(rhs: double);
```

C++

```
__fastcall Variant operator |(const Variant& rhs) const;
__fastcall Variant operator |(const AnsiString& rhs) const;
__fastcall Variant operator |(int rhs) const;
__fastcall Variant operator |(double rhs) const;
```

Description

The | System::Variant (see page 3430)::| returns a new System::Variant::Variant (see page 3430) that is the result of a bitwise or operation on this System::Variant::Variant (see page 3430) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.16 Variant::|= Operator

Returns this System::Variant::Variant (see page 3430) after performing a bitwise or operation.

Pascal

```
operator |=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator |=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.17 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Variant::+ (double) (see page 3455)	Adds a specified value to the System::Variant::Variant (see page 3430).
	Variant::+ (int) (see page 3455)	Adds a specified value to the System::Variant::Variant (see page 3430).
	Variant::+ (Variant&) (see page 3455)	Adds a specified value to the System::Variant::Variant (see page 3430).

38.1.24.5.17.1 Variant::+ Operator (Variant&)

Adds a specified value to the System::Variant::Variant (see page 3430).

Pascal

```
operator +(const rhs: Variant&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Variant operator +(const Variant& rhs) const;
__fastcall Variant operator +(int rhs) const;
__fastcall Variant operator +(double rhs) const;
```

Description

The + System::Variant (see page 3430)::+ returns a new System::Variant::Variant (see page 3430) that is the sum of this System::Variant::Variant (see page 3430) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.18 Variant::+= Operator

Adds the System::Variant::Variant (see page 3430) specified by rhs to this System::Variant::Variant (see page 3430) object.

Pascal

```
operator +=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator +=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.19 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Variant::< (char*) (see page 3456)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.
	Variant::< (double) (see page 3456)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.
	Variant::< (int) (see page 3456)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.
	Variant::< (TDateTime) (see page 3456)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.
	Variant::< (Variant&) (see page 3456)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.

38.1.24.5.19.1 Variant::< Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3430)'s value is less than a specified value.

Pascal

```
operator <(const rhs: Variant&);
operator <(const rhs: char*);
operator <(rhs: Integer);
operator <(rhs: double);
operator <(rhs: TDateTime);
```

C++

```
__fastcall bool operator <(const Variant& rhs) const;
__fastcall bool operator <(const char* rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
__fastcall bool operator <(TDateTime rhs) const;
```

Description

The < System::Variant (see page 3430)::< returns true if the System::Variant::Variant (see page 3430)'s value is less than the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.20 << Operator

This is the overview for the << operator overload.

Overload List

	Name	Description
	Variant::<< (AnsiString&) (see page 3457)	This is <<, a member of class Variant.
	Variant::<< (double) (see page 3457)	This is <<, a member of class Variant.
	Variant::<< (int) (see page 3457)	This is <<, a member of class Variant.
	Variant::<< (Variant&) (see page 3457)	This is <<, a member of class Variant.

38.1.24.5.20.1 Variant::<< Operator (AnsiString&)

Pascal

```
operator <<(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator <<(const AnsiString& rhs) const;
```

Description

This is <<, a member of class Variant.

38.1.24.5.20.2 Variant::<< Operator (Variant&)

Pascal

```
operator <<(const rhs: Variant&);
```

C++

```
__fastcall Variant operator <<(const Variant& rhs) const;
```

Description

This is <<, a member of class Variant.

38.1.24.5.20.3 Variant::<< Operator (double)

Pascal

```
operator <<(rhs: double);
```

C++

```
__fastcall Variant operator <<(double rhs) const;
```

Description

This is <<, a member of class Variant.

38.1.24.5.20.4 Variant::<< Operator (int)

Pascal

```
operator <<(rhs: Integer);
```

C++

```
__fastcall Variant operator <<(int rhs) const;
```

Description

This is <<, a member of class Variant.

38.1.24.5.21 Variant::<<= Operator

Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3430).

Pascal

```
operator <<=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator <<=(const Variant& rhs);
```

Description

If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.22 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Variant::<= (char*) (see page 3458)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.
	Variant::<= (double) (see page 3458)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.
	Variant::<= (int) (see page 3458)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.
	Variant::<= (TDateTime) (see page 3458)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.
	Variant::<= (Variant&) (see page 3458)	Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.

38.1.24.5.22.1 Variant::<= Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3430)'s value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Variant&);
operator <=(const rhs: char*);
operator <=(rhs: Integer);
operator <=(rhs: double);
operator <=(rhs: TDateTime);
```

C++

```
__fastcall bool operator <=(const Variant& rhs) const;
__fastcall bool operator <=(const char* rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
__fastcall bool operator <=(TDateTime rhs) const;
```

Description

The <= System::Variant (see page 3430)::<= returns true if the System::Variant::Variant (see page 3430)'s value is less than

or equal to the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.23 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Variant::= (__int64) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (__int64*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (AnsiString&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (bool) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (char) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (char*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (Currency&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (Currency*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (double) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (double*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (float) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (float*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (IDispatch*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).

	Variant::= (IDispatch**) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (IInterface*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (IInterface**) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (int) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (int*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (IUnknown*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (IUnknown**) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (long) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (long*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (short) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (short*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (signed char) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (signed char*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (System::DelphiInterface<IDispatch>&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (TDateTime&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (TDateTime*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (UnicodeString&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).

	Variant::= (unsigned char) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned char*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned int) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned int*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned long) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned long*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned short) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (unsigned short*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (Variant&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (Variant*) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).
	Variant::= (WideString&) (see page 3461)	Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).

38.1.24.5.23.1 Variant::= Operator (AnsiString&)

Copies the System::Variant::Variant (see page 3430) specified by rhs into this System::Variant::Variant (see page 3430).

Pascal

```
operator =(const rhs: Variant&);
operator =(const src: bool);
operator =(const src: char);
operator =(const src: signed char);
operator =(const src: unsigned char);
operator =(const src: short);
operator =(const src: unsigned short);
operator =(const src: Integer);
operator =(const src: unsigned int);
operator =(const src: long);
operator =(const src: unsigned long);
operator =(const src: float);
operator =(const src: double);
operator =(const src: __int64);
operator =(const src: AnsiString&);
```

```
operator =(const src: Currency&);
operator =(const src: TDateTime&);
operator =(const src: WideString&);
operator =(const src: UnicodeString&);
operator =(const src: System::DelphiInterface<IDispatch>&);
operator =(var src: IUnknown);
operator =(var src: IInterface);
operator =(var src: IDispatch);
operator =(const src: char*);
operator =(var src: signed char);
operator =(var src: unsigned char);
operator =(var src: short);
operator =(var src: unsigned short);
operator =(var src: Integer);
operator =(var src: unsigned int);
operator =(var src: long);
operator =(var src: unsigned long);
operator =(var src: float);
operator =(var src: double);
operator =(var src: __int64);
operator =(var src: IUnknown*);
operator =(var src: IInterface*);
operator =(var src: IDispatch*);
operator =(var src: Currency);
operator =(var src: TDateTime);
operator =(var src: Variant);
```

C++

```
__fastcall Variant& operator =(const Variant& rhs);
__fastcall Variant& operator =(const bool src);
__fastcall Variant& operator =(const char src);
__fastcall Variant& operator =(const signed char src);
__fastcall Variant& operator =(const unsigned char src);
__fastcall Variant& operator =(const short src);
__fastcall Variant& operator =(const unsigned short src);
__fastcall Variant& operator =(const int src);
__fastcall Variant& operator =(const unsigned int src);
__fastcall Variant& operator =(const long src);
__fastcall Variant& operator =(const unsigned long src);
__fastcall Variant& operator =(const float src);
__fastcall Variant& operator =(const double src);
__fastcall Variant& operator =(const __int64 src);
__fastcall Variant& operator =(const AnsiString& src);
__fastcall Variant& operator =(const Currency& src);
__fastcall Variant& operator =(const TDateTime& src);
__fastcall Variant& operator =(const WideString& src);
__fastcall Variant& operator =(const UnicodeString& src);
__fastcall Variant& operator =(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant& operator =(IUnknown* src);
__fastcall Variant& operator =(IInterface* src);
__fastcall Variant& operator =(IDispatch* src);
__fastcall Variant& operator =(const char* src);
__fastcall Variant& operator =(signed char* src);
__fastcall Variant& operator =(unsigned char* src);
__fastcall Variant& operator =(short* src);
__fastcall Variant& operator =(unsigned short* src);
__fastcall Variant& operator =(int* src);
__fastcall Variant& operator =(unsigned int* src);
__fastcall Variant& operator =(long* src);
__fastcall Variant& operator =(unsigned long* src);
__fastcall Variant& operator =(float* src);
__fastcall Variant& operator =(double* src);
__fastcall Variant& operator =(__int64* src);
__fastcall Variant& operator =(IUnknown** src);
__fastcall Variant& operator =(IInterface** src);
__fastcall Variant& operator =(IDispatch** src);
```

```
__fastcall Variant& operator =(Currency* src);
__fastcall Variant& operator =(TDateTime* src);
__fastcall Variant& operator =(Variant* src);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.24 Variant::-= Operator

Subtracts the System::Variant::Variant (see page 3430) specified by rhs from this System::Variant::Variant (see page 3430) object.

Pascal

```
operator -=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator -=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4997) determine the meaning of this operation. If the System::Variant::Variants (see page 4997) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.25 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Variant::== (AnsiString&) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.
	Variant::== (char*) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.
	Variant::== (double) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.
	Variant::== (int) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.
	Variant::== (TDateTime) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.
	Variant::== (Variant&) (see page 3463)	Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.

38.1.24.5.25.1 Variant::== Operator (AnsiString&)

Compares the System::Variant::Variant (see page 3430) with another value, testing for equality.

Pascal

```
operator ==(const rhs: Variant&);
operator ==(const rhs: char*);
operator ==(rhs: Integer);
operator ==(const rhs: AnsiString&);
operator ==(rhs: double);
operator ==(rhs: TDateTime);
```

C++

```
__fastcall bool operator ==(const Variant& rhs) const;
__fastcall bool operator ==(const char* rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(const AnsiString& rhs) const;
__fastcall bool operator ==(double rhs) const;
__fastcall bool operator ==(TDateTime rhs) const;
```

Description

The == System::Variant (see page 3430)::== returns true if the System::Variant::Variant (see page 3430)'s value is the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.26 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Variant::> (char*) (see page 3465)	This is >, a member of class Variant.
	Variant::> (double) (see page 3465)	This is >, a member of class Variant.
	Variant::> (int) (see page 3465)	This is >, a member of class Variant.
	Variant::> (TDateTime) (see page 3464)	This is >, a member of class Variant.
	Variant::> (Variant&) (see page 3464)	This is >, a member of class Variant.

38.1.24.5.26.1 Variant::> Operator (TDateTime)

Pascal

```
operator >(rhs: TDateTime);
```

C++

```
__fastcall bool operator >(TDateTime rhs) const;
```

Description

This is >, a member of class Variant.

38.1.24.5.26.2 Variant::> Operator (Variant&)

Pascal

```
operator >(const rhs: Variant&);
```

C++

```
__fastcall bool operator >(const Variant& rhs) const;
```

Description

This is >, a member of class Variant.

38.1.24.5.26.3 Variant::> Operator (char*)

Pascal

```
operator >(const rhs: char*);
```

C++

```
__fastcall bool operator >(const char* rhs) const;
```

Description

This is >, a member of class Variant.

38.1.24.5.26.4 Variant::> Operator (double)

Pascal

```
operator >(rhs: double);
```

C++

```
__fastcall bool operator >(double rhs) const;
```

Description

This is >, a member of class Variant.

38.1.24.5.26.5 Variant::> Operator (int)

Pascal

```
operator >(rhs: Integer);
```

C++

```
__fastcall bool operator >(int rhs) const;
```

Description

This is >, a member of class Variant.

38.1.24.5.27 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Variant::>= (char*) (see page 3466)	This is >=, a member of class Variant.
	Variant::>= (double) (see page 3466)	This is >=, a member of class Variant.
	Variant::>= (int) (see page 3466)	This is >=, a member of class Variant.
	Variant::>= (TDateTime) (see page 3466)	This is >=, a member of class Variant.
	Variant::>= (Variant&) (see page 3466)	This is >=, a member of class Variant.

38.1.24.5.27.1 Variant::>= Operator (TDateTime)

Pascal

```
operator >=(rhs: TDateTime);
```

C++

```
__fastcall bool operator >=(TDateTime rhs) const;
```

Description

This is >=, a member of class Variant.

38.1.24.5.27.2 Variant::>= Operator (Variant&)

Pascal

```
operator >=(const rhs: Variant&);
```

C++

```
__fastcall bool operator >=(const Variant& rhs) const;
```

Description

This is >=, a member of class Variant.

38.1.24.5.27.3 Variant::>= Operator (char*)

Pascal

```
operator >=(const rhs: char*);
```

C++

```
__fastcall bool operator >=(const char* rhs) const;
```

Description

This is >=, a member of class Variant.

38.1.24.5.27.4 Variant::>= Operator (double)

Pascal

```
operator >=(rhs: double);
```

C++

```
__fastcall bool operator >=(double rhs) const;
```

Description

This is >=, a member of class Variant.

38.1.24.5.27.5 Variant::>= Operator (int)

Pascal

```
operator >=(rhs: Integer);
```

C++

```
__fastcall bool operator >=(int rhs) const;
```

Description

This is >=, a member of class Variant.

38.1.24.5.28 >> Operator

This is the overview for the >> operator overload.

Overload List

	Name	Description
	Variant::>> (AnsiString&) (see page 3467)	This is >>, a member of class Variant.
	Variant::>> (double) (see page 3467)	This is >>, a member of class Variant.
	Variant::>> (int) (see page 3467)	This is >>, a member of class Variant.
	Variant::>> (Variant&) (see page 3467)	This is >>, a member of class Variant.

38.1.24.5.28.1 Variant::>> Operator (AnsiString&)

Pascal

```
operator >>(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator >>(const AnsiString& rhs) const;
```

Description

This is >>, a member of class Variant.

38.1.24.5.28.2 Variant::>> Operator (Variant&)

Pascal

```
operator >>(const rhs: Variant&);
```

C++

```
__fastcall Variant operator >>(const Variant& rhs) const;
```

Description

This is >>, a member of class Variant.

38.1.24.5.28.3 Variant::>> Operator (double)

Pascal

```
operator >>(rhs: double);
```

C++

```
__fastcall Variant operator >>(double rhs) const;
```

Description

This is >>, a member of class Variant.

38.1.24.5.28.4 Variant::>> Operator (int)

Pascal

```
operator >>(rhs: Integer);
```

C++

```
__fastcall Variant operator >>(int rhs) const;
```

Description

This is >>, a member of class Variant.

38.1.24.5.29 Variant::>>= Operator

Pascal

```
operator >>=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator >>=(const Variant& rhs);
```

Description

This is >>=, a member of class Variant.

38.1.24.5.30 Variant::AnsiString Operator

Converts the System::Variant::Variant (see page 3430) to an AnsiString object.

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

This conversion System::Variant::AnsiString returns an AnsiString object that is a conversion of this System::Variant::Variant (see page 3430).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.31 Variant::bool Operator

Converts the System::Variant::Variant (see page 3430) to a boolean value.

Pascal

```
operator bool;
```

C++

```
__fastcall operator bool() const;
```

Description

This conversion System::Variant::bool returns a bool that is a conversion of this System::Variant::Variant (see page 3430).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.32 Variant::char Operator

Pascal

```
operator char;
```

C++

```
__fastcall operator char() const;
```

Description

This is char, a member of class Variant.

38.1.24.5.33 Variant::Currency Operator

Converts the System::Variant::Variant (see page 3430) to a Currency object.

Pascal

```
operator Currency;
```

C++

```
__fastcall operator Currency() const;
```

Description

The Currency conversion System::Variant::Currency returns a Currency or a reference to a Currency that is a conversion of this System::Variant::Variant (see page 3430). If the Currency* conversion is used, the System::Variant::Currency sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.34 Variant::Currency* Operator

Pascal

```
operator Currency*;
```

C++

```
__fastcall operator Currency*();
```

Description

This is Currency*, a member of class Variant.

38.1.24.5.35 Variant::double Operator

Converts the System::Variant::Variant (see page 3430) to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::Variant::double returns a double or a reference to a double that is a conversion of this System::Variant::Variant (see page 3430). If the double* conversion is used, the System::Variant::double sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.36 Variant::double* Operator

Pascal

```
operator double*;
```

C++

```
__fastcall operator double*();
```

Description

This is double*, a member of class Variant.

38.1.24.5.37 Variant::float Operator

Converts the System::Variant::Variant (see page 3430) to a float value.

Pascal

```
operator float;
```

C++

```
__fastcall operator float() const;
```

Description

The float conversion System::Variant::float returns a float or a reference to a float that is a conversion of this System::Variant::Variant (see page 3430). If the float* conversion is used, the System::Variant::float sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.38 Variant::float* Operator

Pascal

```
operator float*;
```

C++

```
__fastcall operator float*();
```

Description

This is float*, a member of class Variant.

38.1.24.5.39 Variant::IDispatch* Operator

Pascal

```
operator IDispatch*;
```

C++

```
__fastcall operator IDispatch*();
```

Description

This is IDispatch*, a member of class Variant.

38.1.24.5.40 Variant::IInterface* Operator

Pascal

```
operator IInterface*;
```

C++

```
__fastcall operator IInterface*();
```

Description

This is IInterface*, a member of class Variant.

38.1.24.5.41 Variant::int Operator

Converts the System::Variant::Variant (see page 3430) to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::Variant::int returns an integer or a reference to an integer that is a conversion of this System::Variant::Variant (see page 3430). If the int* conversion is used, the System::Variant::int sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.42 Variant::int* Operator

Pascal

```
operator int*;
```

C++

```
__fastcall operator int*();
```

Description

This is int*, a member of class Variant.

38.1.24.5.43 Variant::IUnknown* Operator

Pascal

```
operator IUnknown*;
```

C++

```
__fastcall operator IUnknown*();
```

Description

This is IUnknown*, a member of class Variant.

38.1.24.5.44 Variant::long Operator

Pascal

```
operator long;
```

C++

```
__fastcall operator long() const;
```

Description

This is long, a member of class Variant.

38.1.24.5.45 Variant::long* Operator

Pascal

```
operator long*;
```

C++

```
__fastcall operator long*();
```

Description

This is long*, a member of class Variant.

38.1.24.5.46 Variant::short Operator

Converts the System::Variant::Variant (see page 3430) to a short value.

Pascal

```
operator short;
```

C++

```
__fastcall operator short() const;
```

Description

The short conversion System::Variant::short returns a short or a reference to a short that is a conversion of this System::Variant::Variant (see page 3430). If the short* conversion is used, the System::Variant::short sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.47 Variant::short* Operator

Pascal

```
operator short*;
```

C++

```
__fastcall operator short*();
```

Description

This is short*, a member of class Variant.

38.1.24.5.48 Variant::signed char Operator

Pascal

```
operator signed char;
```

C++

```
__fastcall operator signed char() const;
```

Description

This is signed char, a member of class Variant.

38.1.24.5.49 Variant::signed char* Operator

Pascal

```
operator signed char*;
```

C++

```
__fastcall operator signed char*();
```

Description

This is signed char*, a member of class Variant.

38.1.24.5.50 Variant::TDateTime Operator

Converts the System::Variant::Variant (see page 3430) to a TDateTime object.

Pascal

```
operator TDateTime;
```

C++

```
__fastcall operator TDateTime() const;
```

Description

The TDateTime conversion System::Variant::TDateTime returns a TDateTime or a reference to a TDateTime that is a conversion of this System::Variant::Variant (see page 3430). If the TDateTime* conversion is used, the System::Variant::TDateTime sets the varByRef (see page 3930) bit, meaning that the System::Variant::Variant (see page 3430) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

38.1.24.5.51 Variant::TDateTime* Operator

Pascal

```
operator TDateTime*;
```

C++

```
__fastcall operator TDateTime*();
```

Description

This is TDateTime*, a member of class Variant.

38.1.24.5.52 Variant::UnicodeString Operator

Pascal

```
operator UnicodeString;
```

C++

```
__fastcall operator UnicodeString() const;
```

Description

This is UnicodeString, a member of class Variant.

38.1.24.5.53 Variant::unsigned char Operator

Pascal

```
operator unsigned char;
```

C++

```
__fastcall operator unsigned char() const;
```

Description

This is unsigned char, a member of class Variant.

38.1.24.5.54 Variant::unsigned char* Operator

Pascal

```
operator unsigned char*;
```

C++

```
__fastcall operator unsigned char*();
```

Description

This is unsigned char*, a member of class Variant.

38.1.24.5.55 Variant::unsigned int Operator

Pascal

```
operator unsigned int;
```

C++

```
__fastcall operator unsigned int() const;
```

Description

This is unsigned int, a member of class Variant.

38.1.24.5.56 Variant::unsigned int* Operator

Pascal

```
operator unsigned int*;
```

C++

```
__fastcall operator unsigned int*();
```

Description

This is unsigned int*, a member of class Variant.

38.1.24.5.57 Variant::unsigned long Operator

Pascal

```
operator unsigned long;
```

C++

```
__fastcall operator unsigned long() const;
```

Description

This is unsigned long, a member of class Variant.

38.1.24.5.58 Variant::unsigned long* Operator

Pascal

```
operator unsigned long*;
```

C++

```
__fastcall operator unsigned long*();
```

Description

This is unsigned long*, a member of class Variant.

38.1.24.5.59 Variant::unsigned short Operator

Pascal

```
operator unsigned short;
```

C++

```
__fastcall operator unsigned short() const;
```

Description

This is unsigned short, a member of class Variant.

38.1.24.5.60 Variant::unsigned short* Operator

Pascal

```
operator unsigned short*;
```

C++

```
__fastcall operator unsigned short*();
```

Description

This is unsigned short*, a member of class Variant.

38.1.24.5.61 Variant::WideString Operator

Pascal

```
operator WideString;
```

C++

```
__fastcall operator WideString() const;
```

Description

This is WideString, a member of class Variant.

38.1.25 WideString Class

System::WideString represents strings of 16-bit characters.

Class Hierarchy

Pascal

```
WideString = class;
```

C++

```
class WideString;
```

File

wstring.h

Description

System::WideString represents strings of 16-bit characters. Each character is stored as a wchar_t value. The C++ class System::WideString is functionally equivalent to a Delphi intrinsic type, also called System::WideString.

Note: Although System::WideString is a platform-independent data (see page 3483) type and often appears in cross-platform code, System::WideString is also useful for working with Microsoft's Component Object Model (COM) System::WideString values are easily converted to or from the BSTR values used in COM.

38.1.25.1 WideString Members

The following tables list the members exposed by WideString.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3480)	This is record System::WideString::TStringFloatFormat.

Public Methods

	Name	Description
	~WideString (see page 3478)	Frees an instance ofSystem::WideString::WideString (see page 3476).
	c_bstr (see page 3481)	Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3476).
	cat_printf (see page 3482)	This is cat_printf, a member of class WideString.
	cat_sprintf (see page 3482)	This is cat_sprintf, a member of class WideString.

	cat_vprintf (see page 3482)	This is cat_vprintf, a member of class WideString.
	Copy (see page 3482)	Returns a copy of theSystem::WideString::WideString (see page 3476).
	CurrToStr (see page 3483)	This is CurrToStr, a member of class WideString.
	CurrToStrF (see page 3483)	This is CurrToStrF, a member of class WideString.
	data (see page 3483)	This is data, a member of class WideString.
	Delete (see page 3484)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 3476).
	FloatToStrF (see page 3484)	This is FloatToStrF, a member of class WideString.
	Format (see page 3484)	This is Format, a member of class WideString.
	FormatFloat (see page 3484)	This is FormatFloat, a member of class WideString.
	Insert (see page 3485)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3476).
	IntToHex (see page 3485)	This is IntToHex, a member of class WideString.
	IsDelimiter (see page 3485)	This is IsDelimiter, a member of class WideString.
	IsEmpty (see page 3485)	Returns true if the System::WideString::WideString (see page 3476) is empty.
	IsPathDelimiter (see page 3486)	This is IsPathDelimiter, a member of class WideString.
	LastChar (see page 3486)	This is LastChar, a member of class WideString.
	LastDelimiter (see page 3486)	This is LastDelimiter, a member of class WideString.
	Length (see page 3486)	Returns the length of theSystem::WideString::WideString (see page 3476).
	LowerCase (see page 3487)	This is LowerCase, a member of class WideString.
	Pos (see page 3487)	Returns the index at which a specified substring begins.
	printf (see page 3487)	This is printf, a member of class WideString.
	SetLength (see page 3487)	Changes the length of theSystem::WideString::WideString (see page 3476).
	sprintf (see page 3488)	This is sprintf, a member of class WideString.
	StringOfChar (see page 3488)	This is StringOfChar, a member of class WideString.
	SubString (see page 3488)	Returns a specified substring of theSystem::WideString::WideString (see page 3476).
	swap (see page 3489)	This is swap, a member of class WideString.
	ToDouble (see page 3489)	This is ToDouble, a member of class WideString.
	ToInt (see page 3489)	This is ToInt, a member of class WideString.
	ToIntDef (see page 3489)	This is ToIntDef, a member of class WideString.
	Trim (see page 3490)	This is Trim, a member of class WideString.
	TrimLeft (see page 3490)	This is TrimLeft, a member of class WideString.
	TrimRight (see page 3490)	This is TrimRight, a member of class WideString.
	UpperCase (see page 3490)	This is UpperCase, a member of class WideString.
	vprintf (see page 3491)	This is vprintf, a member of class WideString.
	WideString (see page 3478)	This is the overview for the WideString constructor overload.

Public Operators

	Name	Description
	!= (see page 3491)	This is the overview for the != operator overload.
	[] (see page 3492)	This is the overview for the [] operator overload.
	+ (see page 3492)	Concatenates twoSystem::WideString::WideStrings (see page 6767).
	+= (see page 3493)	Concatenates this System::WideString::WideString (see page 3476) and rhs, and assigns the result to this System::WideString::WideString (see page 3476).
	< (see page 3493)	This is the overview for the < operator overload.
	<= (see page 3493)	This is the overview for the <= operator overload.
	= (see page 3494)	Assigns the value of theSystem::WideString::WideString (see page 3476) to equal a source string.
	== (see page 3494)	This is the overview for the == operator overload.
	> (see page 3495)	This is the overview for the > operator overload.
	>= (see page 3496)	This is the overview for the >= operator overload.

38.1.25.2 WideString::~WideString Destructor

Frees an instance ofSystem::WideString::WideString (see page 3476).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~WideString();
```

Description

System::WideString (see page 3476)::~WideString (see page 3476) is the destructor forSystem::WideString::WideString (see page 3476).

38.1.25.3 WideString Constructor

This is the overview for the WideString constructor overload.

Overload List

	Name	Description
	WideString::WideString () (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (__int64) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (AnsiStringT<codePage> &) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (char) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (char*) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (char16_t*, int) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (char32_t*, int) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (double) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (float) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (int) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (long) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (long double) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (short) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).

	WideString::WideString (UnicodeString&) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (unsigned __int64) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (unsigned int) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (unsigned long) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (unsigned short) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (WideChar) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (WideChar*) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (WideChar*, int) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).
	WideString::WideString (WideString&) (see page 3479)	Creates an instance of System::WideString::WideString (see page 3476).

38.1.25.3.1 WideString::WideString Constructor ()

Creates an instance of System::WideString::WideString (see page 3476).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: WideString&);
constructor Create(const src: WideChar*; len: Integer);
constructor Create(const src: WideChar*);
constructor Create(const src: char16_t*; numChar16: Integer = -1);
constructor Create(const src: char32_t*; numChar32: Integer = -1);
constructor Create(const src: WideChar);
constructor Create(src: char);
constructor Create(src: short);
constructor Create(: unsigned short);
constructor Create(src: Integer);
constructor Create(: unsigned int);
constructor Create(: long);
constructor Create(: unsigned long);
constructor Create(: __int64);
constructor Create(: unsigned __int64);
constructor Create(src: float);
constructor Create(src: double);
constructor Create(src: long double);
constructor Create(const src: UnicodeString&);
constructor Create(const src: AnsiStringT<codePage> &);
```

C++

```
__fastcall WideString();
__fastcall WideString(const char* src);
__fastcall WideString(const WideString& src);
__fastcall WideString(const WideChar* src, int len);
__fastcall WideString(const WideChar* src);
__fastcall WideString(const char16_t* src, int numChar16 = -1);
__fastcall WideString(const char32_t* src, int numChar32 = -1);
__fastcall explicit WideString(const WideChar src);
```

```
__fastcall explicit WideString(char src);
__fastcall explicit WideString(short src);
__fastcall explicit WideString(unsigned short);
__fastcall explicit WideString(int src);
__fastcall explicit WideString(unsigned int);
__fastcall explicit WideString(long);
__fastcall explicit WideString(unsigned long);
__fastcall explicit WideString(__int64);
__fastcall explicit WideString(unsigned __int64);
__fastcall explicit WideString(float src);
__fastcall explicit WideString(double src);
__fastcall explicit WideString(long double src);
__fastcall WideString(const UnicodeString& src);
template <unsigned short codePage> __fastcall WideString(const AnsiStringT<codePage> & src);
```

Description

Call System::WideString::WideString (see page 3476) to create and initialize a wide string. If the constructor is given no arguments, the resulting System::WideString::WideString (see page 3476) is an empty string. If the constructor is passed a single wide character, the resulting System::WideString::WideString (see page 3476) has length one, and contains that character. All other versions of the constructor copy the string passed as an argument, converting characters to wchar_t if necessary. When constructing a System::WideString::WideString (see page 3476) from an array of char or wchar_t, the src string should be null terminated, unless a second argument is given that indicates the number of characters in the array.

38.1.25.4 WideString Enumerations

The enumerations of the WideString class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3480)	This is record System::WideString::TStringFloatFormat.

38.1.25.4.1 System::WideString::TStringFloatFormat Enumeration

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

wstring.h

Description

This is record System::WideString::TStringFloatFormat.

38.1.25.5 WideString Methods

The methods of the WideString class are listed here.

Public Methods

	Name	Description
	c_bstr (see page 3481)	Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3476).
	cat_printf (see page 3482)	This is cat_printf, a member of class WideString.
	cat_sprintf (see page 3482)	This is cat_sprintf, a member of class WideString.
	cat_vprintf (see page 3482)	This is cat_vprintf, a member of class WideString.
	Copy (see page 3482)	Returns a copy of theSystem::WideString::WideString (see page 3476).
	CurrToStr (see page 3483)	This is CurrToStr, a member of class WideString.
	CurrToStrF (see page 3483)	This is CurrToStrF, a member of class WideString.
	data (see page 3483)	This is data, a member of class WideString.
	Delete (see page 3484)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 3476).
	FloatToStrF (see page 3484)	This is FloatToStrF, a member of class WideString.
	Format (see page 3484)	This is Format, a member of class WideString.
	FormatFloat (see page 3484)	This is FormatFloat, a member of class WideString.
	Insert (see page 3485)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3476).
	IntToHex (see page 3485)	This is IntToHex, a member of class WideString.
	IsDelimiter (see page 3485)	This is IsDelimiter, a member of class WideString.
	IsEmpty (see page 3485)	Returns true if the System::WideString::WideString (see page 3476) is empty.
	IsPathDelimiter (see page 3486)	This is IsPathDelimiter, a member of class WideString.
	LastChar (see page 3486)	This is LastChar, a member of class WideString.
	LastDelimiter (see page 3486)	This is LastDelimiter, a member of class WideString.
	Length (see page 3486)	Returns the length of theSystem::WideString::WideString (see page 3476).
	LowerCase (see page 3487)	This is LowerCase, a member of class WideString.
	Pos (see page 3487)	Returns the index at which a specified substring begins.
	printf (see page 3487)	This is printf, a member of class WideString.
	SetLength (see page 3487)	Changes the length of theSystem::WideString::WideString (see page 3476).
	sprintf (see page 3488)	This is sprintf, a member of class WideString.
	StringOfChar (see page 3488)	This is StringOfChar, a member of class WideString.
	SubString (see page 3488)	Returns a specified substring of theSystem::WideString::WideString (see page 3476).
	swap (see page 3489)	This is swap, a member of class WideString.
	ToDouble (see page 3489)	This is ToDouble, a member of class WideString.
	ToInt (see page 3489)	This is ToInt, a member of class WideString.
	ToIntDef (see page 3489)	This is ToIntDef, a member of class WideString.
	Trim (see page 3490)	This is Trim, a member of class WideString.
	TrimLeft (see page 3490)	This is TrimLeft, a member of class WideString.
	TrimRight (see page 3490)	This is TrimRight, a member of class WideString.
	UpperCase (see page 3490)	This is UpperCase, a member of class WideString.
	vprintf (see page 3491)	This is vprintf, a member of class WideString.

38.1.25.5.1 WideString::c_bstr Method

Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3476).

Pascal

```
function c_bstr: WideChar*;
```

C++

```
__fastcall WideChar* c_bstr() const;
```

Description

Use c_System::WideString::c_bstr to access the BSTR that implements theSystem::WideString::WideString (see page 3476).

Note: BSTR values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Window.

See Also

System::WideString::c_bstr

System::WideString::Copy (see page 3482)

System::WideString::&

38.1.25.5.2 WideString::cat_printf Method

Pascal

```
function cat_printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const wchar_t* format, ...);
```

Description

This is cat_printf, a member of class WideString.

38.1.25.5.3 WideString::cat_sprintf Method

Pascal

```
function cat_sprintf(const format: wchar_t*; : ...): WideString&; cdecl;
```

C++

```
__cdecl WideString& cat_sprintf(const wchar_t* format, ...);
```

Description

This is cat_sprintf, a member of class WideString.

38.1.25.5.4 WideString::cat_vprintf Method

Pascal

```
function cat_vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const wchar_t* format, va_list);
```

Description

This is cat_vprintf, a member of class WideString.

38.1.25.5.5 WideString::Copy Method

Returns a copy of theSystem::WideString::WideString (see page 3476).

Pascal

```
function Copy(var : wchar_t; len: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* Copy(wchar_t *, int len) const;
```

Description

Use System::WideString::Copy to copy theSystem::WideString::WideString (see page 3476) into an array of wchar_t. Use the Length (see page 3486) method to ensure that the destination buffer is large enough to hold the contents of theSystem::WideString::WideString (see page 3476).

WideString (see page 3476) also supports a static copy method that copies an arbitrary source string.

See Also

System::WideString::bstr

System::WideString::c_bstr (see page 3481)

System::WideString::&

System::WideString::Length (see page 3486)

38.1.25.5.6 WideString::CurrToStr Method

Pascal

```
function CurrToStr(value: Currency): WideString; static;
```

C++

```
static __fastcall WideString CurrToStr(Currency value);
```

Description

This is CurrToStr, a member of class WideString.

38.1.25.5.7 WideString::CurrToStrF Method

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer): WideString;
static;
```

C++

```
static __fastcall WideString CurrToStrF(Currency value, TStringFloatFormat format, int digits);
```

Description

This is CurrToStrF, a member of class WideString.

38.1.25.5.8 WideString::data Method

Pascal

```
function data: WideChar*;
```

C++

```
__fastcall WideChar* data();
```

Description

This is data, a member of class WideString.

38.1.25.5.9 WideString::Delete Method

System::WideString::Deletes a substring from the System::WideString::WideString (see page 3476).

Pascal

```
procedure Delete(index: Integer; count: Integer);
```

C++

```
__fastcall void Delete(int index, int count);
```

Description

System::WideString::Delete removes count characters from the System::WideString::WideString (see page 3476), beginning with the character at position index.

See Also

System::WideString::Insert (see page 3485)

System::WideString::[] (see page 3492)

38.1.25.5.10 WideString::FloatToStrF Method

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): WideString; static;
```

C++

```
static __fastcall WideString FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

This is FloatToStrF, a member of class WideString.

38.1.25.5.11 WideString::Format Method

Pascal

```
function Format(const format: WideString&; const args: TVarRec *; size: Integer): WideString;
static;
```

C++

```
static __fastcall WideString Format(const WideString& format, const TVarRec * args, int size);
```

Description

This is Format, a member of class WideString.

38.1.25.5.12 WideString::FormatFloat Method

Pascal

```
function FormatFloat(const format: WideString&; const value: long double&): WideString; static;
```

C++

```
static __fastcall WideString FormatFloat(const WideString& format, const long double& value);
```

Description

This is FormatFloat, a member of class WideString.

38.1.25.5.13 WideString::Insert Method

System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3476).

Pascal

```
procedure Insert(const str: WideString&; index: Integer);
```

C++

```
__fastcall void Insert(const WideString& str, int index);
```

Description

Call System::WideString::Insert to inserts the string str into thisSystem::WideString::WideString (see page 3476) beginning at the position index.

See Also

System::WideString::Delete (see page 3484)

System::WideString::[] (see page 3492)

38.1.25.5.14 WideString::IntToHex Method

Pascal

```
function IntToHex(value: Integer; digits: Integer): WideString; static;
```

C++

```
static __fastcall WideString IntToHex(int value, int digits);
```

Description

This is IntToHex, a member of class WideString.

38.1.25.5.15 WideString::IsDelimiter Method

Pascal

```
function IsDelimiter(const delimiters: WideString&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const WideString& delimiters, int index) const;
```

Description

This is IsDelimiter, a member of class WideString.

38.1.25.5.16 WideString::IsEmpty Method

Returns true if the System::WideString::WideString (see page 3476) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

Call System::WideString::IsEmpty to determine whether the System::WideString::WideString (see page 3476) represents an empty string.

See Also

System::WideString::Empty

38.1.25.5.17 WideString::IsPathDelimiter Method

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

This is IsPathDelimiter, a member of class WideString.

38.1.25.5.18 WideString::LastChar Method

Pascal

```
function LastChar: wchar_t*;
```

C++

```
__fastcall wchar_t* LastChar() const;
```

Description

This is LastChar, a member of class WideString.

38.1.25.5.19 WideString::LastDelimiter Method

Pascal

```
function LastDelimiter(const delimiters: WideString&): Integer;
```

C++

```
__fastcall int LastDelimiter(const WideString& delimiters) const;
```

Description

This is LastDelimiter, a member of class WideString.

38.1.25.5.20 WideString::Length Method

Returns the length of theSystem::WideString::WideString (see page 3476).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::WideString::Length returns the number of characters in thisSystem::WideString::WideString (see page 3476).

38.1.25.5.21 WideString::LowerCase Method

Pascal

```
function LowerCase: WideString;
```

C++

```
__fastcall WideString LowerCase() const;
```

Description

This is LowerCase, a member of class WideString.

38.1.25.5.22 WideString::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: WideString&): Integer;
```

C++

```
__fastcall int Pos(const WideString& subStr) const;
```

Description

Use System::WideString::Pos to locate a substring of theSystem::WideString::WideString (see page 3476). System::WideString::Pos returns the index of the first character in subStr. If subStr is not contained in theSystem::WideString::WideString (see page 3476), System::WideString::Pos returns 0.

See Also

System::WideString::Length (see page 3486)

System::WideString::SubString (see page 3488)

38.1.25.5.23 WideString::printf Method

Pascal

```
function printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const wchar_t* format, ...);
```

Description

This is printf, a member of class WideString.

38.1.25.5.24 WideString::SetLength Method

Changes the length of theSystem::WideString::WideString (see page 3476).

Pascal

```
procedure SetLength(newLength: Integer);
```

C++

```
__fastcall void SetLength(int newLength);
```

Description

System::WideString::SetLength changes theSystem::WideString::WideString (see page 3476)'s length to newLength, truncating if necessary.

System::WideString::SetLength reallocates the underlying array of characters that stores the value of theSystem::WideString::WideString (see page 3476). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate theSystem::WideString::WideString (see page 3476), an EOutOfMemory exception is thrown. Following a call to System::WideString::SetLength, theSystem::WideString::WideString (see page 3476) is guaranteed to reference a unique string — that is a string with a reference count of one.

See Also

System::WideString::Length (see page 3486)

38.1.25.5.25 WideString::sprintf Method

Pascal

```
function sprintf(const format: wchar_t*; : ...): WideString&; cdecl;
```

C++

```
__cdecl WideString& sprintf(const wchar_t* format, ...);
```

Description

This is sprintf, a member of class WideString.

38.1.25.5.26 WideString::StringOfChar Method

Pascal

```
function StringOfChar(ch: WideChar; count: Integer): WideString; static;
```

C++

```
static __fastcall WideString StringOfChar(WideChar ch, int count);
```

Description

This is StringOfChar, a member of class WideString.

38.1.25.5.27 WideString::SubString Method

Returns a specified substring of theSystem::WideString::WideString (see page 3476).

Pascal

```
function SubString(index: Integer; count: Integer): WideString;
```

C++

```
__fastcall WideString SubString(int index, int count) const;
```

Description

System::WideString::SubString returns a newSystem::WideString::WideString (see page 3476) that is a substring of thisSystem::WideString::WideString (see page 3476). The substring contains count characters beginning at index.

See Also

System::WideString::Length (see page 3486)

System::WideString::pos

38.1.25.5.28 WideString::swap Method

Pascal

```
function swap(var other: WideString): WideString&;
```

C++

```
WideString& swap(WideString& other);
```

Description

This is swap, a member of class WideString.

38.1.25.5.29 WideString::ToDouble Method

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

This is ToDouble, a member of class WideString.

38.1.25.5.30 WideString::ToInt Method

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

This is ToInt, a member of class WideString.

38.1.25.5.31 WideString::ToIntDef Method

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

This is ToIntDef, a member of class WideString.

38.1.25.5.32 WideString::Trim Method

Pascal

```
function Trim: WideString;
```

C++

```
__fastcall WideString Trim() const;
```

Description

This is Trim, a member of class WideString.

38.1.25.5.33 WideString::TrimLeft Method

Pascal

```
function TrimLeft: WideString;
```

C++

```
__fastcall WideString TrimLeft() const;
```

Description

This is TrimLeft, a member of class WideString.

38.1.25.5.34 WideString::TrimRight Method

Pascal

```
function TrimRight: WideString;
```

C++

```
__fastcall WideString TrimRight() const;
```

Description

This is TrimRight, a member of class WideString.

38.1.25.5.35 WideString::UpperCase Method

Pascal

```
function UpperCase: WideString;
```

C++

```
__fastcall WideString UpperCase() const;
```

Description

This is UpperCase, a member of class WideString.

38.1.25.5.36 WideString::vprintf Method

Pascal

```
function vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const wchar_t* format, va_list);
```

Description

This is vprintf, a member of class WideString.

38.1.25.6 WideString Operators

The operators of the WideString class are listed here.

Public Operators

	Name	Description
	!= (see page 3491)	This is the overview for the != operator overload.
	[] (see page 3492)	This is the overview for the [] operator overload.
	+ (see page 3492)	Concatenates twoSystem::WideString::WideStrings (see page 6767).
	+= (see page 3493)	Concatenates this System::WideString::WideString (see page 3476) and rhs, and assigns the result to this System::WideString::WideString (see page 3476).
	< (see page 3493)	This is the overview for the < operator overload.
	<= (see page 3493)	This is the overview for the <= operator overload.
	= (see page 3494)	Assigns the value of theSystem::WideString::WideString (see page 3476) to equal a source string.
	== (see page 3494)	This is the overview for the == operator overload.
	> (see page 3495)	This is the overview for the > operator overload.
	>= (see page 3496)	This is the overview for the >= operator overload.

38.1.25.6.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	WideString::!= (char*) (see page 3491)	Tests whether theSystem::WideString::WideString (see page 3476) is different from anotherSystem::WideString::WideString (see page 3476).
	WideString::!= (WideString&) (see page 3491)	Tests whether theSystem::WideString::WideString (see page 3476) is different from anotherSystem::WideString::WideString (see page 3476).

38.1.25.6.1.1 WideString::!= Operator (WideString&)

Tests whether theSystem::WideString::WideString (see page 3476) is different from anotherSystem::WideString::WideString (see page 3476).

Pascal

```
operator !=(const rhs: WideString&);
operator !=(const s: char*);
```

C++

```
__fastcall bool operator !=(const WideString& rhs) const;
__fastcall bool operator !=(const char* s) const;
```

Description

The != System::WideString (see page 3476)::!= returns true if the strings are not the same, false otherwise.

See Also

System::WideString::== (see page 3495)

38.1.25.6.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	WideString::[] (int) (see page 3492)	Returns the character at a specified index in the string.
	WideString::[] (int) (see page 3492)	Returns the character at a specified index in the string.

38.1.25.6.2.1 WideString::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall WideChar & operator [](const int idx);
__fastcall const WideChar& operator [](const int idx) const;
```

Description

The System::WideString (see page 3476)::[] returns the character in the string at index value idx. The System::WideString (see page 3476)::[] assumes a base index of 1. For example, given the string “Hello” and an index value of 2, the character returned would be ‘e’.

38.1.25.6.3 WideString::+ Operator

Concatenates twoSystem::WideString::WideStrings (see page 6767).

Pascal

```
operator +(const rhs: WideString&);
```

C++

```
__fastcall WideString operator +(const WideString& rhs) const;
```

Description

The + System::WideString (see page 3476)::+ returns aSystem::WideString::WideString (see page 3476) that has the value passed as rhs concatenated to the end. A related friend System::WideString (see page 3476)::+ takes two arguments and returns the concatenation of lhs with rhs.

38.1.25.6.4 WideString::+= Operator

Concatenates this System::WideString::WideString (see page 3476) and rhs, and assigns the result to this System::WideString::WideString (see page 3476).

Pascal

```
operator +=(const rhs: WideString&);
```

C++

```
__fastcall WideString& operator +=(const WideString& rhs);
```

Description

Use += to concatenate rhs to the end of the System::WideString::WideString (see page 3476). This System::WideString (see page 3476)::+= is not supported when the System::WideString::WideString (see page 3476) is a property value.

38.1.25.6.5 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	WideString::< (char*) (see page 3493)	Compares two strings.
	WideString::< (WideString&) (see page 3493)	Compares two strings.

38.1.25.6.5.1 WideString::< Operator (WideString&)

Compares two strings.

Pascal

```
operator <(const rhs: WideString&);
operator <(const s: char*);
```

C++

```
__fastcall bool operator <(const WideString& rhs) const;
__fastcall bool operator <(const char* s) const;
```

Description

The < System::WideString (see page 3476)::< returns true if thisSystem::WideString::WideString (see page 3476) is less than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 3494)

System::WideString::> (see page 3495)

38.1.25.6.6 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	WideString::<= (char*) (see page 3494)	Compares two strings.
	WideString::<= (WideString&) (see page 3494)	Compares two strings.

38.1.25.6.6.1 WideString::<= Operator (WideString&)

Compares two strings.

Pascal

```
operator <=(const rhs: WideString&);
operator <=(const s: char*);
```

C++

```
__fastcall bool operator <=(const WideString& rhs) const;
__fastcall bool operator <=(const char* s) const;
```

Description

The <= System::WideString (see page 3476)::<= returns true if thisSystem::WideString::WideString (see page 3476) is less than or equal to rhs, false otherwise. The strings are compared using their numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 3496)

System::WideString::< (see page 3493)

38.1.25.6.7 WideString::= Operator

Assigns the value of theSystem::WideString::WideString (see page 3476) to equal a source string.

Pascal

```
operator =(const rhs: WideString&);
```

C++

```
__fastcall WideString& operator =(const WideString& rhs);
```

Description

Use the assignment System::WideString (see page 3476)::= to set the value of theSystem::WideString::WideString (see page 3476) to match the string given by rhs.

Note: BSTR values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Windows.

38.1.25.6.8 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	WideString::== (char*) (see page 3495)	Compares theSystem::WideString::WideString (see page 3476) with a specified string, returning true if they are the same.
	WideString::== (WideString&) (see page 3495)	Compares theSystem::WideString::WideString (see page 3476) with a specified string, returning true if they are the same.

38.1.25.6.8.1 WideString::== Operator (WideString&)

Compares theSystem::WideString::WideString (see page 3476) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: WideString&);
operator ==(const s: char*);
```

C++

```
__fastcall bool operator ==(const WideString& rhs) const;
__fastcall bool operator ==(const char* s) const;
```

Description

The equality System::WideString (see page 3476)::== returns true if thisSystem::WideString::WideString (see page 3476) is the same as rhs, false otherwise.

See Also

System::WideString::!= (see page 3491)

38.1.25.6.9 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	WideString::> (char*) (see page 3495)	Compares two strings.
	WideString::> (WideString&) (see page 3495)	Compares two strings.

38.1.25.6.9.1 WideString::> Operator (WideString&)

Compares two strings.

Pascal

```
operator >(const rhs: WideString&);
operator >(const s: char*);
```

C++

```
__fastcall bool operator >(const WideString& rhs) const;
__fastcall bool operator >(const char* s) const;
```

Description

The System::WideString (see page 3476)::> returns true if thisSystem::WideString::WideString (see page 3476) is greater

than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 3496)

System::WideString::< (see page 3493)

38.1.25.6.10 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	WideString::>= (char*) (see page 3496)	Compares two strings.
	WideString::>= (WideString&) (see page 3496)	Compares two strings.

38.1.25.6.10.1 WideString::>= Operator (WideString&)

Compares two strings.

Pascal

```
operator >=(const rhs: WideString&);
operator >=(const s: char*);
```

C++

```
__fastcall bool operator >=(const WideString& rhs) const;
__fastcall bool operator >=(const char* s) const;
```

Description

The System::WideString (see page 3476)::>= returns true if thisSystem::WideString::WideString (see page 3476) is greater than or equal to rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 3494)

System::WideString::> (see page 3495)

38.2 Interfaces

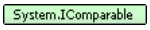
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IComparable (see page 3497)	This is class System.IComparable.
	IDispatch (see page 3498)	This is class System.IDispatch.
	IEnumerable (see page 3500)	This is class System.IEnumerable.
	IEnumerator (see page 3501)	This is class System.IEnumerator.
	IInterface (see page 3503)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 3505)	IInvokable is the base class for invokable interfaces.

38.2.1 IComparable Interface

Class Hierarchy



Pascal

```
IComparable = interface;
```

C++

```
__interface IComparable;
```

File

System

Description

This is class System.IComparable.

38.2.1.1 IComparable Members

The following tables list the members exposed by IComparable.

Public Methods

	Name	Description
	CompareTo (see page 3498)	This is CompareTo, a member of class IComparable.

38.2.1.2 IComparable Methods

The methods of the IComparable class are listed here.

Public Methods

	Name	Description
	CompareTo (see page 3498)	This is CompareTo, a member of class IComparable.

38.2.1.2.1 IComparable.CompareTo Method

Pascal

```
function CompareTo(Obj: TObject): Integer;
```

C++

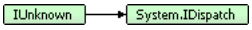
```
__fastcall int CompareTo(TObject * Obj);
```

Description

This is CompareTo, a member of class IComparable.

38.2.2 IDispatch Interface

Class Hierarchy



Pascal

```
IDispatch = interface(IUnknown);
```

C++

```
__interface IDispatch : public IUnknown;
```

File

System

Description

This is class System.IDispatch.

38.2.2.1 IDispatch Members

The following tables list the members exposed by IDispatch.

Public Methods

	Name	Description
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.

38.2.2.2 IDispatch Methods

The methods of the IDispatch class are listed here.

Public Methods

	Name	Description
	GetIDsOfNames (see page 3499)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 3499)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3499)	This is GetTypeInfoCount, a member of class IDispatch.

	Invoke (see page 3499)	This is Invoke, a member of class IDispatch.

38.2.2.2.1 IDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class IDispatch.

38.2.2.2.2 IDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class IDispatch.

38.2.2.2.3 IDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class IDispatch.

38.2.2.2.4 IDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

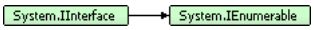
```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class IDispatch.

38.2.3 IEnumerable Interface

Class Hierarchy



Pascal

```
IEnumerable = interface(IInterface);
```

C++

```
__interface IEnumerable : public IInterface;
```

File

System

Description

This is class System.IEnumerable.

38.2.3.1 IEnumerable Members

The following tables list the members exposed by IEnumerable.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	GetEnumerator (see page 3500)	This is GetEnumerator, a member of class IEnumerable.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

38.2.3.2 IEnumerable Methods

The methods of the IEnumerable class are listed here.

Public Methods

	Name	Description
	GetEnumerator (see page 3500)	This is GetEnumerator, a member of class IEnumerable.

38.2.3.2.1 IEnumerable.GetEnumerator Method

Pascal

```
function GetEnumerator: IEnumerator;
```

C++

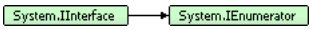
```
__fastcall IEnumerator GetEnumerator();
```

Description

This is GetEnumerator, a member of class IEnumerable.

38.2.4 IEnumerator Interface

Class Hierarchy



Pascal

```
IEnumerator = interface(IInterface);
```

C++

```
__interface IEnumerator : public IInterface;
```

File

System

Description

This is class System.IEnumerator.

38.2.4.1 IEnumerator Members

The following tables list the members exposed by IEnumerator.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	GetCurrent (see page 3501)	This is GetCurrent, a member of class IEnumerator.
	MoveNext (see page 3502)	This is MoveNext, a member of class IEnumerator.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Reset (see page 3502)	This is Reset, a member of class IEnumerator.

Public Properties

	Name	Description
R	Current (see page 3502)	This is Current, a member of class IEnumerator.

38.2.4.2 IEnumerator Methods

The methods of the IEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 3501)	This is GetCurrent, a member of class IEnumerator.
	MoveNext (see page 3502)	This is MoveNext, a member of class IEnumerator.
	Reset (see page 3502)	This is Reset, a member of class IEnumerator.

38.2.4.2.1 IEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TObject;
```

C++

```
__fastcall TObject * GetCurrent();
```

Description

This is GetCurrent, a member of class IEnumerator.

38.2.4.2.2 IEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class IEnumerator.

38.2.4.2.3 IEnumerator.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class IEnumerator.

38.2.4.3 IEnumerator Properties

The properties of the IEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 3502)	This is Current, a member of class IEnumerator.

38.2.4.3.1 IEnumerator.Current Property

Pascal

```
property Current: TObject;
```

C++

```
__property TObject * Current;
```

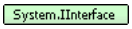
Description

This is Current, a member of class IEnumerator.

38.2.5 IInterface Interface

IInterface is the base class for all interfaces defined in Delphi code.

Class Hierarchy



Pascal

```
IInterface = interface;
```

C++

```
__interface IInterface;
```

File

System

Description

IInterface is the base interface for interfaces defined in Delphi code. When used in C++ code, IInterface is considered a child interface of IUnknown (see page 3828) and IUnknown (see page 3828) is considered the base interface for all interfaces.

Note: IInterface introduces the QueryInterface (see page 3505) method, which is useful for discovering and using other interfaces implemented by the same object. IInterface also introduces the reference counting methods, _AddRef (see page 3504) and _Release (see page 3504). The Delphi compiler automatically provides calls to these methods when interfaces are used. The System::TInterfacedObject (see page 3355) class is provided as a base class for objects that implement interfaces. Use of System::TInterfacedObject (see page 3355) is not mandatory, but will often be easier than designing an implementer class from scratch.

Note: C++ method declarations that use IInterface refer to the _di_IInterface type instead. This is a DelphiInterface (see page 3263) wrapper around the IInterface interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInterface > _di_IInterface;
```

See Also

System::TInterfacedObject (see page 3355)

IInvokable (see page 3505)

IUnknown (see page 3828)

System::DelphiInterface (see page 3263)

38.2.5.1 IInterface Members

The following tables list the members exposed by IInterface.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.

	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

38.2.5.2 IInterface Methods

The methods of the IInterface class are listed here.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

38.2.5.2.1 IInterface._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release (see page 3504) method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release (see page 3504) so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 3504)

38.2.5.2.2 IInterface._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 3504)

38.2.5.2.3 IInterface.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the object that implements this interface supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

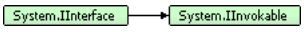
See Also

_AddRef (see page 3504)

38.2.6 IInvokable Interface

IInvokable is the base class for invokable interfaces.

Class Hierarchy



Pascal

```
IInvokable = interface(IInterface);
```

C++

```
__interface IInvokable : public IInterface;
```

File

System

Description

Use IInvokable as a base interface when defining invokable interfaces. IInvokable does not add any properties or methods to the methods it inherits from IInterface (see page 3503). It exists, rather, to ensure that the interface an all its descendants are compiled with runtime type information (RTTI). This RTTI can be used by Web Service applications or the clients of Web Service applications for translating between the code that implements a method and the symbolic encoding of that method call that is sent between client and server applications.

See Also

InvRegistry

TInvokableClassRegistry

TInvokableClass

Understanding Invokable Interfaces

38.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	__ArrayToPasStr (see page 3518)	This is function System.__ArrayToPasStr.
	__CLenToPasStr (see page 3518)	This is function System.__CLenToPasStr.
	__CToPasStr (see page 3518)	This is function System.__CToPasStr.
	__currMul (see page 3519)	This is function System::__currMul.
	__IOTest (see page 3519)	This is function System.__IOTest.
	__lldiv (see page 3519)	This is function System.__lldiv.
	__lldivo (see page 3520)	This is function System.__lldivo.
	__llmod (see page 3520)	This is function System.__llmod.
	__llmodo (see page 3520)	This is function System.__llmodo.
	__llmul (see page 3521)	This is function System.__llmul.
	__llmulo (see page 3521)	This is function System.__llmulo.
	__llshl (see page 3521)	This is function System.__llshl.
	__lludiv (see page 3522)	This is function System.__lludiv.
	__llumod (see page 3522)	This is function System.__llumod.
	__llushr (see page 3522)	This is function System.__llushr.
	__PasToCStr (see page 3523)	This is function System.__PasToCStr.
	_AbstractError (see page 3523)	This is function System._AbstractError.
	_AddRef (see page 3523)	This is function System._AddRef.
	_AddRefArray (see page 3524)	This is function System._AddRefArray.
	_AddRefRecord (see page 3524)	This is function System._AddRefRecord.
	_AfterConstruction (see page 3524)	This is function System._AfterConstruction.
	_Append (see page 3525)	This is function System._Append.
	_AsClass (see page 3525)	This is function System._AsClass.
	_Assert (see page 3525)	This is function System._Assert.
	_Assign (see page 3526)	This is function System._Assign.
	_AStrCmp (see page 3526)	This is function System._AStrCmp.
	_BeforeDestruction (see page 3526)	This is function System._BeforeDestruction.
	_BlockRead (see page 3527)	This is function System._BlockRead.
	_BlockWrite (see page 3527)	This is function System._BlockWrite.
	_BoundErr (see page 3527)	This is function System._BoundErr.
	_CallDynaClass (see page 3528)	This is function System._CallDynaClass.
	_CallDynaInst (see page 3528)	This is function System._CallDynaInst.
	_CheckAutoResult (see page 3528)	This is function System._CheckAutoResult.
	_ClassCreate (see page 3529)	This is function System._ClassCreate.
	_ClassDestroy (see page 3529)	This is function System._ClassDestroy.
	_Close (see page 3529)	This is function System._Close.
	_Copy (see page 3530)	This is function System._Copy.
	_CopyArray (see page 3530)	This is function System._CopyArray.
	_CopyObject (see page 3530)	This is function System._CopyObject.
	_CopyRecord (see page 3531)	This is function System._CopyRecord.
	_CVR_PROBE (see page 3531)	This is function System._CVR_PROBE.
	_CVR_STMTPROBE (see page 3531)	This is function System._CVR_STMTPROBE.
	_Delete (see page 3532)	This is function System._Delete.
	_Dispose (see page 3532)	This is function System._Dispose.

	_DoneExcept (see page 3532)	This is function System._DoneExcept.
	_DynArrayAddRef (see page 3533)	This is function System._DynArrayAddRef.
	_DynArrayAsg (see page 3533)	This is function System._DynArrayAsg.
	_DynArrayClear (see page 3533)	This is function System._DynArrayClear.
	_DynArrayCopy (see page 3534)	This is function System._DynArrayCopy.
	_DynArrayCopyRange (see page 3534)	This is function System._DynArrayCopyRange.
	_DynArrayHigh (see page 3534)	This is function System._DynArrayHigh.
	_DynArrayLength (see page 3535)	This is function System._DynArrayLength.
	_DynArraySetLength (see page 3535)	This is function System._DynArraySetLength.
	_EnsureAnsiString (see page 3535)	This is function System._EnsureAnsiString.
	_EnsureUnicodeString (see page 3536)	This is function System._EnsureUnicodeString.
	_EofFile (see page 3536)	This is function System._EofFile.
	_EofText (see page 3536)	This is function System._EofText.
	_Eoln (see page 3537)	This is function System._Eoln.
	_Erase (see page 3537)	This is function System._Erase.
	_Ext2Real (see page 3537)	This is function System._Ext2Real.
	_Fail (see page 3538)	This is function System._Fail.
	_FilePos (see page 3538)	This is function System._FilePos.
	_FileSize (see page 3538)	This is function System._FileSize.
	_FillChar (see page 3539)	This is function System._FillChar.
	_Finalize (see page 3539)	This is function System._Finalize.
	_FinalizeArray (see page 3539)	This is function System._FinalizeArray.
	_FinalizeRecord (see page 3540)	This is function System._FinalizeRecord.
	_FindDynaClass (see page 3540)	This is function System._FindDynaClass.
	_FindDynaInst (see page 3540)	This is function System._FindDynaInst.
	_Flush (see page 3541)	This is function System._Flush.
	_FreeMem (see page 3541)	This is function System._FreeMem.
	_FSafeDivide (see page 3541)	This is function System._FSafeDivide.
	_FSafeDivideR (see page 3542)	This is function System._FSafeDivideR.
	_GetHelperIntf (see page 3542)	This is function System._GetHelperIntf.
	_GetMem (see page 3542)	This is function System._GetMem.
	_Halt (see page 3543)	This is function System._Halt.
	_Halt0 (see page 3543)	This is function System._Halt0.
	_HandleAnyException (see page 3543)	This is function System._HandleAnyException.
	_HandleAutoException (see page 3544)	This is function System._HandleAutoException.
	_HandleFinally (see page 3544)	This is function System._HandleFinally.
	_HandleOnException (see page 3544)	This is function System._HandleOnException.
	_Initialize (see page 3545)	This is function System._Initialize.
	_InitializeArray (see page 3545)	This is function System._InitializeArray.
	_InitializeRecord (see page 3545)	This is function System._InitializeRecord.
	_InitImports (see page 3546)	This is function System._InitImports.
	_InitResStringImports (see page 3546)	This is function System._InitResStringImports.
	_InitResStrings (see page 3546)	This is function System._InitResStrings.
	_InitUnicodeStrings (see page 3547)	This is function System._InitUnicodeStrings.
	_InitWideStrings (see page 3547)	This is function System._InitWideStrings.
	_Insert (see page 3547)	This is function System._Insert.
	_InternalLStrFromUStr (see page 3548)	This is function System._InternalLStrFromUStr.
	_InternalUStrFromLStr (see page 3548)	This is function System._InternalUStrFromLStr.
	_IntfAddRef (see page 3548)	This is function System._IntfAddRef.
	_IntfCast (see page 3549)	This is function System._IntfCast.
	_IntfClear (see page 3549)	This is function System._IntfClear.
	_IntfCopy (see page 3549)	This is function System._IntfCopy.
	_IntfDispCall (see page 3550)	This is function System._IntfDispCall.
	_IntfVarCall (see page 3550)	This is function System._IntfVarCall.

	_IntOver (see page 3550)	This is function System._IntOver.
	_IsClass (see page 3551)	This is function System._IsClass.
	_LGetDir (see page 3551)	This is function System._LGetDir.
	_LStrAddRef (see page 3551)	This is function System._LStrAddRef.
	_LStrArrayClr (see page 3552)	This is function System._LStrArrayClr.
	_LStrAsg (see page 3552)	This is function System._LStrAsg.
	_LStrCat (see page 3552)	This is function System._LStrCat.
	_LStrCat3 (see page 3553)	This is function System._LStrCat3.
	_LStrCatN (see page 3553)	This is function System._LStrCatN.
	_LStrClr (see page 3553)	This is function System._LStrClr.
	_LStrCmp (see page 3554)	This is function System._LStrCmp.
	_LStrCopy (see page 3554)	This is function System._LStrCopy.
	_LStrDelete (see page 3554)	This is function System._LStrDelete.
	_LStrEqual (see page 3555)	This is function System._LStrEqual.
	_LStrFromArray (see page 3555)	This is function System._LStrFromArray.
	_LStrFromChar (see page 3555)	This is function System._LStrFromChar.
	_LStrFromPChar (see page 3556)	This is function System._LStrFromPChar.
	_LStrFromPCharLen (see page 3556)	This is function System._LStrFromPCharLen.
	_LStrFromPWChar (see page 3556)	This is function System._LStrFromPWChar.
	_LStrFromPWCharLen (see page 3557)	This is function System._LStrFromPWCharLen.
	_LStrFromString (see page 3557)	This is function System._LStrFromString.
	_LStrFromUStr (see page 3557)	This is function System._LStrFromUStr.
	_LStrFromWArray (see page 3558)	This is function System._LStrFromWArray.
	_LStrFromWChar (see page 3558)	This is function System._LStrFromWChar.
	_LStrFromWStr (see page 3558)	This is function System._LStrFromWStr.
	_LStrInsert (see page 3559)	This is function System._LStrInsert.
	_LStrLAsg (see page 3559)	This is function System._LStrLAsg.
	_LStrLen (see page 3559)	This is function System._LStrLen.
	_LStrPos (see page 3560)	This is function System._LStrPos.
	_LStrSetLength (see page 3560)	This is function System._LStrSetLength.
	_LStrToPChar (see page 3560)	This is function System._LStrToPChar.
	_LStrToString (see page 3561)	This is function System._LStrToString.
	_New (see page 3561)	This is function System._New.
	_NewAnsiString (see page 3561)	This is function System._NewAnsiString.
	_NewUnicodeString (see page 3562)	This is function System._NewUnicodeString.
	_NewWideString (see page 3562)	This is function System._NewWideString.
	_ObjCopy (see page 3562)	This is function System._ObjCopy.
	_ObjSetup (see page 3563)	This is function System._ObjSetup.
	_PackageLoad (see page 3563)	This is function System._PackageLoad.
	_PackageUnload (see page 3563)	This is function System._PackageUnload.
	_PCharLen (see page 3564)	This is function System._PCharLen.
	_Pos (see page 3564)	This is function System._Pos.
	_Pow10 (see page 3564)	This is function System._Pow10.
	_PStrCat (see page 3565)	This is function System._PStrCat.
	_PStrCmp (see page 3565)	This is function System._PStrCmp.
	_PStrCpy (see page 3565)	This is function System._PStrCpy.
	_PStrNCat (see page 3566)	This is function System._PStrNCat.
	_PStrNCpy (see page 3566)	This is function System._PStrNCpy.
	_PWCharLen (see page 3566)	This is function System._PWCharLen.
	_RaiseAgain (see page 3567)	This is function System._RaiseAgain.
	_RaiseExcept (see page 3567)	This is function System._RaiseExcept.
	_ReadChar (see page 3567)	This is function System._ReadChar.
	_ReadCString (see page 3568)	This is function System._ReadCString.
	_ReadExt (see page 3568)	This is function System._ReadExt.

	_ReadInt64 (see page 3568)	This is function System._ReadInt64.
	_ReadLn (see page 3569)	This is function System._ReadLn.
	_ReadLong (see page 3569)	This is function System._ReadLong.
	_ReadLString (see page 3569)	This is function System._ReadLString.
	_ReadRec (see page 3570)	This is function System._ReadRec.
	_ReadString (see page 3570)	This is function System._ReadString.
	_ReadUString (see page 3570)	This is function System._ReadUString.
	_ReadWChar (see page 3571)	This is function System._ReadWChar.
	_ReadWCString (see page 3571)	This is function System._ReadWCString.
	_ReadWString (see page 3571)	This is function System._ReadWString.
	_Real2Ext (see page 3572)	This is function System._Real2Ext.
	_ReallocMem (see page 3572)	This is function System._ReallocMem.
	_Rename (see page 3572)	This is function System._Rename.
	_ResetFile (see page 3573)	This is function System._ResetFile.
	_ResetText (see page 3573)	This is function System._ResetText.
	_RewritFile (see page 3573)	This is function System._RewritFile.
	_RewritText (see page 3574)	This is function System._RewritText.
	_ROUND (see page 3574)	This is function System._ROUND.
	_roundToInt64 (see page 3574)	This is function System::_roundToInt64.
	_Run0Error (see page 3575)	This is function System._Run0Error.
	_RunError (see page 3575)	This is function System._RunError.
	_Seek (see page 3575)	This is function System._Seek.
	_SeekEof (see page 3576)	This is function System._SeekEof.
	_SeekEoln (see page 3576)	This is function System._SeekEoln.
	_SetElem (see page 3576)	This is function System._SetElem.
	_SetEq (see page 3577)	This is function System._SetEq.
	_SetExpand (see page 3577)	This is function System._SetExpand.
	_SetIntersect (see page 3577)	This is function System._SetIntersect.
	_SetIntersect3 (see page 3578)	This is function System._SetIntersect3.
	_SetLe (see page 3578)	This is function System._SetLe.
	_SetLength (see page 3578)	This is function System._SetLength.
	_SetRange (see page 3579)	This is function System._SetRange.
	_SetString (see page 3579)	This is function System._SetString.
	_SetSub (see page 3579)	This is function System._SetSub.
	_SetSub3 (see page 3580)	This is function System._SetSub3.
	_SetTextBuf (see page 3580)	This is function System._SetTextBuf.
	_SetUnion (see page 3580)	This is function System._SetUnion.
	_SetUnion3 (see page 3581)	This is function System._SetUnion3.
	_SGetDir (see page 3581)	This is function System._SGetDir.
	_StartExe (see page 3581)	This is function System._StartExe.
	_StartLib (see page 3582)	This is function System._StartLib.
	_Str0Ext (see page 3582)	This is function System._Str0Ext.
	_Str0Int64 (see page 3582)	This is function System._Str0Int64.
	_Str0Long (see page 3583)	This is function System._Str0Long.
	_Str0UInt64 (see page 3583)	This is function System._Str0UInt64.
	_Str1Ext (see page 3583)	This is function System._Str1Ext.
	_Str2Ext (see page 3584)	This is function System._Str2Ext.
	_StrInt64 (see page 3584)	This is function System._StrInt64.
	_StrLong (see page 3584)	This is function System._StrLong.
	_StrUInt64 (see page 3585)	This is function System._StrUInt64.
	_TRUNC (see page 3585)	This is function System._TRUNC.
	_Truncate (see page 3585)	This is function System._Truncate.
	_TryFinallyExit (see page 3586)	This is function System._TryFinallyExit.
	_UGetDir (see page 3586)	This is function System._UGetDir.

	_UniqueStringA (see page 3586)	This is function System._UniqueStringA.
	_UniqueStringU (see page 3587)	This is function System._UniqueStringU.
	_UniqueStringW (see page 3587)	This is function System._UniqueStringW.
	_UStrAddRef (see page 3587)	This is function System._UStrAddRef.
	_UStrArrayClr (see page 3588)	This is function System._UStrArrayClr.
	_UStrAsg (see page 3588)	This is function System._UStrAsg.
	_UStrCat (see page 3588)	This is function System._UStrCat.
	_UStrCat3 (see page 3589)	This is function System._UStrCat3.
	_UStrCatN (see page 3589)	This is function System._UStrCatN.
	_UStrClr (see page 3589)	This is function System._UStrClr.
	_UStrCmp (see page 3590)	This is function System._UStrCmp.
	_UStrCopy (see page 3590)	This is function System._UStrCopy.
	_UStrDelete (see page 3590)	This is function System._UStrDelete.
	_UStrEqual (see page 3591)	This is function System._UStrEqual.
	_UStrFromArray (see page 3591)	This is function System._UStrFromArray.
	_UStrFromChar (see page 3591)	This is function System._UStrFromChar.
	_UStrFromLStr (see page 3592)	This is function System._UStrFromLStr.
	_UStrFromPChar (see page 3592)	This is function System._UStrFromPChar.
	_UStrFromPCharLen (see page 3592)	This is function System._UStrFromPCharLen.
	_UStrFromPWChar (see page 3593)	This is function System._UStrFromPWChar.
	_UStrFromPWCharLen (see page 3593)	This is function System._UStrFromPWCharLen.
	_UStrFromString (see page 3593)	This is function System._UStrFromString.
	_UStrFromWArray (see page 3594)	This is function System._UStrFromWArray.
	_UStrFromWChar (see page 3594)	This is function System._UStrFromWChar.
	_UStrFromWStr (see page 3594)	This is function System._UStrFromWStr.
	_UStrInsert (see page 3595)	This is function System._UStrInsert.
	_UStrLAsg (see page 3595)	This is function System._UStrLAsg.
	_UStrLen (see page 3595)	This is function System._UStrLen.
	_UStrSetLength (see page 3596)	This is function System._UStrSetLength.
	_UStrToPWChar (see page 3596)	This is function System._UStrToPWChar.
	_UStrToString (see page 3596)	This is function System._UStrToString.
	_ValExt (see page 3597)	This is function System._ValExt.
	_ValInt64 (see page 3597)	This is function System._ValInt64.
	_ValLong (see page 3597)	This is function System._ValLong.
	_WCharToString (see page 3598)	This is function System._WCharToString.
	_WGetDir (see page 3598)	This is function System._WGetDir.
	_Write0Bool (see page 3598)	This is function System._Write0Bool.
	_Write0Char (see page 3599)	This is function System._Write0Char.
	_Write0CString (see page 3599)	This is function System._Write0CString.
	_Write0Ext (see page 3599)	This is function System._Write0Ext.
	_Write0Int64 (see page 3600)	This is function System._Write0Int64.
	_Write0Long (see page 3600)	This is function System._Write0Long.
	_Write0LString (see page 3600)	This is function System._Write0LString.
	_Write0String (see page 3601)	This is function System._Write0String.
	_Write0UInt64 (see page 3601)	This is function System._Write0UInt64.
	_Write0UString (see page 3601)	This is function System._Write0UString.
	_Write0Variant (see page 3602)	This is function System._Write0Variant.
	_Write0WChar (see page 3602)	This is function System._Write0WChar.
	_Write0WCString (see page 3602)	This is function System._Write0WCString.
	_Write0WString (see page 3603)	This is function System._Write0WString.
	_Write1Ext (see page 3603)	This is function System._Write1Ext.
	_Write2Ext (see page 3603)	This is function System._Write2Ext.
	_WriteBool (see page 3604)	This is function System._WriteBool.
	_WriteChar (see page 3604)	This is function System._WriteChar.

	_WriteCString (see page 3604)	This is function System._WriteCString.
	_WriteInt64 (see page 3605)	This is function System._WriteInt64.
	_WriteLn (see page 3605)	This is function System._WriteLn.
	_WriteLong (see page 3605)	This is function System._WriteLong.
	_WriteLString (see page 3606)	This is function System._WriteLString.
	_WriteRec (see page 3606)	This is function System._WriteRec.
	_WriteString (see page 3606)	This is function System._WriteString.
	_WriteUInt64 (see page 3607)	This is function System._WriteUInt64.
	_WriteUString (see page 3607)	This is function System._WriteUString.
	_WriteVariant (see page 3607)	This is function System._WriteVariant.
	_WriteWChar (see page 3608)	This is function System._WriteWChar.
	_WriteWCString (see page 3608)	This is function System._WriteWCString.
	_WriteWString (see page 3608)	This is function System._WriteWString.
	_WStrAddRef (see page 3609)	This is function System._WStrAddRef.
	_WStrArrayClr (see page 3609)	This is function System._WStrArrayClr.
	_WStrAsg (see page 3609)	This is function System._WStrAsg.
	_WStrCat (see page 3610)	This is function System._WStrCat.
	_WStrCat3 (see page 3610)	This is function System._WStrCat3.
	_WStrCatN (see page 3610)	This is function System._WStrCatN.
	_WStrClr (see page 3611)	This is function System._WStrClr.
	_WStrCmp (see page 3611)	This is function System._WStrCmp.
	_WStrCopy (see page 3611)	This is function System._WStrCopy.
	_WStrDelete (see page 3612)	This is function System._WStrDelete.
	_WStrEqual (see page 3612)	This is function System._WStrEqual.
	_WStrFromArray (see page 3612)	This is function System._WStrFromArray.
	_WStrFromChar (see page 3613)	This is function System._WStrFromChar.
	_WStrFromLStr (see page 3613)	This is function System._WStrFromLStr.
	_WStrFromPChar (see page 3613)	This is function System._WStrFromPChar.
	_WStrFromPCharLen (see page 3614)	This is function System._WStrFromPCharLen.
	_WStrFromPWChar (see page 3614)	This is function System._WStrFromPWChar.
	_WStrFromPWCharLen (see page 3614)	This is function System._WStrFromPWCharLen.
	_WStrFromString (see page 3615)	This is function System._WStrFromString.
	_WStrFromUStr (see page 3615)	This is function System._WStrFromUStr.
	_WStrFromWArray (see page 3615)	This is function System._WStrFromWArray.
	_WStrFromWChar (see page 3616)	This is function System._WStrFromWChar.
	_WStrInsert (see page 3616)	This is function System._WStrInsert.
	_WStrLAsg (see page 3616)	This is function System._WStrLAsg.
	_WStrLCmp (see page 3617)	This is function System._WStrLCmp.
	_WStrLen (see page 3617)	This is function System._WStrLen.
	_WStrSetLength (see page 3617)	This is function System._WStrSetLength.
	_WStrToPWChar (see page 3618)	This is function System._WStrToPWChar.
	_WStrToString (see page 3618)	This is function System._WStrToString.
	+ (see page 3618)	This is function System::+.
	+ (see page 3619)	This is function System::+.
	Abs (see page 3619)	Returns an absolute value.
	AcquireExceptionObject (see page 3619)	Retains access to exception object.
	AddModuleUnloadProc (see page 3620)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 3620)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 3621)	Returns a pointer to a specified object.
	AllocMem (see page 3622)	AllocMem allocates a memory block and initializes each byte to zero.
	AnsiToUtf8 (see page 3624)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 3624)	Prepares an existing file for adding text to the end.
	ArcTan (see page 3625)	Calculates the arctangent of a given number.

	Assert (see page 3627)	Tests whether a Boolean expression is true.
	Assert (see page 3627)	Tests whether a Boolean expression is true.
	Assigned (see page 3628)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 3629)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 3631)	Attempts to use a shared memory manager.
	BeginThread (see page 3632)	Spawns a separate thread of execution.
	BlockRead (see page 3633)	Reads one or more records from an open file into a variable.
	BlockRead (see page 3633)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 3634)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 3634)	Writes one or more records from a variable to an open file.
	Break (see page 3636)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 3638)	Changes the current directory.
	ChDir (see page 3638)	Changes the current directory.
	CheckSafecallResult (see page 3640)	This is function System::CheckSafecallResult.
	Chr (see page 3640)	Returns the character for a specified ASCII value.
	Close (see page 3643)	Terminates the association between a file variable and an external file.
	CloseFile (see page 3643)	Terminates the association between file variable and an external disk file.
	CompareTo (see page 3645)	This is function System.CompareTo.
	CompToCurrency (see page 3645)	Converts a Comp value to a Currency (see page 3242) value.
	CompToDouble (see page 3646)	Converts a Comp value to a double value.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Concat (see page 3646)	Concatenates two or more strings into one.
	Continue (see page 3647)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 3648)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 3650)	Calculates the cosine of an angle.
	CurrencyToComp (see page 3651)	Converts a Currency (see page 3242) value to a Comp value.
	Dec (see page 3652)	Decrements a variable by 1 or N.
	Dec (see page 3652)	Decrements a variable by 1 or N.
	Delete (see page 3653)	Removes a substring from a string.
	Dispose (see page 3654)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 3656)	Converts a double value to a Comp.
	DynArrayClear (see page 3656)	This is function System.DynArrayClear.
	DynArrayDim (see page 3657)	This is function System.DynArrayDim.
	DynArraySetLength (see page 3657)	This is function System.DynArraySetLength.
	EndThread (see page 3657)	Terminates the execution of a thread.
	EnumModules (see page 3658)	Executes a callback for all the modules in an application.
	EnumModules (see page 3658)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 3658)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 3658)	Executes a callback for all resource modules in an application.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eof (see page 3659)	Tests whether the file position is at the end of a file.
	Eoln (see page 3660)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 3660)	Tests whether the file pointer is at the end of a line.
	Equals (see page 3661)	This is function System.Equals.
	Erase (see page 3661)	Deletes an external file.
	Error (see page 3662)	Error is used to generate a run-time exception.
	ExceptAddr (see page 3663)	Returns the address at which the current exception was raised.
	ExceptObject (see page 3664)	Returns current exception object.
	Exclude (see page 3664)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 3665)	Exits from the current procedure.
	Exp (see page 3667)	Returns the exponential of X.

	Exp (see page 3667)	Returns the exponential of X.
	FilePos (see page 3668)	Returns the current file position.
	FileSize (see page 3669)	Returns the number of records in a file.
	FillChar (see page 3670)	Fills contiguous bytes with a specified value.
	Finalize (see page 3672)	Uninitializes a dynamically allocated variable.
	Finalize (see page 3672)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 3672)	Returns the instance handle for the module in which a class type is defined.
	FindDynaInst (see page 3673)	This is function System::FindDynaInst.
	FindHInstance (see page 3673)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 3674)	Returns the instance handle of the resource module associated with a specified HInstance.
	Flush (see page 3674)	Empties the buffer of a text file opened for output.
	FPower10 (see page 3675)	This is function System.FPower10.
	Frac (see page 3675)	Returns the fractional part of a real number.
	FreeMem (see page 3676)	FreeMem frees a memory block.
	FreeMem (see page 3676)	FreeMem frees a memory block.
	FreeMemory (see page 3678)	FreeMemory frees a memory block.
	Get8087CW (see page 3679)	Returns the value of the 8087 code word.
	GetCurrent (see page 3679)	This is function System.GetCurrent.
	GetDir (see page 3679)	Returns the current directory.
	GetEnumerator (see page 3681)	This is function System.GetEnumerator.
	GetHeapStatus (see page 3681)	Returns the current status of the memory manager.
	GetLastError (see page 3681)	Returns the last error reported by an operating system API call.
	GetMem (see page 3683)	GetMem allocates a memory block.
	GetMemory (see page 3684)	GetMemory allocates a memory block.
	GetMemoryManager (see page 3686)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 3686)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 3690)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 3691)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 3691)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 3692)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 3692)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 3693)	Initiates abnormal termination of a program.
	Halt (see page 3693)	Initiates abnormal termination of a program.
	Hi (see page 3694)	Returns the high-order byte of X as an unsigned value.
	High (see page 3694)	Returns the highest value in the range of an argument.
	Inc (see page 3696)	Increments an ordinal value by one or N.
	Inc (see page 3696)	Increments an ordinal value by one or N.
	Include (see page 3697)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 3698)	Initializes a dynamically allocated variable.
	Initialize (see page 3698)	Initializes a dynamically allocated variable.
	Insert (see page 3698)	Inserts a substring into a string beginning at a specified point.
	Int (see page 3699)	Returns the integer part of a real number.
	IOResult (see page 3700)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 3701)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3751) procedure.
	IsVariantManagerSet (see page 3701)	Indicates whether variant support is implemented in an application.
	Length (see page 3702)	Returns the number of characters in a string or elements in an array.
	Ln (see page 3703)	Returns the natural log of a real expression.
	Lo (see page 3704)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 3705)	This is function System.LoadResourceModule.
	LoadResourceString (see page 3705)	This is function System::LoadResourceString.
	LoadResString (see page 3705)	This is function System.LoadResString.
	Low (see page 3706)	Returns the lowest value in a range.
	Mark (see page 3707)	This is function System.Mark.

	MemoryBarrier (see page 3707)	This is function System.MemoryBarrier.
	MkDir (see page 3708)	Creates a new subdirectory.
	MkDir (see page 3708)	Creates a new subdirectory.
	MonitorEnter (see page 3709)	This is function System.MonitorEnter.
	MonitorExit (see page 3709)	This is function System.MonitorExit.
	MonitorPulse (see page 3710)	This is function System.MonitorPulse.
	MonitorPulseAll (see page 3710)	This is function System.MonitorPulseAll.
	MonitorTryEnter (see page 3710)	This is function System.MonitorTryEnter.
	MonitorWait (see page 3711)	This is function System.MonitorWait.
	Move (see page 3711)	Copies bytes from a source to a destination.
	MoveChars (see page 3712)	This is function System.MoveChars.
	New (see page 3712)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 3714)	Returns true if argument is an odd number.
	OleStrToString (see page 3715)	Copies data received from a COM interface to a string.
	OleStrToStrVar (see page 3715)	Copies a string from its COM representation to an existing Delphi string.
	OleStrToStrVar (see page 3715)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 3716)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 3716)	Returns the number of parameters passed on the command line.
	ParamStr (see page 3718)	Returns a specified parameter from the command-line.
	Pi (see page 3719)	Returns 3.1415926535897932385.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3720)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 3721)	Returns the predecessor of the argument.
	Ptr (see page 3723)	Converts a specified address to a pointer.
	PUCS4Chars (see page 3724)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 3724)	This is function System.RaiseList.
	Random (see page 3725)	Generates random numbers within a specified range.
	Random (see page 3725)	Generates random numbers within a specified range.
	Randomize (see page 3726)	Initializes the random number generator with a random value.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.
	Read (see page 3727)	Read reads data from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReadLn (see page 3728)	Reads a line of text from a file.
	ReallocMem (see page 3729)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 3730)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 3731)	Registers an expected memory leak.
	RegisterModule (see page 3731)	This is function System.RegisterModule.
	Release (see page 3732)	This is function System.Release.
	ReleaseExceptionObject (see page 3732)	Releases exception object acquired by AcquireExceptionObject (see page 3619).
	RemoveModuleUnloadProc (see page 3732)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 3732)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 3733)	Changes the name of an external file.
	Rename (see page 3733)	Changes the name of an external file.
	Reset (see page 3734)	Opens an existing file.
	Reset (see page 3734)	Opens an existing file.

	Reset (see page 3734)	Opens an existing file.
	Reset (see page 3734)	Opens an existing file.
	Rewrite (see page 3736)	Creates a new file and opens it.
	Rewrite (see page 3736)	Creates a new file and opens it.
	RmDir (see page 3738)	Deletes an empty subdirectory.
	RmDir (see page 3738)	Deletes an empty subdirectory.
	Round (see page 3739)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 3740)	Stops execution and generates a runtime error.
	RunError (see page 3740)	Stops execution and generates a runtime error.
	Seek (see page 3741)	Moves the current position of a file to a specified component.
	SeekEof (see page 3742)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 3742)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 3743)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 3743)	Returns the end-of-line status of a file, ignoring whitespace.
	Set8087CW (see page 3744)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3901) declared in the system unit.
	SetCodePage (see page 3746)	This is function System.SetCodePage.
	SetInOutRes (see page 3746)	This is function System.SetInOutRes.
	SetLastError (see page 3747)	SetLastError sets the last API error code.
	SetLength (see page 3748)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 3750)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 3751)	Sets entry points of the memory manager.
	SetMemoryManager (see page 3751)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 3756)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 3756)	SetMultiByteConversionCodePage overrides the default multi byte code conversion page.
	SetRaiseList (see page 3757)	This is function System.SetRaiseList.
	SetString (see page 3757)	Sets the contents and length of the given string.
	SetTextBuf (see page 3758)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 3758)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 3759)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 3759)	Shares the module's memory manager.
	Sin (see page 3760)	Returns the sine of the angle in radians.
	SizeOf (see page 3761)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 3762)	Returns a sub-section of an array.
	Sqr (see page 3762)	Returns the square of a number.
	Sqr (see page 3762)	Returns the square of a number.
	Sqrt (see page 3763)	Returns the square root of X.
	Str (see page 3764)	Formats a string and returns it to a variable.
	Str (see page 3764)	Formats a string and returns it to a variable.
	StringCodePage (see page 3765)	This is function System.StringCodePage.
	StringCodePage (see page 3765)	This is function System.StringCodePage.
	StringElementSize (see page 3765)	This is function System.StringElementSize.
	StringElementSize (see page 3766)	This is function System.StringElementSize.
	StringOfChar (see page 3766)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 3766)	Returns a string with a specified number of repeating characters.
	StringRefCount (see page 3766)	This is function System.StringRefCount.
	StringRefCount (see page 3767)	This is function System.StringRefCount.
	StringToOleStr (see page 3767)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToOleStr (see page 3767)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 3768)	Returns a UNICODE string from an AnsiString.
	Succ (see page 3768)	Returns the successor of an argument.
	Swap (see page 3769)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 3769)	This is function System.SysAllocMem.
	SysFreeMem (see page 3770)	Frees the memory pointed to by a specified pointer.

	SysGetMem (see page 3770)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 3771)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 3772)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 3772)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 3772)	This is function System.TextStart.
	Trunc (see page 3773)	Truncates a real number to an integer.
	Truncate (see page 3774)	Deletes all the records after the current file position.
	UCS4StringToUnicodeString (see page 3775)	This is function System.UCS4StringToUnicodeString.
	UCS4StringToWideString (see page 3775)	Converts a string of 4-byte Unicode characters into a WideString (see page 3476) value.
	UnicodeStringToUCS4String (see page 3775)	This is function System.UnicodeStringToUCS4String.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3776)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3777)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 3777)	Unregisters an expected memory leak.
	UnregisterModule (see page 3778)	This is function System.UnregisterModule.
	UpCase (see page 3778)	Converts a character to uppercase.
	UpCase (see page 3778)	Converts a character to uppercase.
	UTF8Decode (see page 3779)	Converts a UTF8 string to a Unicode string (WideString (see page 3476)).
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8Encode (see page 3780)	Converts a WideString (see page 3476) value to a UTF8 string.
	UTF8EncodeToShortString (see page 3780)	This is function System.UTF8EncodeToShortString.
	UTF8EncodeToShortString (see page 3780)	This is function System.UTF8EncodeToShortString.
	UTF8EncodeToShortString (see page 3781)	This is function System.UTF8EncodeToShortString.
	Utf8ToAnsi (see page 3781)	Converts a UTF8 string to a string of Ansi characters.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3782)	This is function System.UTF8ToString.
	UTF8ToString (see page 3783)	This is function System.UTF8ToString.
	Utf8ToUnicode (see page 3783)	Converts a string of UTF-8 characters to a Unicode string.
	Utf8ToUnicode (see page 3783)	Converts a string of UTF-8 characters to a Unicode string.
	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToUnicodeString (see page 3784)	This is function System.UTF8ToUnicodeString.
	UTF8ToWideString (see page 3785)	This is function System.UTF8ToWideString.
	Val (see page 3785)	Converts a string to a numeric representation.
	VarArrayRedim (see page 3786)	Resizes a Variant (see page 3430) array.
	VarClear (see page 3786)	Empties a Variant (see page 3430), so that it is unassigned.
	WideCharLenToString (see page 3787)	Converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 3788)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 3788)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharToString (see page 3788)	Converts null-terminated Unicode string to single or multi byte character data.
	WideCharToStrVar (see page 3788)	Converts Unicode string to a single or multi byte character data.
	WideStringToUCS4String (see page 3789)	Converts a WideString (see page 3476) value to a string of 4-byte Unicode characters.
	Write (see page 3789)	Writes to either a typed file or a text file
	Write (see page 3789)	Writes to either a typed file or a text file
	Write (see page 3789)	Writes to either a typed file or a text file

	Write (see page 3789)	Writes to either a typed file or a text file
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3792)	Writes to a text file and adds an end-of-line marker.
	YieldProcessor (see page 3793)	This is function System.YieldProcessor.

38.3.1 System.__ArrayToPasStr Function

Pascal

```
procedure __ArrayToPasStr(Dest: PShortString; const Source: PAnsiChar; Len: Integer);
```

C++

```
__ArrayToPasStr(PShortString Dest, const PAnsiChar Source, int Len);
```

File

System

Description

This is function System.__ArrayToPasStr.

38.3.2 System.__CLenToPasStr Function

Pascal

```
procedure __CLenToPasStr(Dest: PShortString; const Source: PAnsiChar; MaxLen: Integer);
```

C++

```
__CLenToPasStr(PShortString Dest, const PAnsiChar Source, int MaxLen);
```

File

System

Description

This is function System.__CLenToPasStr.

38.3.3 System.__CToPasStr Function

Pascal

```
procedure __CToPasStr(Dest: PShortString; const Source: PAnsiChar);
```

C++

```
__CToPasStr(PShortString Dest, const PAnsiChar Source);
```

File

System

Description

This is function System.__CToPasStr.

38.3.4 System::__currMul Function

Pascal

```
procedure __currMul(const c1: System::Currency; var c2: System::Currency);
```

C++

```
__fastcall void __currMul(const System::Currency c1, System::Currency & c2);
```

File

syscurr.h

Description

This is function System::__currMul.

38.3.5 System.__IOTest Function

Pascal

```
procedure __IOTest;
```

C++

```
__IOTest();
```

File

System

Description

This is function System.__IOTest.

38.3.6 System.__lldiv Function

Pascal

```
procedure __lldiv;
```

C++

```
__lldiv();
```

File

System

Description

This is function System.__lldiv.

38.3.7 System.__lldivo Function

Pascal

```
procedure __lldivo;
```

C++

```
__lldivo();
```

File

System

Description

This is function System.__lldivo.

38.3.8 System.__llmod Function

Pascal

```
procedure __llmod;
```

C++

```
__llmod();
```

File

System

Description

This is function System.__llmod.

38.3.9 System.__llmodo Function

Pascal

```
procedure __llmodo;
```

C++

```
__llmodo();
```

File

System

Description

This is function System.__llmodo.

38.3.10 System.__llmul Function

Pascal

```
procedure __llmul;
```

C++

```
__llmul();
```

File

System

Description

This is function System.__llmul.

38.3.11 System.__llmulo Function

Pascal

```
procedure __llmulo;
```

C++

```
__llmulo();
```

File

System

Description

This is function System.__llmulo.

38.3.12 System.__llshl Function

Pascal

```
procedure __llshl;
```

C++

```
__llshl();
```

File

System

Description

This is function System.__llshl.

38.3.13 System.__lludiv Function

Pascal

```
procedure __lludiv;
```

C++

```
__lludiv();
```

File

System

Description

This is function System.__lludiv.

38.3.14 System.__llumod Function

Pascal

```
procedure __llumod;
```

C++

```
__llumod();
```

File

System

Description

This is function System.__llumod.

38.3.15 System.__llushr Function

Pascal

```
procedure __llushr;
```

C++

```
__llushr();
```

File

System

Description

This is function System.__llushr.

38.3.16 System.__PasToCStr Function

Pascal

```
procedure __PasToCStr(const Source: PShortString; const Dest: PAnsiChar);
```

C++

```
__PasToCStr(const PShortString Source, const PAnsiChar Dest);
```

File

System

Description

This is function System.__PasToCStr.

38.3.17 System._AbstractError Function

Pascal

```
procedure _AbstractError;
```

C++

```
_AbstractError();
```

File

System

Description

This is function System._AbstractError.

38.3.18 System._AddRef Function

Pascal

```
procedure _AddRef;
```

C++

```
_AddRef();
```

File

System

Description

This is function System._AddRef.

38.3.19 System._AddRefArray Function

Pascal

```
procedure _AddRefArray;
```

C++

```
_AddRefArray();
```

File

System

Description

This is function System._AddRefArray.

38.3.20 System._AddRefRecord Function

Pascal

```
procedure _AddRefRecord;
```

C++

```
_AddRefRecord();
```

File

System

Description

This is function System._AddRefRecord.

38.3.21 System._AfterConstruction Function

Pascal

```
function _AfterConstruction(Instance: TObject): TObject;
```

C++

```
TObject * _AfterConstruction(TObject * Instance);
```

File

System

Description

This is function System._AfterConstruction.

38.3.22 System._Append Function

Pascal

```
function _Append(var t: TTextRec): Integer;
```

C++

```
int _Append(TTextRec t);
```

File

System

Description

This is function System._Append.

38.3.23 System._AsClass Function

Pascal

```
function _AsClass(Child: TObject; Parent: TClass): TObject;
```

C++

```
TObject * _AsClass(TObject * Child, TClass Parent);
```

File

System

Description

This is function System._AsClass.

38.3.24 System._Assert Function

Pascal

```
procedure _Assert(const Message: String; const Filename: String; LineNumber: Integer);
```

C++

```
_Assert(const UnicodeString Message, const UnicodeString Filename, int LineNumber);
```

File

System

Description

This is function System._Assert.

38.3.25 System._Assign Function

Pascal

```
function _Assign(var t: TTextRec; const s: PChar): Integer;
```

C++

```
int _Assign(TTextRec t, const const char * s);
```

File

System

Description

This is function System._Assign.

38.3.26 System._AStrCmp Function

Pascal

```
procedure _AStrCmp;
```

C++

```
_AStrCmp();
```

File

System

Description

This is function System._AStrCmp.

38.3.27 System._BeforeDestruction Function

Pascal

```
function _BeforeDestruction(Instance: TObject; OuterMost: ShortInt): TObject;
```

C++

```
TObject * _BeforeDestruction(TObject * Instance, ShortInt OuterMost);
```

File

System

Description

This is function System._BeforeDestruction.

38.3.28 System._BlockRead Function

Pascal

```
function _BlockRead(var f: TFileRec; buffer: Pointer; recCnt: Longint; var recsRead: Longint):
Longint;
```

C++

```
Longint _BlockRead(TFileRec f, void * buffer, Longint recCnt, Longint recsRead);
```

File

System

Description

This is function System._BlockRead.

38.3.29 System._BlockWrite Function

Pascal

```
function _BlockWrite(var f: TFileRec; buffer: Pointer; recCnt: Longint; var recsWritten:
Longint): Longint;
```

C++

```
Longint _BlockWrite(TFileRec f, void * buffer, Longint recCnt, Longint recsWritten);
```

File

System

Description

This is function System._BlockWrite.

38.3.30 System._BoundErr Function

Pascal

```
procedure _BoundErr;
```

C++

```
_BoundErr();
```

File

System

Description

This is function System._BoundErr.

38.3.31 System._CallDynaClass Function

Pascal

```
procedure _CallDynaClass;
```

C++

```
_CallDynaClass();
```

File

System

Description

This is function System._CallDynaClass.

38.3.32 System._CallDynaInst Function

Pascal

```
procedure _CallDynaInst;
```

C++

```
_CallDynaInst();
```

File

System

Description

This is function System._CallDynaInst.

38.3.33 System._CheckAutoResult Function

Pascal

```
function _CheckAutoResult(ResultCode: HResult): HResult;
```

C++

```
HRESULT _CheckAutoResult(HRESULT ResultCode);
```

File

System

Description

This is function System._CheckAutoResult.

38.3.34 System._ClassCreate Function

Pascal

```
function _ClassCreate(AClass: TClass; Alloc: Boolean): TObject;
```

C++

```
TObject * _ClassCreate(TClass AClass, Boolean Alloc);
```

File

System

Description

This is function System._ClassCreate.

38.3.35 System._ClassDestroy Function

Pascal

```
procedure _ClassDestroy(Instance: TObject);
```

C++

```
_ClassDestroy(TObject * Instance);
```

File

System

Description

This is function System._ClassDestroy.

38.3.36 System._Close Function

Pascal

```
function _Close(var t: TTextRec): Integer;
```

C++

```
int _Close(TTextRec t);
```

File

System

Description

This is function System._Close.

38.3.37 System._Copy Function

Pascal

```
procedure _Copy;
```

C++

```
_Copy();
```

File

System

Description

This is function System._Copy.

38.3.38 System._CopyArray Function

Pascal

```
procedure _CopyArray;
```

C++

```
_CopyArray();
```

File

System

Description

This is function System._CopyArray.

38.3.39 System._CopyObject Function

Pascal

```
procedure _CopyObject;
```

C++

```
_CopyObject();
```

File

System

Description

This is function System._CopyObject.

38.3.40 System._CopyRecord Function

Pascal

```
procedure _CopyRecord;
```

C++

```
_CopyRecord();
```

File

System

Description

This is function System._CopyRecord.

38.3.41 System._CVR_PROBE Function

Pascal

```
procedure _CVR_PROBE(mi: PCVModInfo; probeNum: Cardinal); cdecl;
```

C++

```
__cdecl _CVR_PROBE(PCVModInfo mi, unsigned probeNum);
```

File

System

Description

This is function System._CVR_PROBE.

38.3.42 System._CVR_STMTPROBE Function

Pascal

```
function _CVR_STMTPROBE(mi: PCVModInfo; probeNum: Cardinal; TrueFalse: Cardinal): Boolean;
cdecl;
```

C++

```
__cdecl Boolean _CVR_STMTPROBE(PCVModInfo mi, unsigned probeNum, unsigned TrueFalse);
```

File

System

Description

This is function System._CVR_STMTPROBE.

38.3.43 System._Delete Function

Pascal

```
procedure _Delete;
```

C++

```
_Delete();
```

File

System

Description

This is function System._Delete.

38.3.44 System._Dispose Function

Pascal

```
procedure _Dispose(p: Pointer; typeInfo: Pointer);
```

C++

```
_Dispose(void * p, void * typeInfo);
```

File

System

Description

This is function System._Dispose.

38.3.45 System._DoneExcept Function

Pascal

```
procedure _DoneExcept;
```

C++

```
_DoneExcept();
```

File

System

Description

This is function System._DoneExcept.

38.3.46 System._DynArrayAddRef Function

Pascal

```
procedure _DynArrayAddRef;
```

C++

```
_DynArrayAddRef();
```

File

System

Description

This is function System._DynArrayAddRef.

38.3.47 System._DynArrayAsg Function

Pascal

```
procedure _DynArrayAsg;
```

C++

```
_DynArrayAsg();
```

File

System

Description

This is function System._DynArrayAsg.

38.3.48 System._DynArrayClear Function

Pascal

```
procedure _DynArrayClear(var a: Pointer; typeInfo: Pointer);
```

C++

```
_DynArrayClear(void * a, void * typeInfo);
```

File

System

Description

This is function System._DynArrayClear.

38.3.49 System._DynArrayCopy Function

Pascal

```
procedure _DynArrayCopy(a: Pointer; typeInfo: Pointer; var Result: Pointer);
```

C++

```
_DynArrayCopy(void * a, void * typeInfo, void * Result);
```

File

System

Description

This is function System._DynArrayCopy.

38.3.50 System._DynArrayCopyRange Function

Pascal

```
procedure _DynArrayCopyRange(a: Pointer; typeInfo: Pointer; index: Integer; count: Integer;
var Result: Pointer);
```

C++

```
_DynArrayCopyRange(void * a, void * typeInfo, int index, int count, void * Result);
```

File

System

Description

This is function System._DynArrayCopyRange.

38.3.51 System._DynArrayHigh Function

Pascal

```
procedure _DynArrayHigh;
```

C++

```
_DynArrayHigh();
```

File

System

Description

This is function System._DynArrayHigh.

38.3.52 System._DynArrayLength Function

Pascal

```
procedure _DynArrayLength;
```

C++

```
_DynArrayLength();
```

File

System

Description

This is function System._DynArrayLength.

38.3.53 System._DynArraySetLength Function

Pascal

```
procedure _DynArraySetLength;
```

C++

```
_DynArraySetLength();
```

File

System

Description

This is function System._DynArraySetLength.

38.3.54 System._EnsureAnsiString Function

Pascal

```
function _EnsureAnsiString(var S: AnsiString; CodePage: Word): Pointer;
```

C++

```
void * _EnsureAnsiString(AnsiString S, Word CodePage);
```

File

System

Description

This is function System._EnsureAnsiString.

38.3.55 System._EnsureUnicodeString Function

Pascal

```
function _EnsureUnicodeString(var S: UnicodeString): Pointer;
```

C++

```
void * _EnsureUnicodeString(UnicodeString S);
```

File

System

Description

This is function System._EnsureUnicodeString.

38.3.56 System._EofFile Function

Pascal

```
function _EofFile(var f: TFileRec): Boolean;
```

C++

```
Boolean _EofFile(TFileRec f);
```

File

System

Description

This is function System._EofFile.

38.3.57 System._EofText Function

Pascal

```
function _EofText(var t: TTextRec): Boolean;
```

C++

```
Boolean _EofText(TTextRec t);
```

File

System

Description

This is function System._EofText.

38.3.58 System._Eoln Function

Pascal

```
function _Eoln(var t: TTextRec): Boolean;
```

C++

```
Boolean _Eoln(TTextRec t);
```

File

System

Description

This is function System._Eoln.

38.3.59 System._Erase Function

Pascal

```
procedure _Erase(var f: TFileRec);
```

C++

```
_Erase(TFileRec f);
```

File

System

Description

This is function System._Erase.

38.3.60 System._Ext2Real Function

Pascal

```
procedure _Ext2Real;
```

C++

```
_Ext2Real();
```

File

System

Description

This is function System._Ext2Real.

38.3.61 System._Fail Function

Pascal

```
procedure _Fail;
```

C++

```
_Fail();
```

File

System

Description

This is function System._Fail.

38.3.62 System._FilePos Function

Pascal

```
function _FilePos(var f: TFileRec): Longint;
```

C++

```
Longint _FilePos(TFileRec f);
```

File

System

Description

This is function System._FilePos.

38.3.63 System._FileSize Function

Pascal

```
function _FileSize(var f: TFileRec): Longint;
```

C++

```
Longint _FileSize(TFileRec f);
```

File

System

Description

This is function System._FileSize.

38.3.64 System._FillChar Function

Pascal

```
procedure _FillChar(var Dest; count: Integer; Value: Char);
```

C++

```
_FillChar( Dest, int count, Char Value);
```

File

System

Description

This is function System._FillChar.

38.3.65 System._Finalize Function

Pascal

```
procedure _Finalize(p: Pointer; typeInfo: Pointer);
```

C++

```
_Finalize(void * p, void * typeInfo);
```

File

System

Description

This is function System._Finalize.

38.3.66 System._FinalizeArray Function

Pascal

```
procedure _FinalizeArray(p: Pointer; typeInfo: Pointer; elemCount: Cardinal);
```

C++

```
_FinalizeArray(void * p, void * typeInfo, unsigned elemCount);
```

File

System

Description

This is function System._FinalizeArray.

38.3.67 System._FinalizeRecord Function

Pascal

```
procedure _FinalizeRecord(P: Pointer; typeInfo: Pointer);
```

C++

```
_FinalizeRecord(void * P, void * typeInfo);
```

File

System

Description

This is function System._FinalizeRecord.

38.3.68 System._FindDynaClass Function

Pascal

```
procedure _FindDynaClass;
```

C++

```
_FindDynaClass();
```

File

System

Description

This is function System._FindDynaClass.

38.3.69 System._FindDynaInst Function

Pascal

```
procedure _FindDynaInst;
```

C++

```
_FindDynaInst();
```

File

System

Description

This is function System._FindDynaInst.

38.3.70 System._Flush Function

Pascal

```
function _Flush(var t: TTextRec): Integer;
```

C++

```
int _Flush(TTextRec t);
```

File

System

Description

This is function System._Flush.

38.3.71 System._FreeMem Function

Pascal

```
function _FreeMem(P: Pointer): Integer;
```

C++

```
int _FreeMem(void * P);
```

File

System

Description

This is function System._FreeMem.

38.3.72 System._FSafeDivide Function

Pascal

```
procedure _FSafeDivide;
```

C++

```
_FSafeDivide();
```

File

System

Description

This is function System._FSafeDivide.

38.3.73 System._FSafeDivideR Function

Pascal

```
procedure _FSafeDivideR;
```

C++

```
_FSafeDivideR();
```

File

System

Description

This is function System._FSafeDivideR.

38.3.74 System._GetHelperIntf Function

Pascal

```
function _GetHelperIntf(Instance: TObject; HelperClass: TClass): IInterface;
```

C++

```
IInterface _GetHelperIntf(TObject * Instance, TClass HelperClass);
```

File

System

Description

This is function System._GetHelperIntf.

38.3.75 System._GetMem Function

Pascal

```
function _GetMem(Size: Integer): Pointer;
```

C++

```
void * _GetMem(int Size);
```

File

System

Description

This is function System._GetMem.

38.3.76 System._Halt Function

Pascal

```
procedure _Halt(Code: Integer);
```

C++

```
_Halt(int Code);
```

File

System

Description

This is function System._Halt.

38.3.77 System._Halt0 Function

Pascal

```
procedure _Halt0;
```

C++

```
_Halt0();
```

File

System

Description

This is function System._Halt0.

38.3.78 System._HandleAnyException Function

Pascal

```
procedure _HandleAnyException;
```

C++

```
_HandleAnyException();
```

File

System

Description

This is function System._HandleAnyException.

38.3.79 System._HandleAutoException Function

Pascal

```
procedure _HandleAutoException;
```

C++

```
_HandleAutoException();
```

File

System

Description

This is function System._HandleAutoException.

38.3.80 System._HandleFinally Function

Pascal

```
procedure _HandleFinally;
```

C++

```
_HandleFinally();
```

File

System

Description

This is function System._HandleFinally.

38.3.81 System._HandleOnException Function

Pascal

```
procedure _HandleOnException;
```

C++

```
_HandleOnException();
```

File

System

Description

This is function System._HandleOnException.

38.3.82 System._Initialize Function

Pascal

```
procedure _Initialize(p: Pointer; typeInfo: Pointer);
```

C++

```
_Initialize(void * p, void * typeInfo);
```

File

System

Description

This is function System._Initialize.

38.3.83 System._InitializeArray Function

Pascal

```
procedure _InitializeArray(p: Pointer; typeInfo: Pointer; elemCount: Cardinal);
```

C++

```
_InitializeArray(void * p, void * typeInfo, unsigned elemCount);
```

File

System

Description

This is function System._InitializeArray.

38.3.84 System._InitializeRecord Function

Pascal

```
procedure _InitializeRecord(p: Pointer; typeInfo: Pointer);
```

C++

```
_InitializeRecord(void * p, void * typeInfo);
```

File

System

Description

This is function System._InitializeRecord.

38.3.85 System._InitImports Function

Pascal

```
procedure _InitImports;
```

C++

```
_InitImports();
```

File

System

Description

This is function System._InitImports.

38.3.86 System._InitResStringImports Function

Pascal

```
procedure _InitResStringImports;
```

C++

```
_InitResStringImports();
```

File

System

Description

This is function System._InitResStringImports.

38.3.87 System._InitResStrings Function

Pascal

```
procedure _InitResStrings;
```

C++

```
_InitResStrings();
```

File

System

Description

This is function System._InitResStrings.

38.3.88 System._InitUnicodeStrings Function

Pascal

```
procedure _InitUnicodeStrings;
```

C++

```
_InitUnicodeStrings();
```

File

System

Description

This is function System._InitUnicodeStrings.

38.3.89 System._InitWideStrings Function

Pascal

```
procedure _InitWideStrings;
```

C++

```
_InitWideStrings();
```

File

System

Description

This is function System._InitWideStrings.

38.3.90 System._Insert Function

Pascal

```
procedure _Insert;
```

C++

```
_Insert();
```

File

System

Description

This is function System._Insert.

38.3.91 System._InternalLStrFromUStr Function

Pascal

```
function _InternalLStrFromUStr(var Dest: AnsiString; const Source: UnicodeString; CodePage:
Word): Pointer;
```

C++

```
void * _InternalLStrFromUStr(AnsiString Dest, const UnicodeString Source, Word CodePage);
```

File

System

Description

This is function System._InternalLStrFromUStr.

38.3.92 System._InternalUStrFromLStr Function

Pascal

```
function _InternalUStrFromLStr(var Dest: UnicodeString; const Source: AnsiString): Pointer;
```

C++

```
void * _InternalUStrFromLStr(UnicodeString Dest, const AnsiString Source);
```

File

System

Description

This is function System._InternalUStrFromLStr.

38.3.93 System._IntfAddRef Function

Pascal

```
procedure _IntfAddRef(const Dest: IInterface);
```

C++

```
_IntfAddRef(const IInterface Dest);
```

File

System

Description

This is function System._IntfAddRef.

38.3.94 System._IntfCast Function

Pascal

```
procedure _IntfCast(var Dest: IInterface; const Source: IInterface; const IID: TGUID);
```

C++

```
_IntfCast(IInterface Dest, const IInterface Source, const TGUID IID);
```

File

System

Description

This is function System._IntfCast.

38.3.95 System._IntfClear Function

Pascal

```
function _IntfClear(var Dest: IInterface): Pointer;
```

C++

```
void * _IntfClear(IInterface Dest);
```

File

System

Description

This is function System._IntfClear.

38.3.96 System._IntfCopy Function

Pascal

```
procedure _IntfCopy(var Dest: IInterface; const Source: IInterface);
```

C++

```
_IntfCopy(IInterface Dest, const IInterface Source);
```

File

System

Description

This is function System._IntfCopy.

38.3.97 System._IntfDispCall Function

Pascal

```
procedure _IntfDispCall; cdecl;
```

C++

```
__cdecl _IntfDispCall();
```

File

System

Description

This is function System._IntfDispCall.

38.3.98 System._IntfVarCall Function

Pascal

```
procedure _IntfVarCall; cdecl;
```

C++

```
__cdecl _IntfVarCall();
```

File

System

Description

This is function System._IntfVarCall.

38.3.99 System._IntOver Function

Pascal

```
procedure _IntOver;
```

C++

```
_IntOver();
```

File

System

Description

This is function System._IntOver.

38.3.100 System._IsClass Function

Pascal

```
function _IsClass(Child: TObject; Parent: TClass): Boolean;
```

C++

```
Boolean _IsClass(TObject * Child, TClass Parent);
```

File

System

Description

This is function System._IsClass.

38.3.101 System._LGetDir Function

Pascal

```
procedure _LGetDir(D: Byte; var S: AnsiString);
```

C++

```
_LGetDir(Byte D, AnsiString S);
```

File

System

Description

This is function System._LGetDir.

38.3.102 System._LStrAddRef Function

Pascal

```
function _LStrAddRef(var str): Pointer;
```

C++

```
void * _LStrAddRef( str);
```

File

System

Description

This is function System._LStrAddRef.

38.3.103 System._LStrArrayClr Function

Pascal

```
procedure _LStrArrayClr(var StrArray; cnt: longint);
```

C++

```
_LStrArrayClr( StrArray, longint cnt);
```

File

System

Description

This is function System._LStrArrayClr.

38.3.104 System._LStrAsg Function

Pascal

```
procedure _LStrAsg(var dest; const source);
```

C++

```
_LStrAsg( dest, const  source);
```

File

System

Description

This is function System._LStrAsg.

38.3.105 System._LStrCat Function

Pascal

```
procedure _LStrCat;
```

C++

```
_LStrCat();
```

File

System

Description

This is function System._LStrCat.

38.3.106 System._LStrCat3 Function

Pascal

```
procedure _LStrCat3;
```

C++

```
_LStrCat3();
```

File

System

Description

This is function System._LStrCat3.

38.3.107 System._LStrCatN Function

Pascal

```
procedure _LStrCatN;
```

C++

```
_LStrCatN();
```

File

System

Description

This is function System._LStrCatN.

38.3.108 System._LStrClr Function

Pascal

```
procedure _LStrClr(var S);
```

C++

```
_LStrClr( S);
```

File

System

Description

This is function System._LStrClr.

38.3.109 System._LStrCmp Function

Pascal

```
procedure _LStrCmp;
```

C++

```
_LStrCmp();
```

File

System

Description

This is function System._LStrCmp.

38.3.110 System._LStrCopy Function

Pascal

```
procedure _LStrCopy;
```

C++

```
_LStrCopy();
```

File

System

Description

This is function System._LStrCopy.

38.3.111 System._LStrDelete Function

Pascal

```
procedure _LStrDelete;
```

C++

```
_LStrDelete();
```

File

System

Description

This is function System._LStrDelete.

38.3.112 System._LStrEqual Function

Pascal

```
procedure _LStrEqual;
```

C++

```
_LStrEqual();
```

File

System

Description

This is function System._LStrEqual.

38.3.113 System._LStrFromArray Function

Pascal

```
procedure _LStrFromArray(var Dest: AnsiString; Source: PAnsiChar; Length: Integer; CodePage:
Word);
```

C++

```
_LStrFromArray(AnsiString Dest, PAnsiChar Source, int Length, Word CodePage);
```

File

System

Description

This is function System._LStrFromArray.

38.3.114 System._LStrFromChar Function

Pascal

```
procedure _LStrFromChar(var Dest: AnsiString; Source: AnsiChar; CodePage: Word);
```

C++

```
_LStrFromChar(AnsiString Dest, AnsiChar Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromChar.

38.3.115 System._LStrFromPChar Function

Pascal

```
procedure _LStrFromPChar(var Dest: AnsiString; Source: PAnsiChar; CodePage: Word);
```

C++

```
_LStrFromPChar(AnsiString Dest, PAnsiChar Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromPChar.

38.3.116 System._LStrFromPCharLen Function

Pascal

```
procedure _LStrFromPCharLen(var Dest: AnsiString; Source: PAnsiChar; Length: Integer;
CodePage: Word);
```

C++

```
_LStrFromPCharLen(AnsiString Dest, PAnsiChar Source, int Length, Word CodePage);
```

File

System

Description

This is function System._LStrFromPCharLen.

38.3.117 System._LStrFromPWChar Function

Pascal

```
procedure _LStrFromPWChar(var Dest: AnsiString; Source: PWideChar; CodePage: Word);
```

C++

```
_LStrFromPWChar(AnsiString Dest, PWideChar Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromPWChar.

38.3.118 System._LStrFromPWCharLen Function

Pascal

```
procedure _LStrFromPWCharLen(var Dest: AnsiString; Source: PWideChar; Length: Integer;
CodePage: Word);
```

C++

```
_LStrFromPWCharLen(AnsiString Dest, PWideChar Source, int Length, Word CodePage);
```

File

System

Description

This is function System._LStrFromPWCharLen.

38.3.119 System._LStrFromString Function

Pascal

```
procedure _LStrFromString(var Dest: AnsiString; const Source: ShortString; CodePage: Word);
```

C++

```
_LStrFromString(AnsiString Dest, const ShortString Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromString.

38.3.120 System._LStrFromUStr Function

Pascal

```
procedure _LStrFromUStr(var Dest: AnsiString; const Source: UnicodeString; CodePage: Word);
```

C++

```
_LStrFromUStr(AnsiString Dest, const UnicodeString Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromUStr.

38.3.121 System._LStrFromWArray Function

Pascal

```
procedure _LStrFromWArray(var Dest: AnsiString; Source: PWideChar; Length: Integer; CodePage:
Word);
```

C++

```
_LStrFromWArray(AnsiString Dest, PWideChar Source, int Length, Word CodePage);
```

File

System

Description

This is function System._LStrFromWArray.

38.3.122 System._LStrFromWChar Function

Pascal

```
procedure _LStrFromWChar(var Dest: AnsiString; Source: WideChar; CodePage: Word);
```

C++

```
_LStrFromWChar(AnsiString Dest, WideChar Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromWChar.

38.3.123 System._LStrFromWStr Function

Pascal

```
procedure _LStrFromWStr(var Dest: AnsiString; const Source: WideString; CodePage: Word);
```

C++

```
_LStrFromWStr(AnsiString Dest, const BSTR Source, Word CodePage);
```

File

System

Description

This is function System._LStrFromWStr.

38.3.124 System._LStrInsert Function

Pascal

```
procedure _LStrInsert;
```

C++

```
_LStrInsert();
```

File

System

Description

This is function System._LStrInsert.

38.3.125 System._LStrLAsg Function

Pascal

```
procedure _LStrLAsg(var dest; const source);
```

C++

```
_LStrLAsg( dest, const  source);
```

File

System

Description

This is function System._LStrLAsg.

38.3.126 System._LStrLen Function

Pascal

```
function _LStrLen(const S: AnsiString): Longint;
```

C++

```
Longint _LStrLen(const AnsiString S);
```

File

System

Description

This is function System._LStrLen.

38.3.127 System._LStrPos Function

Pascal

```
procedure _LStrPos;
```

C++

```
_LStrPos();
```

File

System

Description

This is function System._LStrPos.

38.3.128 System._LStrSetLength Function

Pascal

```
procedure _LStrSetLength;
```

C++

```
_LStrSetLength();
```

File

System

Description

This is function System._LStrSetLength.

38.3.129 System._LStrToPChar Function

Pascal

```
function _LStrToPChar(const s: AnsiString): PAnsiChar;
```

C++

```
PAnsiChar _LStrToPChar(const AnsiString s);
```

File

System

Description

This is function System._LStrToPChar.

38.3.130 System._LStrToString Function

Pascal

```
procedure _LStrToString;
```

C++

```
_LStrToString();
```

File

System

Description

This is function System._LStrToString.

38.3.131 System._New Function

Pascal

```
function _New(size: Longint; typeInfo: Pointer): Pointer;
```

C++

```
void * _New(Longint size, void * typeInfo);
```

File

System

Description

This is function System._New.

38.3.132 System._NewAnsiString Function

Pascal

```
function _NewAnsiString(length: Longint; CodePage: Word): Pointer;
```

C++

```
void * _NewAnsiString(Longint length, Word CodePage);
```

File

System

Description

This is function System._NewAnsiString.

38.3.133 System._NewUnicodeString Function

Pascal

```
function _NewUnicodeString(CharLength: Longint): Pointer;
```

C++

```
void * _NewUnicodeString(Longint CharLength);
```

File

System

Description

This is function System._NewUnicodeString.

38.3.134 System._NewWideString Function

Pascal

```
function _NewWideString(CharLength: Longint): Pointer;
```

C++

```
void * _NewWideString(Longint CharLength);
```

File

System

Description

This is function System._NewWideString.

38.3.135 System._ObjCopy Function

Pascal

```
procedure _ObjCopy;
```

C++

```
_ObjCopy();
```

File

System

Description

This is function System._ObjCopy.

38.3.136 System._ObjSetup Function

Pascal

```
procedure _ObjSetup;
```

C++

```
_ObjSetup();
```

File

System

Description

This is function System._ObjSetup.

38.3.137 System._PackageLoad Function

Pascal

```
procedure _PackageLoad(const Table: PackageInfo; Module: PLibModule);
```

C++

```
_PackageLoad(const PackageInfo Table, PLibModule Module);
```

File

System

Description

This is function System._PackageLoad.

38.3.138 System._PackageUnload Function

Pascal

```
procedure _PackageUnload(const Table: PackageInfo; Module: PLibModule);
```

C++

```
_PackageUnload(const PackageInfo Table, PLibModule Module);
```

File

System

Description

This is function System._PackageUnload.

38.3.139 System._PCharLen Function

Pascal

```
function _PCharLen(P: PAnsiChar): Longint;
```

C++

```
Longint _PCharLen(PAnsiChar P);
```

File

System

Description

This is function System._PCharLen.

38.3.140 System._Pos Function

Pascal

```
procedure _Pos;
```

C++

```
_Pos();
```

File

System

Description

This is function System._Pos.

38.3.141 System._Pow10 Function

Pascal

```
procedure _Pow10;
```

C++

```
_Pow10();
```

File

System

Description

This is function System._Pow10.

38.3.142 System._PStrCat Function

Pascal

```
procedure _PStrCat;
```

C++

```
_PStrCat();
```

File

System

Description

This is function System._PStrCat.

38.3.143 System._PStrCmp Function

Pascal

```
procedure _PStrCmp;
```

C++

```
_PStrCmp();
```

File

System

Description

This is function System._PStrCmp.

38.3.144 System._PStrCpy Function

Pascal

```
procedure _PStrCpy(Dest: PShortString; Source: PShortString);
```

C++

```
_PStrCpy(PShortString Dest, PShortString Source);
```

File

System

Description

This is function System._PStrCpy.

38.3.145 System._PStrNCat Function

Pascal

```
procedure _PStrNCat;
```

C++

```
_PStrNCat();
```

File

System

Description

This is function System._PStrNCat.

38.3.146 System._PStrNCpy Function

Pascal

```
procedure _PStrNCpy(Dest: PShortString; Source: PShortString; MaxLen: Byte);
```

C++

```
_PStrNCpy(PShortString Dest, PShortString Source, Byte MaxLen);
```

File

System

Description

This is function System._PStrNCpy.

38.3.147 System._PWCharLen Function

Pascal

```
function _PWCharLen(P: PWideChar): Longint;
```

C++

```
Longint _PWCharLen(PWideChar P);
```

File

System

Description

This is function System._PWCharLen.

38.3.148 System._RaiseAgain Function

Pascal

```
procedure _RaiseAgain;
```

C++

```
_RaiseAgain();
```

File

System

Description

This is function System._RaiseAgain.

38.3.149 System._RaiseExcept Function

Pascal

```
procedure _RaiseExcept;
```

C++

```
_RaiseExcept();
```

File

System

Description

This is function System._RaiseExcept.

38.3.150 System._ReadChar Function

Pascal

```
function _ReadChar(var t: TTextRec): AnsiChar;
```

C++

```
AnsiChar _ReadChar(TTextRec t);
```

File

System

Description

This is function System._ReadChar.

38.3.151 System._ReadCString Function

Pascal

```
procedure _ReadCString(var t: TTextRec; s: PAnsiChar; maxLen: Longint);
```

C++

```
_ReadCString(TTextRec t, PAnsiChar s, Longint maxLen);
```

File

System

Description

This is function System._ReadCString.

38.3.152 System._ReadExt Function

Pascal

```
function _ReadExt(var t: TTextRec): Extended;
```

C++

```
Extended _ReadExt(TTextRec t);
```

File

System

Description

This is function System._ReadExt.

38.3.153 System._ReadInt64 Function

Pascal

```
procedure _ReadInt64;
```

C++

```
_ReadInt64();
```

File

System

Description

This is function System._ReadInt64.

38.3.154 System._ReadLn Function

Pascal

```
procedure _ReadLn(var t: TTextRec);
```

C++

```
_ReadLn(TTextRec t);
```

File

System

Description

This is function System._ReadLn.

38.3.155 System._ReadLong Function

Pascal

```
function _ReadLong(var t: TTextRec): Longint;
```

C++

```
Longint _ReadLong(TTextRec t);
```

File

System

Description

This is function System._ReadLong.

38.3.156 System._ReadLString Function

Pascal

```
procedure _ReadLString(var t: TTextRec; var s: AnsiString; CodePage: Word);
```

C++

```
_ReadLString(TTextRec t, AnsiString s, Word CodePage);
```

File

System

Description

This is function System._ReadLString.

38.3.157 System._ReadRec Function

Pascal

```
function _ReadRec(var f: TFileRec; Buffer: Pointer): Integer;
```

C++

```
int _ReadRec(TFileRec f, void * Buffer);
```

File

System

Description

This is function System._ReadRec.

38.3.158 System._ReadString Function

Pascal

```
procedure _ReadString(var t: TTextRec; s: PShortString; maxLen: Longint);
```

C++

```
_ReadString(TTextRec t, PShortString s, Longint maxLen);
```

File

System

Description

This is function System._ReadString.

38.3.159 System._ReadUString Function

Pascal

```
procedure _ReadUString(var t: TTextRec; var s: UnicodeString);
```

C++

```
_ReadUString(TTextRec t, UnicodeString s);
```

File

System

Description

This is function System._ReadUString.

38.3.160 System._ReadWChar Function

Pascal

```
function _ReadWChar(var t: TTextRec): WideChar;
```

C++

```
WideChar _ReadWChar(TTextRec t);
```

File

System

Description

This is function System._ReadWChar.

38.3.161 System._ReadWCString Function

Pascal

```
procedure _ReadWCString(var t: TTextRec; s: PWideChar; maxBytes: Longint);
```

C++

```
_ReadWCString(TTextRec t, PWideChar s, Longint maxBytes);
```

File

System

Description

This is function System._ReadWCString.

38.3.162 System._ReadWString Function

Pascal

```
procedure _ReadWString(var t: TTextRec; var s: WideString);
```

C++

```
_ReadWString(TTextRec t, BSTR s);
```

File

System

Description

This is function System._ReadWString.

38.3.163 System._Real2Ext Function

Pascal

```
procedure _Real2Ext;
```

C++

```
_Real2Ext();
```

File

System

Description

This is function System._Real2Ext.

38.3.164 System._ReallocMem Function

Pascal

```
function _ReallocMem(var P: Pointer; NewSize: Integer): Pointer;
```

C++

```
void * _ReallocMem(void * P, int NewSize);
```

File

System

Description

This is function System._ReallocMem.

38.3.165 System._Rename Function

Pascal

```
procedure _Rename(var f: TFileRec; newName: PChar);
```

C++

```
_Rename(TFileRec f, const char * newName);
```

File

System

Description

This is function System._Rename.

38.3.166 System._ResetFile Function

Pascal

```
function _ResetFile(var f: TFileRec; recSize: Longint): Integer;
```

C++

```
int _ResetFile(TFileRec f, Longint recSize);
```

File

System

Description

This is function System._ResetFile.

38.3.167 System._ResetText Function

Pascal

```
function _ResetText(var t: TTextRec): Integer;
```

C++

```
int _ResetText(TTextRec t);
```

File

System

Description

This is function System._ResetText.

38.3.168 System._RewritFile Function

Pascal

```
function _RewritFile(var f: TFileRec; recSize: Longint): Integer;
```

C++

```
int _RewritFile(TFileRec f, Longint recSize);
```

File

System

Description

This is function System._RewritFile.

38.3.169 System._RewritText Function

Pascal

```
function _RewritText(var t: TTextRec): Integer;
```

C++

```
int _RewritText(TTextRec t);
```

File

System

Description

This is function System._RewritText.

38.3.170 System._ROUND Function

Pascal

```
procedure _ROUND;
```

C++

```
_ROUND();
```

File

System

Description

This is function System._ROUND.

38.3.171 System::_roundToInt64 Function

Pascal

```
function _roundToInt64(: long double): __int64; cdecl;
```

C++

```
__cdecl __int64 _roundToInt64(long double);
```

File

syscurr.h

Description

This is function System::_roundToInt64.

38.3.172 System._Run0Error Function

Pascal

```
procedure _Run0Error;
```

C++

```
_Run0Error();
```

File

System

Description

This is function System._Run0Error.

38.3.173 System._RunError Function

Pascal

```
procedure _RunError(errorCode: Byte);
```

C++

```
_RunError(Byte errorCode);
```

File

System

Description

This is function System._RunError.

38.3.174 System._Seek Function

Pascal

```
procedure _Seek(var f: TFileRec; recNum: Cardinal);
```

C++

```
_Seek(TFileRec f, unsigned recNum);
```

File

System

Description

This is function System._Seek.

38.3.175 System._SeekEof Function

Pascal

```
function _SeekEof(var t: TTextRec): Boolean;
```

C++

```
Boolean _SeekEof(TTextRec t);
```

File

System

Description

This is function System._SeekEof.

38.3.176 System._SeekEoln Function

Pascal

```
function _SeekEoln(var t: TTextRec): Boolean;
```

C++

```
Boolean _SeekEoln(TTextRec t);
```

File

System

Description

This is function System._SeekEoln.

38.3.177 System._SetElem Function

Pascal

```
procedure _SetElem;
```

C++

```
_SetElem();
```

File

System

Description

This is function System._SetElem.

38.3.178 System._SetEq Function

Pascal

```
procedure _SetEq;
```

C++

```
_SetEq();
```

File

System

Description

This is function System._SetEq.

38.3.179 System._SetExpand Function

Pascal

```
procedure _SetExpand;
```

C++

```
_SetExpand();
```

File

System

Description

This is function System._SetExpand.

38.3.180 System._SetIntersect Function

Pascal

```
procedure _SetIntersect;
```

C++

```
_SetIntersect();
```

File

System

Description

This is function System._SetIntersect.

38.3.181 System._SetIntersect3 Function

Pascal

```
procedure _SetIntersect3;
```

C++

```
_SetIntersect3();
```

File

System

Description

This is function System._SetIntersect3.

38.3.182 System._SetLe Function

Pascal

```
procedure _SetLe;
```

C++

```
_SetLe();
```

File

System

Description

This is function System._SetLe.

38.3.183 System._SetLength Function

Pascal

```
procedure _SetLength(s: PShortString; newLength: Byte);
```

C++

```
_SetLength(PShortString s, Byte newLength);
```

File

System

Description

This is function System._SetLength.

38.3.184 System._SetRange Function

Pascal

```
procedure _SetRange;
```

C++

```
_SetRange();
```

File

System

Description

This is function System._SetRange.

38.3.185 System._SetString Function

Pascal

```
procedure _SetString(s: PShortString; buffer: PAnsiChar; len: Byte);
```

C++

```
_SetString(PShortString s, PAnsiChar buffer, Byte len);
```

File

System

Description

This is function System._SetString.

38.3.186 System._SetSub Function

Pascal

```
procedure _SetSub;
```

C++

```
_SetSub();
```

File

System

Description

This is function System._SetSub.

38.3.187 System._SetSub3 Function

Pascal

```
procedure _SetSub3;
```

C++

```
_SetSub3();
```

File

System

Description

This is function System._SetSub3.

38.3.188 System._SetTextBuf Function

Pascal

```
procedure _SetTextBuf(var t: TTextRec; p: Pointer; size: Longint);
```

C++

```
_SetTextBuf(TTextRec t, void * p, Longint size);
```

File

System

Description

This is function System._SetTextBuf.

38.3.189 System._SetUnion Function

Pascal

```
procedure _SetUnion;
```

C++

```
_SetUnion();
```

File

System

Description

This is function System._SetUnion.

38.3.190 System._SetUnion3 Function

Pascal

```
procedure _SetUnion3;
```

C++

```
_SetUnion3();
```

File

System

Description

This is function System._SetUnion3.

38.3.191 System._SGetDir Function

Pascal

```
procedure _SGetDir(D: Byte; var S: ShortString);
```

C++

```
_SGetDir(Byte D, ShortString S);
```

File

System

Description

This is function System._SGetDir.

38.3.192 System._StartExe Function

Pascal

```
procedure _StartExe(InitTable: PackageInfo; Module: PLibModule);
```

C++

```
_StartExe(PackageInfo InitTable, PLibModule Module);
```

File

System

Description

This is function System._StartExe.

38.3.193 System._StartLib Function

Pascal

```
procedure _StartLib;
```

C++

```
_StartLib();
```

File

System

Description

This is function System._StartLib.

38.3.194 System._Str0Ext Function

Pascal

```
procedure _Str0Ext;
```

C++

```
_Str0Ext();
```

File

System

Description

This is function System._Str0Ext.

38.3.195 System._Str0Int64 Function

Pascal

```
function _Str0Int64(val: Int64): ShortString;
```

C++

```
ShortString _Str0Int64(Int64 val);
```

File

System

Description

This is function System._Str0Int64.

38.3.196 System._Str0Long Function

Pascal

```
procedure _Str0Long(val: Longint; s: PShortString);
```

C++

```
_Str0Long(Longint val, PShortString s);
```

File

System

Description

This is function System._Str0Long.

38.3.197 System._Str0UInt64 Function

Pascal

```
function _Str0UInt64(val: Int64): ShortString;
```

C++

```
ShortString _Str0UInt64(Int64 val);
```

File

System

Description

This is function System._Str0UInt64.

38.3.198 System._Str1Ext Function

Pascal

```
procedure _Str1Ext;
```

C++

```
_Str1Ext();
```

File

System

Description

This is function System._Str1Ext.

38.3.199 System._Str2Ext Function

Pascal

```
procedure _Str2Ext;
```

C++

```
_Str2Ext();
```

File

System

Description

This is function System._Str2Ext.

38.3.200 System._StrInt64 Function

Pascal

```
function _StrInt64(val: Int64; width: Integer): ShortString;
```

C++

```
ShortString _StrInt64(Int64 val, int width);
```

File

System

Description

This is function System._StrInt64.

38.3.201 System._StrLong Function

Pascal

```
procedure _StrLong(val: Longint; width: Longint; s: PShortString);
```

C++

```
_StrLong(Longint val, Longint width, PShortString s);
```

File

System

Description

This is function System._StrLong.

38.3.202 System._StrUInt64 Function

Pascal

```
function _StrUInt64(val: UInt64; width: Integer): ShortString;
```

C++

```
ShortString _StrUInt64(UInt64 val, int width);
```

File

System

Description

This is function System._StrUInt64.

38.3.203 System._TRUNC Function

Pascal

```
procedure _TRUNC;
```

C++

```
_TRUNC();
```

File

System

Description

This is function System._TRUNC.

38.3.204 System._Truncate Function

Pascal

```
procedure _Truncate(var f: TFileRec);
```

C++

```
_Truncate(TFileRec f);
```

File

System

Description

This is function System._Truncate.

38.3.205 System._TryFinallyExit Function

Pascal

```
procedure _TryFinallyExit;
```

C++

```
_TryFinallyExit();
```

File

System

Description

This is function System._TryFinallyExit.

38.3.206 System._UGetDir Function

Pascal

```
procedure _UGetDir(D: Byte; var S: UnicodeString);
```

C++

```
_UGetDir(Byte D, UnicodeString S);
```

File

System

Description

This is function System._UGetDir.

38.3.207 System._UniqueStringA Function

Pascal

```
procedure _UniqueStringA(var str: AnsiString);
```

C++

```
_UniqueStringA(AnsiString str);
```

File

System

Description

This is function System._UniqueStringA.

38.3.208 System._UniqueStringU Function

Pascal

```
procedure _UniqueStringU(var str: UnicodeString);
```

C++

```
_UniqueStringU(UnicodeString str);
```

File

System

Description

This is function System._UniqueStringU.

38.3.209 System._UniqueStringW Function

Pascal

```
procedure _UniqueStringW(var str: WideString);
```

C++

```
_UniqueStringW(BSTR str);
```

File

System

Description

This is function System._UniqueStringW.

38.3.210 System._UStrAddRef Function

Pascal

```
function _UStrAddRef(str: Pointer): Pointer;
```

C++

```
void * _UStrAddRef(void * str);
```

File

System

Description

This is function System._UStrAddRef.

38.3.211 System._UStrArrayClr Function

Pascal

```
procedure _UStrArrayClr(var StrArray; Count: Integer);
```

C++

```
_UStrArrayClr( StrArray, int Count);
```

File

System

Description

This is function System._UStrArrayClr.

38.3.212 System._UStrAsg Function

Pascal

```
procedure _UStrAsg(var Dest: UnicodeString; const Source: UnicodeString);
```

C++

```
_UStrAsg(UnicodeString Dest, const UnicodeString Source);
```

File

System

Description

This is function System._UStrAsg.

38.3.213 System._UStrCat Function

Pascal

```
procedure _UStrCat(var Dest: UnicodeString; const Source: UnicodeString);
```

C++

```
_UStrCat(UnicodeString Dest, const UnicodeString Source);
```

File

System

Description

This is function System._UStrCat.

38.3.214 System._UStrCat3 Function

Pascal

```
procedure _UStrCat3(var Dest: UnicodeString; const Source1: UnicodeString; const Source2:
UnicodeString);
```

C++

```
_UStrCat3(UnicodeString Dest, const UnicodeString Source1, const UnicodeString Source2);
```

File

System

Description

This is function System._UStrCat3.

38.3.215 System._UStrCatN Function

Pascal

```
procedure _UStrCatN;
```

C++

```
_UStrCatN();
```

File

System

Description

This is function System._UStrCatN.

38.3.216 System._UStrClr Function

Pascal

```
procedure _UStrClr(var S);
```

C++

```
_UStrClr( S);
```

File

System

Description

This is function System._UStrClr.

38.3.217 System._UStrCmp Function

Pascal

```
procedure _UStrCmp;
```

C++

```
_UStrCmp();
```

File

System

Description

This is function System._UStrCmp.

38.3.218 System._UStrCopy Function

Pascal

```
function _UStrCopy(const S: UnicodeString; Index: Integer; Count: Integer): UnicodeString;
```

C++

```
UnicodeString _UStrCopy(const UnicodeString S, int Index, int Count);
```

File

System

Description

This is function System._UStrCopy.

38.3.219 System._UStrDelete Function

Pascal

```
procedure _UStrDelete(var S: UnicodeString; Index: Integer; Count: Integer);
```

C++

```
_UStrDelete(UnicodeString S, int Index, int Count);
```

File

System

Description

This is function System._UStrDelete.

38.3.220 System._UStrEqual Function

Pascal

```
procedure _UStrEqual;
```

C++

```
_UStrEqual();
```

File

System

Description

This is function System._UStrEqual.

38.3.221 System._UStrFromArray Function

Pascal

```
procedure _UStrFromArray(var Dest: UnicodeString; Source: PAnsiChar; Length: Integer);
```

C++

```
_UStrFromArray(UnicodeString Dest, PAnsiChar Source, int Length);
```

File

System

Description

This is function System._UStrFromArray.

38.3.222 System._UStrFromChar Function

Pascal

```
procedure _UStrFromChar(var Dest: UnicodeString; Source: AnsiChar);
```

C++

```
_UStrFromChar(UnicodeString Dest, AnsiChar Source);
```

File

System

Description

This is function System._UStrFromChar.

38.3.223 System._UStrFromLStr Function

Pascal

```
procedure _UStrFromLStr(var Dest: UnicodeString; const Source: AnsiString);
```

C++

```
_UStrFromLStr(UnicodeString Dest, const AnsiString Source);
```

File

System

Description

This is function System._UStrFromLStr.

38.3.224 System._UStrFromPChar Function

Pascal

```
procedure _UStrFromPChar(var Dest: UnicodeString; Source: PAnsiChar);
```

C++

```
_UStrFromPChar(UnicodeString Dest, PAnsiChar Source);
```

File

System

Description

This is function System._UStrFromPChar.

38.3.225 System._UStrFromPCharLen Function

Pascal

```
procedure _UStrFromPCharLen(var Dest: UnicodeString; Source: PAnsiChar; Length: Integer);
```

C++

```
_UStrFromPCharLen(UnicodeString Dest, PAnsiChar Source, int Length);
```

File

System

Description

This is function System._UStrFromPCharLen.

38.3.226 System._UStrFromPWChar Function

Pascal

```
procedure _UStrFromPWChar(var Dest: UnicodeString; Source: PWideChar);
```

C++

```
_UStrFromPWChar(UnicodeString Dest, PWideChar Source);
```

File

System

Description

This is function System._UStrFromPWChar.

38.3.227 System._UStrFromPWCharLen Function

Pascal

```
procedure _UStrFromPWCharLen(var Dest: UnicodeString; Source: PWideChar; CharLength: Integer);
```

C++

```
_UStrFromPWCharLen(UnicodeString Dest, PWideChar Source, int CharLength);
```

File

System

Description

This is function System._UStrFromPWCharLen.

38.3.228 System._UStrFromString Function

Pascal

```
procedure _UStrFromString(var Dest: UnicodeString; const Source: ShortString);
```

C++

```
_UStrFromString(UnicodeString Dest, const ShortString Source);
```

File

System

Description

This is function System._UStrFromString.

38.3.229 System._UStrFromWArray Function

Pascal

```
procedure _UStrFromWArray(var Dest: UnicodeString; Source: PWideChar; Length: Integer);
```

C++

```
_UStrFromWArray(UnicodeString Dest, PWideChar Source, int Length);
```

File

System

Description

This is function System._UStrFromWArray.

38.3.230 System._UStrFromWChar Function

Pascal

```
procedure _UStrFromWChar(var Dest: UnicodeString; Source: WideChar);
```

C++

```
_UStrFromWChar(UnicodeString Dest, WideChar Source);
```

File

System

Description

This is function System._UStrFromWChar.

38.3.231 System._UStrFromWStr Function

Pascal

```
procedure _UStrFromWStr(var Dest: UnicodeString; const Source: WideString);
```

C++

```
_UStrFromWStr(UnicodeString Dest, const BSTR Source);
```

File

System

Description

This is function System._UStrFromWStr.

38.3.232 System._UStrInsert Function

Pascal

```
procedure _UStrInsert(const Source: UnicodeString; var Dest: UnicodeString; Index: Integer);
```

C++

```
_UStrInsert(const UnicodeString Source, UnicodeString Dest, int Index);
```

File

System

Description

This is function System._UStrInsert.

38.3.233 System._UStrLAsg Function

Pascal

```
procedure _UStrLAsg(var Dest: UnicodeString; const Source: UnicodeString);
```

C++

```
_UStrLAsg(UnicodeString Dest, const UnicodeString Source);
```

File

System

Description

This is function System._UStrLAsg.

38.3.234 System._UStrLen Function

Pascal

```
function _UStrLen(const S: UnicodeString): Integer;
```

C++

```
int _UStrLen(const UnicodeString S);
```

File

System

Description

This is function System._UStrLen.

38.3.235 System._UStrSetLength Function

Pascal

```
procedure _UStrSetLength(var S: UnicodeString; NewLength: Integer);
```

C++

```
_UStrSetLength(UnicodeString S, int NewLength);
```

File

System

Description

This is function System._UStrSetLength.

38.3.236 System._UStrToPWChar Function

Pascal

```
function _UStrToPWChar(const S: UnicodeString): PWideChar;
```

C++

```
PWideChar _UStrToPWChar(const UnicodeString S);
```

File

System

Description

This is function System._UStrToPWChar.

38.3.237 System._UStrToString Function

Pascal

```
procedure _UStrToString(Dest: PShortString; const Source: UnicodeString; MaxLen: Integer);
```

C++

```
_UStrToString(PShortString Dest, const UnicodeString Source, int MaxLen);
```

File

System

Description

This is function System._UStrToString.

38.3.238 System._ValExt Function

Pascal

```
procedure _ValExt;
```

C++

```
_ValExt();
```

File

System

Description

This is function System._ValExt.

38.3.239 System._ValInt64 Function

Pascal

```
function _ValInt64(const s: string; var code: Integer): Int64;
```

C++

```
Int64 _ValInt64(const UnicodeString s, int code);
```

File

System

Description

This is function System._ValInt64.

38.3.240 System._ValLong Function

Pascal

```
function _ValLong(const s: string; var code: Integer): Longint;
```

C++

```
Longint _ValLong(const UnicodeString s, int code);
```

File

System

Description

This is function System._ValLong.

38.3.241 System._WCharToString Function

Pascal

```
procedure _WCharToString(Dest: PShortString; const Source: WideChar; MaxLen: Integer);
```

C++

```
_WCharToString(PShortString Dest, const WideChar Source, int MaxLen);
```

File

System

Description

This is function System._WCharToString.

38.3.242 System._WGetDir Function

Pascal

```
procedure _WGetDir(D: Byte; var S: WideString);
```

C++

```
_WGetDir(Byte D, BSTR S);
```

File

System

Description

This is function System._WGetDir.

38.3.243 System._Write0Bool Function

Pascal

```
function _Write0Bool(var t: TTextRec; val: Boolean): Pointer;
```

C++

```
void * _Write0Bool(TTextRec t, Boolean val);
```

File

System

Description

This is function System._Write0Bool.

38.3.244 System._Write0Char Function

Pascal

```
function _Write0Char(var t: TTextRec; c: AnsiChar): Pointer;
```

C++

```
void * _Write0Char(TTextRec t, AnsiChar c);
```

File

System

Description

This is function System._Write0Char.

38.3.245 System._Write0CString Function

Pascal

```
function _Write0CString(var t: TTextRec; s: PAnsiChar): Pointer;
```

C++

```
void * _Write0CString(TTextRec t, PAnsiChar s);
```

File

System

Description

This is function System._Write0CString.

38.3.246 System._Write0Ext Function

Pascal

```
procedure _Write0Ext;
```

C++

```
_Write0Ext();
```

File

System

Description

This is function System._Write0Ext.

38.3.247 System._Write0Int64 Function

Pascal

```
procedure _Write0Int64;
```

C++

```
_Write0Int64();
```

File

System

Description

This is function System._Write0Int64.

38.3.248 System._Write0Long Function

Pascal

```
function _Write0Long(var t: TTextRec; val: Longint): Pointer;
```

C++

```
void * _Write0Long(TTextRec t, Longint val);
```

File

System

Description

This is function System._Write0Long.

38.3.249 System._Write0LString Function

Pascal

```
function _Write0LString(var t: TTextRec; const s: AnsiString): Pointer;
```

C++

```
void * _Write0LString(TTextRec t, const AnsiString s);
```

File

System

Description

This is function System._Write0LString.

38.3.250 System._Write0String Function

Pascal

```
function _Write0String(var t: TTextRec; const s: ShortString): Pointer;
```

C++

```
void * _Write0String(TTextRec t, const ShortString s);
```

File

System

Description

This is function System._Write0String.

38.3.251 System._Write0UInt64 Function

Pascal

```
procedure _Write0UInt64;
```

C++

```
_Write0UInt64();
```

File

System

Description

This is function System._Write0UInt64.

38.3.252 System._Write0UString Function

Pascal

```
function _Write0UString(var t: TTextRec; const s: UnicodeString): Pointer;
```

C++

```
void * _Write0UString(TTextRec t, const UnicodeString s);
```

File

System

Description

This is function System._Write0UString.

38.3.253 System._Write0Variant Function

Pascal

```
function _Write0Variant(var T: TTextRec; const V: TVarData): Pointer;
```

C++

```
void * _Write0Variant(TTextRec T, const TVarData V);
```

File

System

Description

This is function System._Write0Variant.

38.3.254 System._Write0WChar Function

Pascal

```
function _Write0WChar(var t: TTextRec; c: WideChar): Pointer;
```

C++

```
void * _Write0WChar(TTextRec t, WideChar c);
```

File

System

Description

This is function System._Write0WChar.

38.3.255 System._Write0WCString Function

Pascal

```
function _Write0WCString(var t: TTextRec; s: PWideChar): Pointer;
```

C++

```
void * _Write0WCString(TTextRec t, PWideChar s);
```

File

System

Description

This is function System._Write0WCString.

38.3.256 System._Write0WString Function

Pascal

```
function _Write0WString(var t: TTextRec; const s: WideString): Pointer;
```

C++

```
void * _Write0WString(TTextRec t, const BSTR s);
```

File

System

Description

This is function System._Write0WString.

38.3.257 System._Write1Ext Function

Pascal

```
procedure _Write1Ext;
```

C++

```
_Write1Ext();
```

File

System

Description

This is function System._Write1Ext.

38.3.258 System._Write2Ext Function

Pascal

```
procedure _Write2Ext;
```

C++

```
_Write2Ext();
```

File

System

Description

This is function System._Write2Ext.

38.3.259 System._WriteBool Function

Pascal

```
function _WriteBool(var t: TTextRec; val: Boolean; width: Longint): Pointer;
```

C++

```
void * _WriteBool(TTextRec t, Boolean val, Longint width);
```

File

System

Description

This is function System._WriteBool.

38.3.260 System._WriteChar Function

Pascal

```
function _WriteChar(var t: TTextRec; c: AnsiChar; width: Integer): Pointer;
```

C++

```
void * _WriteChar(TTextRec t, AnsiChar c, int width);
```

File

System

Description

This is function System._WriteChar.

38.3.261 System._WriteCString Function

Pascal

```
function _WriteCString(var t: TTextRec; s: PAnsiChar; width: Longint): Pointer;
```

C++

```
void * _WriteCString(TTextRec t, PAnsiChar s, Longint width);
```

File

System

Description

This is function System._WriteCString.

38.3.262 System._WriteInt64 Function

Pascal

```
procedure _WriteInt64;
```

C++

```
_WriteInt64();
```

File

System

Description

This is function System._WriteInt64.

38.3.263 System._WriteLn Function

Pascal

```
function _WriteLn(var t: TTextRec): Pointer;
```

C++

```
void * _WriteLn(TTextRec t);
```

File

System

Description

This is function System._WriteLn.

38.3.264 System._WriteLong Function

Pascal

```
function _WriteLong(var t: TTextRec; val: Longint; width: Longint): Pointer;
```

C++

```
void * _WriteLong(TTextRec t, Longint val, Longint width);
```

File

System

Description

This is function System._WriteLong.

38.3.265 System._WriteLString Function

Pascal

```
function _WriteLString(var t: TTextRec; const s: AnsiString; width: Longint): Pointer;
```

C++

```
void * _WriteLString(TTextRec t, const AnsiString s, Longint width);
```

File

System

Description

This is function System._WriteLString.

38.3.266 System._WriteRec Function

Pascal

```
function _WriteRec(var f: TFileRec; buffer: Pointer): Pointer;
```

C++

```
void * _WriteRec(TFileRec f, void * buffer);
```

File

System

Description

This is function System._WriteRec.

38.3.267 System._WriteString Function

Pascal

```
function _WriteString(var t: TTextRec; const s: ShortString; width: Longint): Pointer;
```

C++

```
void * _WriteString(TTextRec t, const ShortString s, Longint width);
```

File

System

Description

This is function System._WriteString.

38.3.268 System._WriteUInt64 Function

Pascal

```
procedure _WriteUInt64;
```

C++

```
_WriteUInt64();
```

File

System

Description

This is function System._WriteUInt64.

38.3.269 System._WriteUString Function

Pascal

```
function _WriteUString(var t: TTextRec; const s: UnicodeString; width: Longint): Pointer;
```

C++

```
void * _WriteUString(TTextRec t, const UnicodeString s, Longint width);
```

File

System

Description

This is function System._WriteUString.

38.3.270 System._WriteVariant Function

Pascal

```
function _WriteVariant(var T: TTextRec; const V: TVarData; Width: Integer): Pointer;
```

C++

```
void * _WriteVariant(TTextRec T, const TVarData V, int Width);
```

File

System

Description

This is function System._WriteVariant.

38.3.271 System._WriteWChar Function

Pascal

```
function _WriteWChar(var t: TTextRec; c: WideChar; width: Integer): Pointer;
```

C++

```
void * _WriteWChar(TTextRec t, WideChar c, int width);
```

File

System

Description

This is function System._WriteWChar.

38.3.272 System._WriteWCString Function

Pascal

```
function _WriteWCString(var t: TTextRec; s: PWideChar; width: Longint): Pointer;
```

C++

```
void * _WriteWCString(TTextRec t, PWideChar s, Longint width);
```

File

System

Description

This is function System._WriteWCString.

38.3.273 System._WriteWString Function

Pascal

```
function _WriteWString(var t: TTextRec; const s: WideString; width: Longint): Pointer;
```

C++

```
void * _WriteWString(TTextRec t, const BSTR s, Longint width);
```

File

System

Description

This is function System._WriteWString.

38.3.274 System._WStrAddRef Function

Pascal

```
function _WStrAddRef(var str: WideString): Pointer;
```

C++

```
void * _WStrAddRef(BSTR str);
```

File

System

Description

This is function System._WStrAddRef.

38.3.275 System._WStrArrayClr Function

Pascal

```
procedure _WStrArrayClr(var StrArray; Count: Integer);
```

C++

```
_WStrArrayClr( StrArray, int Count);
```

File

System

Description

This is function System._WStrArrayClr.

38.3.276 System._WStrAsg Function

Pascal

```
procedure _WStrAsg(var Dest: WideString; const Source: WideString);
```

C++

```
_WStrAsg(BSTR Dest, const BSTR Source);
```

File

System

Description

This is function System._WStrAsg.

38.3.277 System._WStrCat Function

Pascal

```
procedure _WStrCat(var Dest: WideString; const Source: WideString);
```

C++

```
_WStrCat(BSTR Dest, const BSTR Source);
```

File

System

Description

This is function System._WStrCat.

38.3.278 System._WStrCat3 Function

Pascal

```
procedure _WStrCat3(var Dest: WideString; const Source1: WideString; const Source2:
WideString);
```

C++

```
_WStrCat3(BSTR Dest, const BSTR Source1, const BSTR Source2);
```

File

System

Description

This is function System._WStrCat3.

38.3.279 System._WStrCatN Function

Pascal

```
procedure _WStrCatN;
```

C++

```
_WStrCatN();
```

File

System

Description

This is function System._WStrCatN.

38.3.280 System._WStrClr Function

Pascal

```
procedure _WStrClr(var S);
```

C++

```
_WStrClr( S);
```

File

System

Description

This is function System._WStrClr.

38.3.281 System._WStrCmp Function

Pascal

```
procedure _WStrCmp;
```

C++

```
_WStrCmp();
```

File

System

Description

This is function System._WStrCmp.

38.3.282 System._WStrCopy Function

Pascal

```
function _WStrCopy(const S: WideString; Index: Integer; Count: Integer): WideString;
```

C++

```
BSTR _WStrCopy(const BSTR S, int Index, int Count);
```

File

System

Description

This is function System._WStrCopy.

38.3.283 System._WStrDelete Function

Pascal

```
procedure _WStrDelete(var S: WideString; Index: Integer; Count: Integer);
```

C++

```
_WStrDelete(BSTR S, int Index, int Count);
```

File

System

Description

This is function System._WStrDelete.

38.3.284 System._WStrEqual Function

Pascal

```
procedure _WStrEqual;
```

C++

```
_WStrEqual();
```

File

System

Description

This is function System._WStrEqual.

38.3.285 System._WStrFromArray Function

Pascal

```
procedure _WStrFromArray(var Dest: WideString; Source: PAnsiChar; Length: Integer);
```

C++

```
_WStrFromArray(BSTR Dest, PAnsiChar Source, int Length);
```

File

System

Description

This is function System._WStrFromArray.

38.3.286 System._WStrFromChar Function

Pascal

```
procedure _WStrFromChar(var Dest: WideString; Source: AnsiChar);
```

C++

```
_WStrFromChar(BSTR Dest, AnsiChar Source);
```

File

System

Description

This is function System._WStrFromChar.

38.3.287 System._WStrFromLStr Function

Pascal

```
procedure _WStrFromLStr(var Dest: WideString; const Source: AnsiString);
```

C++

```
_WStrFromLStr(BSTR Dest, const AnsiString Source);
```

File

System

Description

This is function System._WStrFromLStr.

38.3.288 System._WStrFromPChar Function

Pascal

```
procedure _WStrFromPChar(var Dest: WideString; Source: PAnsiChar);
```

C++

```
_WStrFromPChar(BSTR Dest, PAnsiChar Source);
```

File

System

Description

This is function System._WStrFromPChar.

38.3.289 System._WStrFromPCharLen Function

Pascal

```
procedure _WStrFromPCharLen(var Dest: WideString; Source: PAnsiChar; Length: Integer);
```

C++

```
_WStrFromPCharLen(BSTR Dest, PAnsiChar Source, int Length);
```

File

System

Description

This is function System._WStrFromPCharLen.

38.3.290 System._WStrFromPWChar Function

Pascal

```
procedure _WStrFromPWChar(var Dest: WideString; Source: PWideChar);
```

C++

```
_WStrFromPWChar(BSTR Dest, PWideChar Source);
```

File

System

Description

This is function System._WStrFromPWChar.

38.3.291 System._WStrFromPWCharLen Function

Pascal

```
procedure _WStrFromPWCharLen(var Dest: WideString; Source: PWideChar; CharLength: Integer);
```

C++

```
_WStrFromPWCharLen(BSTR Dest, PWideChar Source, int CharLength);
```

File

System

Description

This is function System._WStrFromPWCharLen.

38.3.292 System._WStrFromString Function

Pascal

```
procedure _WStrFromString(var Dest: WideString; const Source: ShortString);
```

C++

```
_WStrFromString(BSTR Dest, const ShortString Source);
```

File

System

Description

This is function System._WStrFromString.

38.3.293 System._WStrFromUStr Function

Pascal

```
procedure _WStrFromUStr(var Dest: WideString; const Source: UnicodeString);
```

C++

```
_WStrFromUStr(BSTR Dest, const UnicodeString Source);
```

File

System

Description

This is function System._WStrFromUStr.

38.3.294 System._WStrFromWArray Function

Pascal

```
procedure _WStrFromWArray(var Dest: WideString; Source: PWideChar; Length: Integer);
```

C++

```
_WStrFromWArray(BSTR Dest, PWideChar Source, int Length);
```

File

System

Description

This is function System._WStrFromWArray.

38.3.295 System._WStrFromWChar Function

Pascal

```
procedure _WStrFromWChar(var Dest: WideString; Source: WideChar);
```

C++

```
_WStrFromWChar(BSTR Dest, WideChar Source);
```

File

System

Description

This is function System._WStrFromWChar.

38.3.296 System._WStrInsert Function

Pascal

```
procedure _WStrInsert(const Source: WideString; var Dest: WideString; Index: Integer);
```

C++

```
_WStrInsert(const BSTR Source, BSTR Dest, int Index);
```

File

System

Description

This is function System._WStrInsert.

38.3.297 System._WStrLAsg Function

Pascal

```
procedure _WStrLAsg(var Dest: WideString; const Source: WideString);
```

C++

```
_WStrLAsg(BSTR Dest, const BSTR Source);
```

File

System

Description

This is function System._WStrLAsg.

38.3.298 System._WStrLCmp Function

Pascal

```
procedure _WStrLCmp;
```

C++

```
_WStrLCmp();
```

File

System

Description

This is function System._WStrLCmp.

38.3.299 System._WStrLen Function

Pascal

```
function _WStrLen(const S: WideString): Longint;
```

C++

```
Longint _WStrLen(const BSTR S);
```

File

System

Description

This is function System._WStrLen.

38.3.300 System._WStrSetLength Function

Pascal

```
procedure _WStrSetLength(var S: WideString; NewLength: Integer);
```

C++

```
_WStrSetLength(BSTR S, int NewLength);
```

File

System

Description

This is function System._WStrSetLength.

38.3.301 System._WStrToPWChar Function

Pascal

```
function _WStrToPWChar(const S: WideString): PWideChar;
```

C++

```
PWideChar _WStrToPWChar(const BSTR S);
```

File

System

Description

This is function System._WStrToPWChar.

38.3.302 System._WStrToString Function

Pascal

```
procedure _WStrToString(Dest: PShortString; const Source: WideString; MaxLen: Integer);
```

C++

```
_WStrToString(PShortString Dest, const BSTR Source, int MaxLen);
```

File

System

Description

This is function System._WStrToString.

38.3.303 System::+ Function

Pascal

```
operator +(const : char*; const : UnicodeString&);
```

C++

```
__fastcall UnicodeString operator +(const char*, const UnicodeString&);
```

File

ustring.h

Description

This is function System::+.

38.3.304 System::+ Function

Pascal

```
operator +(const : wchar_t*; const : UnicodeString&);
```

C++

```
__fastcall UnicodeString operator +(const wchar_t*, const UnicodeString&);
```

File

ustring.h

Description

This is function System::+.

38.3.305 System.Abs Function

Returns an absolute value.

Pascal

```
procedure Abs(X);
```

C++

```
Abs( X);
```

File

System

Description

Abs returns the absolute value of the argument, X.

X is an integer-type or real-type expression.

Delphi Examples:

```
{
This example converts negative floating point numbers and
integers to postive.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := '-2.3';
  Edit3.Text := FloatToStr(Abs(StrToFloat(Edit1.Text)));    { 2.3 }
  Edit2.Text := '-157';
  Edit4.Text := IntToStr(Abs(StrToInt(Edit2.Text)));    { 157 }
end;
```

38.3.306 System.AcquireExceptionObject Function

Retains access to exception object.

Pascal

```
function AcquireExceptionObject: Pointer;
```

C++

```
void * AcquireExceptionObject();
```

File

System

Description

AcquireExceptionObject returns a pointer to the current exception object and prevents the exception object from being deallocated when the current exception handler exits.

Note: AcquireExceptionObject increments the exception object's reference count. Make sure that the reference count is decremented when the exception object is no longer needed. This happens automatically if you use the exception object to re-raise the exception. In all other cases, every call to AcquireExceptionObject must have a matching call to ReleaseExceptionObject (see page 3732). AcquireExceptionObject/ReleaseExceptionObject (see page 3732) sequences can be nested.

See Also

ReleaseExceptionObject (see page 3732)

ExceptObject (see page 3664)

on

catch

38.3.307 System.AddModuleUnloadProc Function

AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.

Pascal

```
procedure AddModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure AddModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
AddModuleUnloadProc(TModuleUnloadProc Proc);
AddModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Use RemoveModuleUnloadProc (see page 3732) to reverse the effect of AddModuleUnloadProc.

See Also

RemoveModuleUnloadProc (see page 3732)

TModuleUnloadProc (see page 3863)

38.3.308 System.Addr Function

Returns a pointer to a specified object.

Pascal

```
function Addr(var X): Pointer;
```

C++

```
void * Addr( X);
```

File

System

Description

The Addr function returns the address of a specified object. X is any variable, procedure or function identifier. The result is a pointer to X.

The result of Addr is of the predefined type Pointer, which means that it is assignment-compatible with all pointer types but can't be dereferenced directly without a typecast.

Note: Addr is equivalent to the @ operator except that it is unaffected by the $T compiler directive.

Delphi Examples:

```
{
This example uses the Addr function to get the address of a
numeric value that is stored as custom data in a tree node.
Once the assignment has been made, changes to the entries in
the NodeNumbers array can be read by dereferencing the
pointers that are stored in the Data property of the Tree
nodes.  This example requires a populated TreeView.
}

var
  NodeNumbers: array [0 .. 200] of String;

procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
 begin
  with TreeView1 do
  begin
    for I := 0 to Items.Count - 1 do
    begin
      NodeNumbers[I] := Items[I].Text;
      Items[I].Data := Addr(NodeNumbers[I]);
      StringGrid1.Cells[0, I + 1] := IntToStr(I);
      StringGrid1.Cells[1, I + 1] := IntToStr(Integer(Items[I].Data));
      StringGrid1.Cells[2, I + 1] := NodeNumbers[I];
    end;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I : Integer;
  addr : Pointer;
begin
```

```
    I := StrToInt(Edit2.Text);
    addr := TreeView1.Items[I].Data;
    string(addr^) := Edit1.Text;  // NodeNumbers has been altered
    TreeView1.Items[I].Text := NodeNumbers[I]; // Update TreeView
    StringGrid1.Cells[2, I + 1] := NodeNumbers[I];  // Update StringGrid
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    StringGrid1.Cells[0, 0] := 'Node index';
    StringGrid1.Cells[1, 0] := 'Node addr';
    StringGrid1.Cells[2, 0] := 'Node text';
  end;
```

See Also

Ptr (see page 3723)

IDH

38.3.309 System.AllocMem Function

AllocMem allocates a memory block and initializes each byte to zero.

Pascal

```
function AllocMem(Size: Cardinal): Pointer;
```

C++

```
void * AllocMem(unsigned Size);
```

File

System

Description

AllocMem allocates a block of the given **Size** on the heap, and returns the address of this memory. Each byte in the allocated buffer is set to zero. To dispose of the buffer, use FreeMem (see page 3676). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory does not need to be zero-initialized, it is more efficient to use GetMem (see page 3683) instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
```

```
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
```

```
    end;
  end;
end;
```

See Also

FreeMem (see page 3676)

FreeMemory (see page 3678)

GetMem (see page 3683)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

ReallocMemory (see page 3730)

38.3.310 System.AnsiToUtf8 Function

Converts a string encoded in Ansi to UTF-8.

Pascal

```
function AnsiToUtf8(const S: string): RawByteString;
```

C++

```
RawByteString AnsiToUtf8(const UnicodeString S);
```

File

System

Description

AnsiToUtf8 converts the string specified by S, which uses the ANSI encoding system, to UTF-8.

See Also

Utf8ToAnsi (see page 3781)

Utf8ToUnicode (see page 3783)

WideStringToUCS4String (see page 3789)

38.3.311 System.Append Function

Prepares an existing file for adding text to the end.

Pascal

```
procedure Append(var F: Text);
```

C++

```
Append(Text F);
```

File

System

Description

Call Append to ensure that a file is opened with write-only access with the file pointer positioned at the end of the file. F is a text file variable and must be associated with an external file using AssignFile (see page 3629). If no external file of the given name exists, an error occurs. If F is already open, it is closed, then reopened. The current file position is set to the end of the file.

Note: If a Ctrl+Z (ASCII 26) is present in the last 128-byte block of the file, the current file position is set so that the next character added to the file overwrites the first Ctrl+Z in the block. In this way, text can be appended to a file that terminates with a Ctrl+Z.

If F was not assigned a name, then, after the call to Append, F refers to the standard output file.

Delphi Examples:

```
{
Click the button and select a file to append a string to the bottom of the file.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

AssignFile (see page 3629)

FileClose

Reset (see page 3734)

Rewrite (see page 3736)

IDH

38.3.312 System.ArcTan Function

Calculates the arctangent of a given number.

Pascal

```
function ArcTan(const X: Extended): Extended;
```

C++

```
Extended ArcTan(const Extended X);
```

File

System

Description

In Delphi code, ArcTan returns the arctangent of X.

X is a real-type expression that gives an angle in radians.

C++ Examples:

```
/*
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Extended ex = 16 * ArcTan(1.0/5.0) - 4.0 * ArcTan(1.0/239.0); // Machin's formula
  String mystring = FloatToStr(ex);
  Edit2->Text = mystring;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Text = "3.14159265358979";
}
```

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

See Also

Cos (see page 3650)

Sin (see page 3760)

ArcTan2

Tan

ArcSin

ArcCos

38.3.313 System.Assert Function

Tests whether a Boolean expression is true.

Pascal

```
procedure Assert(expr: Boolean); overload;
procedure Assert(expr: Boolean; const msg: string); overload;
```

C++

```
Assert(Boolean expr);
Assert(Boolean expr, const UnicodeString msg);
```

File

System

Description

In Delphi code, use Assert as a debugging tool to test that conditions assumed to be true are never violated. Assert provides an opportunity to intercept an unexpected condition and halt a program rather than allow execution to continue under unanticipated conditions.

Assert takes a Boolean expression and an optional message string as parameters. If the Boolean test fails, Assert raises an EAssertionFailed exception. If a message string was passed to Assert, the exception object is created with that string. Otherwise it is created with a default string indicating that the assertion failed. The message is displayed along with the complete path, filename, and the line number on which Assert failed.

The SysUtils (see page 3967) unit causes runtime errors to be turned into exceptions. If SysUtils (see page 3967) is not used anywhere in your application, you will get a runtime error 227 rather than an EAssertionFailed exception. This runtime error will halt the program.

Because assertions are not usually used in shipping versions of a product, compiler directives are provided to disable the generation of assertion code:

$ASSERTIONS ON/OFF(long form)

$C +/-(short form)

These are global switches that affect the entire source file where they occur, regardless of their position in the file. It is not possible to enable and disable assertions for something smaller than a source file. Assertions are on by default.

Delphi Examples:

```
{
This example exercises the System Assert function.  The first
call passes and the second call fails.
}

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
    end;

    TStorage = class(TObject)
      FData: string;
      property Data: string read FData write FData;
    private
      { Private declarations }
    public
      { Public declarations }
    end;

    var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure ModifyStorage(AStorage: TStorage; const s: string);
  begin
    Assert(AStorage <> nil, '');
    AStorage.Data := s;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    Storage: TStorage;
  begin
    Storage := TStorage.Create;
    try
      ModifyStorage(Storage, 'Hello world');
    finally
      Storage.Free;
    end;

    // The following call is buggy and triggers the Assert
    ModifyStorage(nil, 'Ooops');
  end;
```

See Also

EAssertionFailed

38.3.314 System.Assigned Function

Tests for a nil (unassigned) pointer or procedural variable.

Pascal

```
function Assigned(const P): Boolean;
```

C++

```
Boolean Assigned(const  P);
```

File

System

Description

Use Assigned to determine whether the pointer or procedure referenced by P is nil. P must be a variable reference of a pointer or procedural type. Assigned(P) corresponds to the test P<> nil for a pointer variable, and @P <> nil for a procedural variable.

Assigned returns false if P is nil, true otherwise.

Note: Assigned can't detect a dangling pointer--that is, one that isn't nil but no longer points to valid data. For example, in the code example for Assigned, Assigned won't detect the fact that P isn't valid.

Delphi Examples:

```
{
This example demonstrates the System Assigned function succeeds.
}
procedure TForm1.Button1Click(Sender: TObject);
var P: Pointer;
begin
  P := nil;
  if Assigned (P) then
    MessageDlg('You won''t see this', mtInformation, [mbOK], 0);
  GetMem(P, 1024);  {P valid}
  FreeMem(P, 1024); {P no longer valid and still not nil}
  if Assigned (P) then
    MessageDlg('You''ll see this', mtInformation, [mbOK], 0);
end;
```

38.3.315 System.AssignFile Function

Associates the name of an external file with a file variable.

Pascal

```
procedure AssignFile(var F; FileName: string);
```

C++

```
AssignFile( F, UnicodeString FileName);
```

File

System

Description

Call AssignFile to initialize a file variable in Delphi code. F is a file variable of any file type. FileName is a string-type expression or an expression of type PChar if extended syntax is enabled.

After calling AssignFile, F is associated with the external file until F is closed. All further operations on the file variable F operate on the external file named by FileName.

When the FileName parameter is empty, AssignFile associates F with the standard input or standard output file. If assigned an empty name, after a call to Reset (see page 3734) (F), F refers to the standard input file, and after a call to Rewrite (see page 3736) (F), F refers to the standard output file.

Do not use AssignFile on a file variable that is already open.

Note: To avoid scope conflicts, AssignFile replaces the Assign procedure that was available in early versions of the Delphi product. However, for backward compatibility Assign is still available.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
```

```
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.  Click on the text edit to open the
open dialog box.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
{
Click the button to open a TOpenDialog, then select a file
to delete.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

Append (see page 3624)

CloseFile (see page 3643)

Reset (see page 3734)

Rewrite (see page 3736)

IDH

38.3.316 System.AttemptToUseSharedMemoryManager Function

Attempts to use a shared memory manager.

Pascal

```
function AttemptToUseSharedMemoryManager: Boolean;
```

C++

```
Boolean AttemptToUseSharedMemoryManager();
```

File

System

Description

AttemptToUseSharedMemoryManager searches the current process for a shared memory manager. If no memory has been allocated using this module's memory manager, it will switch to using the shared memory manager. It will return **True** if it found a shared memory manager to which it could switch. It will return **False** otherwise.

Note: The memory manager cannot be changed once memory has been allocated, since the new memory manager cannot free the pointers allocated through a different memory manager. This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

38.3.317 System.BeginThread Function

Spawns a separate thread of execution.

Pascal

```
function BeginThread(SecurityAttributes: Pointer; StackSize: LongWord; ThreadFunc:
TThreadFunc; Parameter: Pointer; CreationFlags: LongWord; var ThreadId: LongWord): Integer;
```

C++

```
int BeginThread(void * SecurityAttributes, LongWord StackSize, TThreadFunc ThreadFunc, void *
Parameter, LongWord CreationFlags, LongWord ThreadId);
```

File

System

Description

Use this routine or a TThread object to spawn separate threads of execution. BeginThread spawns a new thread of execution and sets the global IsMultiThread (see page 3897) variable, thereby making the heap thread-safe.

ThreadFunc is the thread function. This is the code that executes in the new thread of execution. ThreadFunc should handle all of its own exceptions. However, because BeginThread sets up an execution frame, the system's default exception handler will catch any exceptions that escape ThreadFunc.

Parameter points to a single 32-bit parameter that is passed, uninterpreted, to ThreadFunc.

ThreadId returns the unique identifier for the new thread. This ID appears in the Thread Status box of the debugger.

SecurityAttributes is a pointer to a Windows SecurityAttributes record (structure). For details on security attributes, see the Microsoft documentation.

StackSize is the size of the stack reserved for the new thread when CreationFlags includes STACK_SIZE_IS_A_RESERVATION. Otherwise, this parameter is ignored.

CreationFlags is a set of bits or'ed together that are drawn from the following:

Value	Meaning
CREATE_SUSPENDED	The thread is created in a suspended state and does not run until the Windows API ResumeThread function is called. If this flag is not included, the new thread starts running as soon as it is created.
STACK_SIZE_PARAM_IS_A_RESERVATION	The StackSize parameter specifies the stack size for the thread. If this flag is not included, the new thread is assigned a default stack size by the linker.

On Windows, BeginThread returns the Windows thread handle. A return value of 0 indicates failure.

Attribute is a structure containing information about the thread such as its priority, its schedule policy, and its stack size.

On Linux, Begin Thread returns 0 if the thread is successfully spawned, an error code otherwise.

See Also

EndThread (see page 3657)

IsMultiThread (see page 3897)

Defining Thread Objects

38.3.318 System.BlockRead Function

Reads one or more records from an open file into a variable.

Pascal

```
procedure BlockRead(var F: File; var Buf; Count: Integer); overload;
procedure BlockRead(var F: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockRead(File F,  Buf, int Count);
BlockRead(File F,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockRead reads Count or fewer records from the file F into memory, starting at the first byte occupied by Buf. The actual number of complete records read (less than or equal to Count) is returned in AmtTransferred.

The entire transferred block occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was not specified).

If the entire block was transferred, AmtTransferred is equal to Count.

If AmtTransferred is less than Count, ReadBlock reached the end of the file before the transfer was complete. If the file's record size is greater than 1, AmtTransferred returns the number of complete records read.

If AmtTransferred isn't specified, an I/O error occurs if the number of records read isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
```

```
      if SaveDialog1.Execute then      { Display Save dialog box}
      begin
        AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
        Rewrite(ToF, 1);  { Record size = 1 }
        Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
          + ' bytes...');
        repeat
          System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
          BlockWrite(ToF, Buf, NumRead, NumWritten);
        until (NumRead = 0) or (NumWritten <> NumRead);
        CloseFile(FromF);
        CloseFile(ToF);
        Canvas.TextOut(120, 10, ' done.');
      end;
    end;
  end;
  {
  The following example opens a file of your choice and reads
  the entire file into a dynamically allocated buffer. The
  buffer and the size of the file are then passed to a routine
  that processes the text, and finally the dynamically
  allocated buffer is freed and the file is closed.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: file;
    Size: Integer;
    Buffer: PAnsiChar;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(AnsiString(Buffer));
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

TFileStream

EInOutError

38.3.319 System.BlockWrite Function

Writes one or more records from a variable to an open file.

Pascal

```
procedure BlockWrite(var f: File; var Buf; Count: Integer); overload;
procedure BlockWrite(var f: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockWrite(File f,  Buf, int Count);
BlockWrite(File f,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockWrite writes Count or fewer records to the file F from memory, starting at the first byte occupied by Buf. The actual number of complete records written (less than or equal to Count) is returned in AmtTransferred.

The entire block transferred occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was unspecified).

If the entire block is transferred, AmtTransferred is equal to Count on return.

If AmtTransferred is less than Count, the disk became full before the transfer was complete. In this case, if the file's record size is greater than 1, AmtTransferred returns the number of complete records written.

BlockWrite advances the current file position by AmtTransferred records.

If AmtTransferred isn't specified, an I/O error occurs if the number written isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
```

See Also

TFileStream

EInOutError

38.3.320 System.Break Function

Causes the flow of control to exit a for, while, or repeat statement.

Pascal

```
procedure Break;
```

C++

```
Break();
```

File

System

Description

The Break procedure causes the flow of control in Delphi code to exit a for, while, or repeat statement and continue at the next statement following the loop statement.

A call to Break must be contained in a for, while, or repeat statement, or the compiler reports an error.

Note: Break does not violate the flow of control dictated by a try..finally construct. If a break occurs inside the try clause, the finally clause is entered.

Delphi Examples:

```
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
    begin
      SelStart := SelSt;
      SelLength := Length(Text) - SelSt;
    end
    else Beep;
```

```
  end;
end;

var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  ComboBox1.Items := CharSetList;
end;
```

See Also

Halt (see page 3693)

38.3.321 System.ChDir Function

Changes the current directory.

Pascal

```
procedure ChDir(const S: string); overload;
procedure ChDir(P: PChar); overload;
```

C++

```
ChDir(const UnicodeString S);
ChDir(const char * P);
```

File

System

Description

Note: ChDir changes the current directory to the path specified by S or P. If this operation fails, EInOutError is raised.

Note: On Windows, the path can include a drive specifier, which causes the current drive to be changed as well.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for an I/O error.

C++ Examples:

```
/*
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
```

```
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
*/
void __fastcall TMyChDir::Button1Click(TObject *Sender)
{
  try
  {
    System::ChDir(Edit1->Text);
  }
  catch (...)
  {
    MessageDlg("Cannot find directory", mtWarning, TMsgDlgButtons() << mbOK, 0);
  }
  Edit2->Text = GetCurrentDir();
  FileListBox1->Directory = GetCurrentDir();
  DirectoryListBox1->Directory = GetCurrentDir();
}

void __fastcall TMyChDir::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}

void __fastcall TMyChDir::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}
```

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
```

```
    FileListBox1.Directory := DirectoryListBox1.Directory;
    Edit2.Text := GetCurrentDir;
  end;
```

See Also

GetDir (see page 3679)

MkDir (see page 3708)

RmDir (see page 3738)

SetCurrentDir

EInOutError

38.3.322 System::CheckSafecallResult Function

Pascal

```
procedure CheckSafecallResult(hr: HRESULT);
```

C++

```
__fastcall void CheckSafecallResult(HRESULT hr);
```

File

sysclass.h

Description

This is function System::CheckSafecallResult.

38.3.323 System.Chr Function

Returns the character for a specified ASCII value.

Pascal

```
function Chr(X: Byte): Char;
```

C++

```
Char Chr(Byte X);
```

File

System

Description

Chr returns the character with the ordinal value (ASCII value) of the byte-type expression, X.

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
```

```
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
    begin
      SelStart := SelSt;
      SelLength := Length(Text) - SelSt;
    end
    else Beep;
  end;
end;

var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  ComboBox1.Items := CharSetList;
end;
```

See Also

Ord (see page 3716)

38.3.324 System.Close Function

Terminates the association between a file variable and an external file.

Pascal

```
procedure Close(var F);
```

C++

```
Close( F);
```

File

System

Description

Close is provided for backward compatibility with existing code. When writing new applications in Delphi, use CloseFile (see page 3643) instead.

F is a file variable of any file type opened using Reset (see page 3734), Rewrite (see page 3736), or Append (see page 3624). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

See Also

Append (see page 3624)

AssignFile (see page 3629)

Reset (see page 3734)

Rewrite (see page 3736)

CloseFile (see page 3643)

38.3.325 System.CloseFile Function

Terminates the association between file variable and an external disk file.

Pascal

```
procedure CloseFile(var F);
```

C++

```
CloseFile( F);
```

File

System

Description

Due to naming conflicts, CloseFile replaces the Close (see page 3643) procedure. Use the CloseFile procedure instead of Close (see page 3643) to terminate the association between a file variable and an external disk file.

F is a file variable of any file type opened using Reset (see page 3734), Rewrite (see page 3736), or Append (see page 3624). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.  Click on the text edit to open the
open dialog box.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
```

```
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

AssignFile (see page 3629)

38.3.326 System.CompareTo Function

Pascal

```
function CompareTo(Value: T): Integer;
```

C++

```
int CompareTo(T Value);
```

File

System

Description

This is function System.CompareTo.

38.3.327 System.CompToCurrency Function

Converts a Comp value to a Currency (see page 3242) value.

Pascal

```
function CompToCurrency(Value: Comp): Currency; cdecl;
```

C++

```
__cdecl Currency CompToCurrency(Comp Value);
```

File

System

Description

CompToCurrency converts a Comp value into a Currency (see page 3242) value containing the decimal representation of that number.

See Also

CompToDouble (see page 3646)

CurrencyToComp (see page 3651)

DoubleToComp (see page 3656)

38.3.328 System.CompToDouble Function

Converts a Comp value to a double value.

Pascal

```
function CompToDouble(Value: Comp): Double; cdecl;
```

C++

```
__cdecl Double CompToDouble(Comp Value);
```

File

System

Description

CompToDouble converts a Comp value into a double value containing the decimal representation of that number.

See Also

CompToCurrency (see page 3645)

CurrencyToComp (see page 3651)

DoubleToComp (see page 3656)

38.3.329 System.Concat Function

Concatenates two or more strings into one.

Pascal

```
function Concat(s1: string): string; overload;
function Concat(s1: string; s2: string): string; overload;
function Concat(s1: string; s2: string; s3: string): string; overload;
```

C++

```
UnicodeString Concat(UnicodeString s1);
UnicodeString Concat(UnicodeString s1, UnicodeString s2);
UnicodeString Concat(UnicodeString s1, UnicodeString s2, UnicodeString s3);
```

File

System

Description

In Delphi code, use Concat to concatenate an arbitrary number of strings. Each parameter is a string-type expression. The result is the concatenation of all the string parameters.

The syntax shown here for Concat demonstrates that the procedure can take a variable number of arguments.

Using the plus (+) operator has the same effect on two strings as using the Concat function:

S := 'ABC' + 'DEF';

Tip: The plus operator is faster than Concat.

Delphi Examples:

```
{
This example requires a button and three text fields.
Select the button to concatenate the two texts into
the third.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  Edit3.Text := Concat(Edit1.Text, Edit2.Text);     { 'ABCDEF' }
end;
```

See Also

Copy (see page 3648)

Delete (see page 3653)

Insert (see page 3698)

Length (see page 3702)

Pos (see page 3720)

38.3.330 System.Continue Function

Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.

Pascal

```
procedure Continue;
```

C++

```
Continue();
```

File

System

Description

In Delphi code, the Continue procedure causes the flow of control to proceed to the next iteration of the enclosing for, while, or repeat statement.

The compiler reports an error if a call to Continue isn't enclosed by a for, while, or repeat statement.

Note: Continue does not violate the flow of control dictated by a try..finally construct. If a call to Continue causes control to leave the try clause, the finally clause is entered.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
```

See Also

Break (see page 3636)

Exit (see page 3665)

Halt (see page 3693)

Abort

38.3.331 System.Copy Function

Returns a substring of a string or a segment of a dynamic array.

Pascal

```
function Copy(S; Index: Integer; Count: Integer): array; overload;
```

C++

```
array Copy( S, int Index, int Count);
```

File

System

Description

S is an expression of a string or dynamic-array type. Index and Count are integer-type expressions. Copy returns a substring or

subarray containing Count characters or elements starting at S[Index]. The substring or subarray is a unique copy (that is, it does not share memory with S, although if the elements of the array are pointers or objects, these are not copied as well.)

If Index is larger than the length of S, Copy returns an empty string or array.

If Count specifies more characters or array elements than are available, only the characters or elements from S[Index] to the end of S are returned.

Note: When S is a dynamic array, you can omit the Index and Count parameters and Copy copies the entire array.

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
```

```
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

Concat (see page 3646)

Delete (see page 3653)

Insert (see page 3698)

Length (see page 3702)

Pos (see page 3720)

Slice (see page 3762)

LeftStr

RightStr

Parameters

38.3.332 System.Cos Function

Calculates the cosine of an angle.

Pascal

```
function Cos(const X: Extended): Extended;
```

C++

```
Extended Cos(const Extended X);
```

File

System

Description

In Delphi code, Cos returns the cosine of the angle X.

X is a real-type expression that represents an angle in radians.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

ArcCos

Sin (see page 3760)

Tan

Cosh

38.3.333 System.CurrencyToComp Function

Converts a Currency (see page 3242) value to a Comp value.

Pascal

```
procedure CurrencyToComp(Value: Currency; var Result: Comp); cdecl;
```

C++

```
__cdecl CurrencyToComp(Currency Value, Comp Result);
```

File

System

Description

CurrencyToComp converts a Currency (see page 3242) value into a Comp.

See Also

CompToDouble (see page 3646)

CompToCurrency (see page 3645)

DoubleToComp (see page 3656)

38.3.334 System.Dec Function

Decrements a variable by 1 or N.

Pascal

```
procedure Dec(var X); overload;
procedure Dec(var X; N: Longint); overload;
```

C++

```
Dec( X);
Dec( X, Longint N);
```

File

System

Description

In Delphi code, Dec subtracts one or N from a variable.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X decrements by 1, or by N if N is specified; that is, Dec(X) corresponds to the statement X := X - 1, and Dec(X, N) corresponds to the statement X := X - N. On some platforms, Dec may generate optimized code, especially useful in tight loops.

If X is a pointer type, it decrements X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Dec(P)` decrements `P` by `SizeOf (see page 3761)(TMyType)`.

Warning: Do not use Dec on ordinal-type properties if the property uses a write procedure.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
```

```
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

Inc (see page 3696)

Pred (see page 3721)

Succ (see page 3768)

38.3.335 System.Delete Function

Removes a substring from a string.

Pascal

```
procedure Delete(var S: string; Index: Integer; Count: Integer);
```

C++

```
Delete(UnicodeString S, int Index, int Count);
```

File

System

Description

In Delphi code, Delete removes a substring of Count characters from string S starting with S[Index]. S is a string-type variable. Index and Count are integer-type expressions.

If index is larger than the length of the string or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, Delete removes the rest of the string. If count is less than or equal to 0, no characters are deleted.

Delphi Examples:

```
{
This example uses the Delete function to edit a string.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  s: string;
begin
  s := 'Honest Abe Lincoln';
```

```
  Canvas.TextOut(10, 10, s);
  Delete(s,8,4);
  Canvas.TextOut(10, 40, s);    { 'Honest Lincoln' }
end;
```

See Also

Concat (see page 3646)

Copy (see page 3648)

Insert (see page 3698)

Length (see page 3702)

Pos (see page 3720)

38.3.336 System.Dispose Function

Releases memory allocated for a dynamic variable.

Pascal

```
procedure Dispose(var P: Pointer);
```

C++

```
Dispose(void * P);
```

File

System

Description

Use Dispose in Delphi code to free the memory which a pointer addresses. After a call to Dispose, the value of P is undefined and it is an error to reference P.

Note: If compiling with {$I+}, use exceptions to handle this error. When an invalid pointer is passed to Dispose it raises an EInvalidPointer exception.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
```

```
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      ARecord := TheList.Items[B];
      Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
  DisplayTList(MyList);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
  DisplayTList(MyList);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  ARecord: PMyList;
  B: Integer;
begin
  if (MyList.Count <> 0) then
  begin
    for B := (MyList.Count - 1) downto 0 do
    begin
      ARecord := MyList.Items[B];
      MyList.Remove(ARecord);
    end;
  end;
  Form1.PaintBox1.Repaint;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TList.Create;
  New(ARecord1);
  ARecord1^.I := 100;
  ARecord1^.C := 'Z';
```

```
    New(ARecord2);
    ARecord2^.I := 200;
    ARecord2^.C := 'X';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Dispose(ARecord1);
    Dispose(ARecord2);
    MyList.Free;
  end;
```

See Also

EInvalidPointer

FreeMem (see page 3676)

GetMem (see page 3683)

New (see page 3712)

38.3.337 System.DoubleToComp Function

Converts a double value to a Comp.

Pascal

```
procedure DoubleToComp(Value: Double; var Result: Comp); cdecl;
```

C++

```
__cdecl DoubleToComp(Double Value, Comp Result);
```

File

System

Description

DoubleToComp converts a Double value into a Comp.

See Also

CompToDouble (see page 3646)

CompToCurrency (see page 3645)

CurrencyToComp (see page 3651)

38.3.338 System.DynArrayClear Function

Pascal

```
procedure DynArrayClear(var a: Pointer; typeInfo: Pointer);
```

C++

```
DynArrayClear(void * a, void * typeInfo);
```

File

System

Description

This is function System.DynArrayClear.

38.3.339 System.DynArrayDim Function

Pascal

```
function DynArrayDim(typeInfo: PDynArrayTypeInfo): Integer;
```

C++

```
int DynArrayDim(PDynArrayTypeInfo typeInfo);
```

File

System

Description

This is function System.DynArrayDim.

38.3.340 System.DynArraySetLength Function

Pascal

```
procedure DynArraySetLength(var a: Pointer; typeInfo: Pointer; dimCnt: Longint; lengthVec:
PLongint);
```

C++

```
DynArraySetLength(void * a, void * typeInfo, Longint dimCnt, PLongint lengthVec);
```

File

System

Description

This is function System.DynArraySetLength.

38.3.341 System.EndThread Function

Terminates the execution of a thread.

Pascal

```
procedure EndThread(ExitCode: Integer);
```

C++

```
EndThread(int ExitCode);
```

File

System

Description

EndThread terminates execution of a thread with the specified ExitCode (see page 3891).

Note: On Windows, EndThread calls the ExitThread WinAPI with the specified ExitCode (see page 3891).

See Also

BeginThread (see page 3632)

TThreadFunc (see page 3867)

38.3.342 System.EnumModules Function

Executes a callback for all the modules in an application.

Pascal

```
procedure EnumModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumModules(TEnumModuleFunc Func, void * Data);
EnumModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumModules enumerates the executable and all packages in an application by passing the handle of the executable and then of each package, in turn, to a user-defined callback function. EnumModules continues until the last module in the application is enumerated, or until the callback function returns false. The parameters to EnumModules areas follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumResourceModules (see page 3658)

LibModuleList (see page 3898)

Hinstance

38.3.343 System.EnumResourceModules Function

Executes a callback for all resource modules in an application.

Pascal

```
procedure EnumResourceModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumResourceModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumResourceModules(TEnumModuleFunc Func, void * Data);
EnumResourceModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumResourceModules executes the callback specified by the Func parameter for all of the instance handles associated with resources of the current program. The instance handle (HInstance) of each resource is passed to Func. EnumResourceModules continues until the last resource module in the application is enumerated, or until the callback returns false. The parameters to EnumResourceModules are as follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumModules (see page 3658)

LibModuleList (see page 3898)

Hinstance

38.3.344 System.Eof Function

Tests whether the file position is at the end of a file.

Pascal

```
function Eof: Boolean; overload;
function Eof(var F): Boolean; overload;
function Eof(var F: Text): Boolean; overload;
```

C++

```
Boolean Eof();
Boolean Eof( F);
Boolean Eof(Text F);
```

File

System

Description

In Delphi code, Eof tests whether the current file position is the end-of-file. F is a file variable that has been opened for reading. If F is omitted, the standard file variable Input (see page 3895) is assumed.

Eof(F) returns true if the current file position is beyond the last character of the file or if the file is empty; otherwise, Eof(F) returns false.

Note: Eof fails if the file F has been opened in write-only mode. For example, you can't use Eof with files opened using the Append (see page 3624) or Rewrite (see page 3736), which open a file in write-only mode.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

DeleteFile

Eoln (see page 3660)

SeekEof (see page 3742)

SetLineBreakStyle (see page 3750)

38.3.345 System.Eoln Function

Tests whether the file pointer is at the end of a line.

Pascal

```
function Eoln: Boolean; overload;
function Eoln(var F: Text): Boolean; overload;
```

C++

```
Boolean Eoln();
Boolean Eoln(Text F);
```

File

System

Description

In Delphi code, Eoln tests whether the current file position is the end-of-line of a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Input (see page 3895) is assumed.

Eoln(F) returns true if the current file position is at an end-of-line or if Eof (see page 3659)(F) is true; otherwise, Eoln(F) returns false.

Delphi Examples:

```
{
This example waits for keyboard input before continuing.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   WriteLn('Type something on the console and hit return');
   Eoln;
   WriteLn('Write this after you type something');
   Readln;
end;
```

See Also

Eof (see page 3659)

SeekEoln (see page 3743)

SetLineBreakStyle (see page 3750)

38.3.346 System.Equals Function

Pascal

```
function Equals(Value: T): Boolean;
```

C++

```
Boolean Equals(T Value);
```

File

System

Description

This is function System.Equals.

38.3.347 System.Erase Function

Deletes an external file.

Pascal

```
procedure Erase(var F);
```

C++

```
Erase( F);
```

File

System

Description

In Delphi code, Erase deletes the external file associated with F. F is a file variable of any file type.

Always close a file before erasing it.

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

Rename (see page 3733)

38.3.348 System.Error Function

Error is used to generate a run-time exception.

Pascal

```
procedure Error(errorCode: TRuntimeError);
```

C++

```
Error(TRuntimeError errorCode);
```

File

System

Description

Error should be invoked when you want to generate an exception. **errorCode** defines the type of error to be reported.

Delphi Examples:

```
{
This example demostrates the use of ErrorProc and how to catch runtime
errors.
}
procedure MyErrorProc(ErrorCode : Byte; ErrorPtr : Pointer);
var
  ErrorName : String;
begin
  { Get the "name" of the enum value using RTTI }
  ErrorName := GetEnumName(TypeInfo(TRuntimeError), ErrorCode);
```

```
    { Show an error box }
    MessageDlg(Format('Error with code %d (%s) risen at $%X.',
        [ErrorCode, ErrorName, Integer(ErrorPtr)]), mtError, [mbOK], 0);
  end;

  procedure TForm2.btIOErrorClick(Sender: TObject);
  begin
    {
    Try to write something onto the console - will raise an
    exception.
    }
    WriteLn('This will generate an error because there is no' +
            ' console attached!');
  end;

  {$OVERFLOWCHECKS ON}
  {$OPTIMIZATION OFF}
  {$HINTS OFF}
  procedure TForm2.btOverflowErrClick(Sender: TObject);
  var
    b : Cardinal;
  begin
    {
    Simulate an overflow: Note, enabled the overflow
    checking and disabled optimizations because Delphi's
    compiler will not compile this code otherwise.
    }
    b := $FFFFFFFF;
    b := b * b;
  end;
  {$HINTS ON}
  {$OPTIMIZATION ON}
  {$OVERFLOWCHECKS OFF}

  procedure TForm2.btRuntimeErrorClick(Sender: TObject);
  begin
    { Simulate an access violation }
    System.Error(reAccessViolation);
  end;

  procedure TForm2.FormCreate(Sender: TObject);
  begin
    { Register my System error proc }
    ErrorProc := MyErrorProc;
  end;
```

See Also

TRunTimeError

38.3.349 System.ExceptAddr Function

Returns the address at which the current exception was raised.

Pascal

```
function ExceptAddr: Pointer;
```

C++

```
void * ExceptAddr();
```

File

System

Description

Use ExceptAddr to obtain the address at which an exception was raised. If there is no current exception, ExceptAddr returns nil (Delphi) or NULL (C++).

See Also

ExceptionErrorMessage

ExceptObject (see page 3664)

38.3.350 System.ExceptObject Function

Returns current exception object.

Pascal

```
function ExceptObject: TObject;
```

C++

```
TObject * ExceptObject();
```

File

System

Description

ExceptObject returns the current exception object. This is the Exception object that represents the exception currently being handled. If no exception is currently being handled, ExceptObject returns nil (Delphi) or NULL (C++).

ExceptObject is useful when the exception variable (declared in the Delphi on block or C++ catch statement) is not accessible. This can occur when the exception handler calls a procedure, putting the exception variable is out of scope. This is also true in the Delphi else exception handler, which cannot declare an exception variable.

Note: The object returned by ExceptObject may not exist after the exception handler finishes. Thus you cannot use ExceptObject to retain a reference to the exception object, or to re-raise the exception. If you need to do either of these things, call AcquireExceptionObject (see page 3619).

See Also

ExceptionErrorMessage

Exception

on

catch

38.3.351 System.Exclude Function

Removes an element from a Delphi set or removes an event handler in .NET applications.

Pascal

```
procedure Exclude(var S: set of T; I: T);
```

C++

```
Exclude(set of T S, T I);
```

File

System

Description

When the first argument of Include (see page 3697) is a set type variable, the Exclude procedure removes an element from the set. If S is a set type variable, and I is an expression of a type compatible with the base type of S. then `Exclude(S,I)` corresponds to `S := S - [I]` but the Exclude procedure generates more efficient code.

On the .NET platform, Exclude is overloaded. When the first argument of Exclude is a multicast event and the second argument is an event handler, Include (see page 3697) removes the event handler from the event's listeners.

See Also

Include (see page 3697)

38.3.352 System.Exit Function

Exits from the current procedure.

Pascal

```
procedure Exit;
```

C++

```
Exit();
```

File

System

Description

In Delphi, the Exit procedure immediately passes control away from the current procedure. If the current procedure is the main program, Exit causes the program to terminate.

Exit will cause the calling procedure to continue with the statement after the point at which the procedure was called.

Note: Exit passes control away from the current procedure, not merely the current block. But Exit does not violate the flow of control dictated by a try..finally construct; if Exit is called inside the try clause, the finally clause is still executed.

Beginning in Delphi 2009, Exit can take a parameter specifying a result. The parameter must be of the same type as the result of the function. For example:

```
function DoSomething(aInteger: integer): string
begin
if aInteger < 0 then
   begin
      Exit('Negative');
   end;
   Result := 'Positive';
end;
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
    begin
      SelStart := SelSt;
      SelLength := Length(Text) - SelSt;
    end
    else Beep;
  end;
end;

var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
```

```
begin
  CharSetList.Add(S);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  ComboBox1.Items := CharSetList;
end;
```

See Also

Break (see page 3636)

Halt (see page 3693)

Continue (see page 3647)

38.3.353 System.Exp Function

Returns the exponential of X.

Pascal

```
function Exp(X: Real): Real;
function Exp(const X: Extended): Extended;
```

C++

```
Real Exp(Real X);
Extended Exp(const Extended X);
```

File

System

Description

In Delphi code, Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Ln (see page 3703)

38.3.354 System.FilePos Function

Returns the current file position.

Pascal

```
function FilePos(var F): Longint;
```

C++

```
Longint FilePos( F);
```

File

System

Description

In Delphi code, use FilePos on the file variable of an open file to determine the current position in the file. If the current position in the file is at the very beginning, FilePos returns 0. Otherwise, FilePos returns the byte offset from the beginning of the current position.

FilePos will not work on a file that is not open.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
```

```
      CloseFile(f);
    end;
  end;
end;
```

See Also

FileSize (see page 3669)

Seek (see page 3741)

38.3.355 System.FileSize Function

Returns the number of records in a file.

Pascal

```
function FileSize(var F): Integer;
```

C++

```
int FileSize( F);
```

File

System

Description

In Delphi code, call FileSize to determine the size of the file specified by the file variable F. The size is expressed as the number of records in a record file. Thus:

If the file is declared as a file of byte, then the record size defaults to one byte, and FileSize returns the number of bytes in the file.

The Reset (see page 3734) procedure can set the record size (in bytes) when it opens the file. In this case, FileSize returns the number of records in the file.

Note: If the file is declared as an untyped file and you don't specify a record size when you call Reset (see page 3734), then FileSize assumes a record size of 128. That is, FileSize gives the number of bytes divided by 128.

To use FileSize, the file must be open. If the file is empty, FileSize(F) returns 0.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
```

```
      AssignFile(f, OpenDialog1.FileName);
      Reset(f);
      try
        size := FileSize(f);
        S := 'File size in bytes: ' + IntToStr(size);
        y := 10;
        Canvas.TextOut(5, y, S);
        y := y + Canvas.TextHeight(S) + 5;
        S := 'Seeking halfway into file...';
        Canvas.TextOut(5, y, S);
        y := y + Canvas.TextHeight(S) + 5;
        Seek(f, size div 2);
        S := 'Position is now ' + IntToStr(FilePos(f));
        Canvas.TextOut(5, y, S);
      finally
        CloseFile(f);
      end;
    end;
  end;
  {
  The following example opens a file of your choice and reads
  the entire file into a dynamically allocated buffer. The
  buffer and the size of the file are then passed to a routine
  that processes the text, and finally the dynamically
  allocated buffer is freed and the file is closed.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: file;
    Size: Integer;
    Buffer: PAnsiChar;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(AnsiString(Buffer));
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

FilePos (see page 3668)

Reset (see page 3734)

38.3.356 System.FillChar Function

Fills contiguous bytes with a specified value.

Pascal

```
procedure FillChar(var X; Count: Integer; Value: Byte);
```

C++

```
FillChar( X, int Count, Byte Value);
```

File

System

Description

In Delphi, FillChar fills Count contiguous bytes (referenced by X) with the value specified by Value (Value can be type Byte or Char).

Warning: This function does not perform any range checking.

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..79] of AnsiChar;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
```

See Also

Move (see page 3711)

StringOfChar (see page 3766)

38.3.357 System.Finalize Function

Uninitializes a dynamically allocated variable.

Pascal

```
procedure Finalize(var V); overload;
procedure Finalize(var V; Count: Integer); overload;
```

C++

```
Finalize( V);
Finalize( V, int Count);
```

File

System

Description

Finalize should be used only in Delphi code where a dynamically allocated variable is deallocated by other means than the Dispose (see page 3654) procedure. Dynamic arrays can never be deallocated using the Dispose (see page 3654) procedure, but can be freed by passing them to Finalize.

For global variables, local variables, objects, and dynamic variables deallocated using Dispose (see page 3654), the compiler generates code that finalizes all long strings, variants, and interfaces contained by the variable when the instance is destroyed.

If a dynamic variable meets the following two conditions, a call to Finalize is required to finalize the variable before it can be deallocated.

The variable is deallocated by other means than the Dispose (see page 3654) standard procedure (for example using FreeMem (see page 3676)).

The variable contains long strings, variants, or interfaces, not all of which are empty or Unassigned.

Finalize simply sets all long strings to empty and all variants and interfaces to Unassigned, thus properly releasing any memory that was referenced by the long strings and variants.

In cases where several variables are deallocated in a contiguous memory block such as a dynamically allocated array of strings, the additional Count parameter can be specified to finalize all variables in one operation.

If the variable specified in a call to Finalize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

See Also

Initialize (see page 3698)

38.3.358 System.FindClassHInstance Function

Returns the instance handle for the module in which a class type is defined.

Pascal

```
function FindClassHInstance(ClassType: TClass): LongWord;
```

C++

```
LongWord FindClassHInstance(TClass ClassType);
```

File

System

Description

FindClassHInstance finds the HInstance of the executable or library that "owns", or contains, the runtime type information (RTTI) and the Virtual Method Table (VMT) for the specified class passed in as the Class Type parameter. When reading a component (which is a type), this information is needed in order to be able to load the form file (resources) associated with that component. The component streaming system uses FindClassHInstance for this purpose.

FindClassHInstance returns 0 if the instance is not found. Otherwise it returns the class owner's HInstance.

See Also

FindResourceHInstance (see page 3674)

HInstance

38.3.359 System::FindDynaInst Function

Pascal

```
function FindDynaInst(var vmt; ID: Integer): Pointer;
```

C++

```
__fastcall void * FindDynaInst(void * vmt, int ID);
```

File

sysclass.h

Description

This is function System::FindDynaInst.

38.3.360 System.FindHInstance Function

Returns the instance handle of the module that contains a specified address.

Pascal

```
function FindHInstance(Address: Pointer): LongWord;
```

C++

```
LongWord FindHInstance(void * Address);
```

File

System

Description

Use FindHInstance to obtain the HInstance of the module that "owns", or contains, the address specified by the Address parameter. If no HInstance is found for that address, FindHInstance returns 0.

See Also

FindClassHInstance (see page 3672)

FindResourceHInstance (see page 3674)

MainInstance (see page 3898)

HInstance

38.3.361 System.FindResourceHInstance Function

Returns the instance handle of the resource module associated with a specified HInstance.

Pascal

```
function FindResourceHInstance(Instance: LongWord): LongWord;
```

C++

```
LongWord FindResourceHInstance(LongWord Instance);
```

File

System

Description

FindResourceHInstance finds and returns the instance handle of the resource module associated with the executable, library, or package identified by the Instance parameter.

FindResourceHInstance returns 0 upon failure.

Note: The Instance parameter must be a valid instance handle.

See Also

FindClassHInstance (see page 3672)

FindHInstance (see page 3673)

38.3.362 System.Flush Function

Empties the buffer of a text file opened for output.

Pascal

```
function Flush(var t: Text): Integer;
```

C++

```
int Flush(Text t);
```

File

System

Description

F is a text file variable.

When a text file is opened for output using Rewrite (see page 3736) or Append (see page 3624), Flush empties the file's buffer. This guarantees that all characters written to the file at that time have actually been written to the external file. Flush has no effect on files opened for input.

Flush returns 0 if the operation succeeded. Otherwise, it returns the error code. When compiled using the {$I-} flag, the IOResult

(see page 3700) method returns this value.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
Click the button and select a file to append a string to the bottom of the file.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

Append (see page 3624)

Rewrite (see page 3736)

38.3.363 System.FPower10 Function

Pascal

```
procedure FPower10;
```

C++

```
FPower10();
```

File

System

Description

This is function System.FPower10.

38.3.364 System.Frac Function

Returns the fractional part of a real number.

Pascal

```
function Frac(const X: Extended): Extended;
```

C++

```
Extended Frac(const Extended X);
```

File

System

Description

In Delphi code, the Frac function returns the fractional part of the argument X.

X is a real-type expression. The result is the fractional part of X; that is, Frac(X) = X - Int (see page 3699)(X).

Delphi Examples:

```
{
This example illustrates the use of the System Frac function.
The number in the text edit can be a negative or positive
real number.
Notice how the type passed (R) matters.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 R: Extended;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 R: Real;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;
```

See Also

Int (see page 3699)

38.3.365 System.FreeMem Function

FreeMem frees a memory block.

Pascal

```
procedure FreeMem(var P: Pointer); overload;
procedure FreeMem(var P: Pointer; Size: Integer); overload;
```

C++

```
FreeMem(void * P);
FreeMem(void * P, int Size);
```

File

System

Description

GetMem (see page 3683) allocates a block on the heap. To dispose of the buffer, use FreeMem.

Note: FreeMemory (see page 3678) is the C++ compatible version of FreeMem.

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
  Buffer: PAnsiChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(AnsiString(Buffer));
      finally
```

```
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
end;
```

See Also

AllocMem (see page 3622)

FreeMemory (see page 3678)

GetMem (see page 3683)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

ReallocMemory (see page 3730)

38.3.366 System.FreeMemory Function

FreeMemory frees a memory block.

Pascal

```
function FreeMemory(P: Pointer): Integer; cdecl;
```

C++

```
__cdecl int FreeMemory(void * P);
```

File

System

Description

GetMemory (see page 3684) allocates a block on the heap. To dispose of the buffer, use FreeMemory.

Note: FreeMemory is the C++ compatible version of FreeMem (see page 3676).

See Also

AllocMem (see page 3622)

FreeMem (see page 3676)

GetMem (see page 3683)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

ReallocMemory (see page 3730)

38.3.367 System.Get8087CW Function

Returns the value of the 8087 code word.

Pascal

```
function Get8087CW: Word;
```

C++

```
Word Get8087CW();
```

File

System

Description

Use Get8087CW to determine the current value of the floating-point unit control word. This control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

See Also

NoErrMsg (see page 3901)

SetExceptionMask

SetPrecisionMode

SetRoundMode

Set8087CW (see page 3744)

38.3.368 System.GetCurrent Function

Pascal

```
function GetCurrent: T;
```

C++

```
T GetCurrent();
```

File

System

Description

This is function System.GetCurrent.

38.3.369 System.GetDir Function

Returns the current directory.

Pascal

```
procedure GetDir(D: Byte; var S: string);
```

C++

```
GetDir(Byte D, UnicodeString S);
```

File

System

Description

Note: On Windows, GetDir sets the S parameter to the current directory on the drive specified by the D parameter. D can be set to any of the following values:

Value	Drive
0	Default
1	A
2	B
3	C

Note: GetDir performs no error checking. For example, if the invalid value 5 (J) were specified, S would return J:\ as if it were the root directory of the invalid drive.

Note: On Linux, GetDir returns the current working directory using the S parameter. The first parameter is ignored.

Delphi Examples:

```
{
This example reports the current directory and drive.  Set
Help on GetDir for reporting information on other drives.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  GetDir(0,s); { 0 = Current drive }
  MessageDlg('Current drive and directory: ' + s, mtInformation, [mbOk] , 0);
end;
```

See Also

ChDir (see page 3638)

MkDir (see page 3708)

RmDir (see page 3738)

DirectoryExists

CreateDir

GetCurrentDir

38.3.370 System.GetEnumerator Function

Pascal

```
function GetEnumerator: IEnumerator;
```

C++

```
IEnumerator GetEnumerator();
```

File

System

Description

This is function System.GetEnumerator.

38.3.371 System.GetHeapStatus Function

Returns the current status of the memory manager.

Pascal

```
function GetHeapStatus: THeapStatus; deprecated; platform;
```

C++

```
THeapStatus GetHeapStatus();
```

File

System

Description

GetHeapStatus returns the current status of the memory manager in a THeapStatus (see page 3802) type. If the ShareMem (see page 359) unit is not included in the current project, GetHeapStatus returns information about the global heap. If the ShareMem (see page 359) unit is included, GetHeapStatus returns information about memory that can be shared between between the current project and other binary modules (such as DLLs) that also make use of the ShareMem (see page 359) unit. GetHeapStatus can only return useful information if the built-in memory manager is used. For example, when using CodeGuard, the memory manager is replaced and GetHeapStatus always returns nil (Delphi) or NULL (C++).

If you want information about the memory manager that is shared between modules, make sure to call the function GetHeapStatus provided in the ShareMem (see page 359) unit. The function GetHeapStatus in the System (see page 3165) unit returns no values for a project that uses shared memory.

Warning: ShareMem (see page 359) must be the first unit included in the project file to prevent mixing of memory management models.

38.3.372 System.GetLastError Function

Returns the last error reported by an operating system API call.

Pascal

```
function GetLastError: Integer; stdcall;
```

C++

```
__stdcall int GetLastError();
```

File

System

Description

GetLastError returns the last error reported by an API call into the operating system. Calling this function usually resets the operating system error state.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);

    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
```

```
      on Ex : EOSError do
         MessageDlg('Caught an OS error with code: ' +
            IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
   end;

   { Let the Delphi's Exception dialog pop-up }
   RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

   { Finally set the last error to none }
   System.SetLastError(ERROR_SUCCESS);

   if GetLastError() <> ERROR_SUCCESS then
      MessageDlg('Whoops, something went wrong in the ' +
                 'mean time!', mtError, [mbOK], 0);

   { No exception should be thrown here }
   RaiseLastOSError();
end;
```

See Also

Win32Check

38.3.373 System.GetMem Function

GetMem allocates a memory block.

Pascal

```
procedure GetMem(var P: Pointer; Size: Integer);
```

C++

```
GetMem(void * P, int Size);
```

File

System

Description

GetMem allocates a block of the given **Size** on the heap, and returns the address of this memory in parameter **P**. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 3676). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 3622) instead.

Delphi Examples:

```
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
  Buffer: PAnsiChar;
begin
```

```
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(AnsiString(Buffer));
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

AllocMem (see page 3622)

FreeMem (see page 3676)

FreeMemory (see page 3678)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

ReallocMemory (see page 3730)

38.3.374 System.GetMemory Function

GetMemory allocates a memory block.

Pascal

```
function GetMemory(Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * GetMemory(int Size);
```

File

System

Description

GetMemory allocates a block of the given **Size** on the heap, and returns the address of this memory. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 3676). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 3622) instead.

Note: GetMemory is the C++ compatible version of GetMem (see page 3683).

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
    begin
      { Allocate space }
      MyBuffer:= GetMemory(DataSize);
      try
        if not GetData(MyBuffer) then
          MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
        else
          { Do something with the data };
          Edit1.Text:= string(PAnsiChar(MyBuffer)); // for a stringfield
      finally
        { Free the space }
        FreeMem(MyBuffer, DataSize);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

AllocMem (see page 3622)

FreeMem (see page 3676)

FreeMemory (see page 3678)

GetMem (see page 3683)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

ReallocMemory (see page 3730)

38.3.375 System.GetMemoryManager Function

Returns the entry points of the currently installed memory manager.

Pascal

```
procedure GetMemoryManager(var MemMgr: TMemoryManager); overload; deprecated;
procedure GetMemoryManager(var MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
GetMemoryManager(TMemoryManager MemMgr);
GetMemoryManager(TMemoryManagerEx MemMgrEx);
```

File

System

Description

GetMemoryManager returns the entry points of the currently installed memory manager in a TMemoryManager (see page 3805) record.

C++ Examples:

```
/*
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
*/

bool isMyMemMgr;
TMemoryManagerEx oldMemMgr;
volatile long getMemCalls;
volatile long freeMemCalls;
volatile long reallocMemCalls;
volatile long allocMemCalls;

void* __fastcall myGetMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&getMemCalls);

    //route the call
    return oldMemMgr.GetMem(size);
}

int __fastcall myFreeMem(void* p)
{
    //safely increment the counter
    InterlockedIncrement(&freeMemCalls);

    //route the call
    return oldMemMgr.FreeMem(p);
}

void* __fastcall myReallocMem(void* p, int size)
{
    //safely increment the counter
    InterlockedIncrement(&reallocMemCalls);

    //route the call
    return oldMemMgr.ReallocMem(p, size);
}
```

```
void* __fastcall myAllocMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&allocMemCalls);

    //route the call
    return oldMemMgr.AllocMem(size);
}

void __fastcall TForm3::btUseSysMemMgrClick(TObject *Sender)
{
    //switch button states
    btUseSysMemMgr->Enabled = false;
    btUseMyMemMgr->Enabled = true;

    //set the old memory manager back
    SetMemoryManager(oldMemMgr);

    isMyMemMgr = false;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;
}

void __fastcall TForm3::FormDestroy(TObject *Sender)
{
    //set the old memory manager back
    if (isMyMemMgr)
    {
        SetMemoryManager(oldMemMgr);
    }
}

void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
    //Note that IntToStr calls will also Allocate/Free
    //memory so at each timer tick the counts will be +4;

    Form3->edGetMem->Text = IntToStr((int)getMemCalls);
    Form3->edFreeMem->Text = IntToStr((int)freeMemCalls);
    Form3->edReallocMem->Text = IntToStr((int)reallocMemCalls);
    Form3->edAllocMem->Text = IntToStr((int)allocMemCalls);
}

void __fastcall TForm3::btUseMyMemMgrClick(TObject *Sender)
{
    TMemoryManagerEx myMemMgr;

    //switch button states
    btUseSysMemMgr->Enabled = true;
    btUseMyMemMgr->Enabled = false;

    //get the old memory manager
    GetMemoryManager(oldMemMgr);

    //create out instance
    myMemMgr.GetMem = myGetMem;
    myMemMgr.FreeMem = myFreeMem;
    myMemMgr.AllocMem = myAllocMem;
    myMemMgr.ReallocMem = myReallocMem;

    //use defaults for this - not important
```

```
    myMemMgr.RegisterExpectedMemoryLeak = oldMemMgr.RegisterExpectedMemoryLeak;
    myMemMgr.UnregisterExpectedMemoryLeak = oldMemMgr.UnregisterExpectedMemoryLeak;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;

    //install the new memory manager
    SetMemoryManager(myMemMgr);

    isMyMemMgr = true;
}

void __fastcall TForm3::btDoSomethingClick(TObject *Sender)
{
    void *ptr;
    //perform some new and delete
    for (int i=0; i < 1000; ++i)
    {
        ptr = GetMemory(4);
        FreeMemory(ptr);
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  IsMyMemMgr      : Boolean;
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;

function MyReallocMem(P: Pointer; Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;
```

```
function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Swith button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;

  { Install the new memory manager }
  SetMemoryManager(MyMemMgr);

  IsMyMemMgr := true;
end;

procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
begin
  { Swith button states }
  btUseSystemMemMgr.Enabled := false;
  btUseMyMemMgr.Enabled := true;

  { Set the old memory manager back! }
  SetMemoryManager(OldMemMgr);

  IsMyMemMgr := false;
```

```
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
  { Set the old memory manager back }
  if IsMyMemMgr then
     SetMemoryManager(OldMemMgr);
end;

procedure TForm2.Timer1Timer(Sender: TObject);
begin
  {
  Note that IntToStr calls will also Allocate/Free
  memory so at each timer tick the counts will be +4;
  }
  Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
  Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
  Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
  Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
end;
```

See Also

SetMemoryManager (see page 3751)

IsMemoryManagerSet (see page 3701)

38.3.376 System.GetMemoryManagerState Function

Fetches state information for the Memory Manager.

Pascal

```
procedure GetMemoryManagerState(var AMemoryManagerState: TMemoryManagerState);
```

C++

```
GetMemoryManagerState(TMemoryManagerState AMemoryManagerState);
```

File

System

Description

GetMemoryManagerState returns a structure containing memory usage statistics for each block type - small, medium and large blocks. In the current implementation, small blocks are blocks that are smaller than roughly 2.5K, large blocks are blocks that are larger than roughly 256K, and medium blocks are the sizes in between. It provides additional detail for the small block type (which is usually the most frequently used block type) down to each individual block size.

The structure returned by this procedure makes a distinction between the reserved address space and the allocated address space for each block type. The allocated address space is that part of the address space that is currently in use by the application, i.e. memory that was allocated through GetMem (see page 3683) and not yet freed. The reserved address space is the amount of address space that was obtained from the operating system for current and future use by the application, i.e. the allocated address space plus all overhead.

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

38.3.377 System.GetMemoryMap Function

Fetches the memory map.

Pascal

```
procedure GetMemoryMap(var AMemoryMap: TMemoryMap);
```

C++

```
GetMemoryMap(TMemoryMap AMemoryMap);
```

File

System

Description

GetMemoryMap returns a map of the address space of the process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. GetMemoryMap obtains a status value for each of them in the TMemoryMap (see page 3862) array. The sum of the number of blocks with the status of **csUnallocated** multiplied by the chunk size (64K) gives an indication of how much more memory the application can allocate before it will run out of address space.

Status Value	Description
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

38.3.378 System.GetMinimumBlockAlignment Function

Gets the minimum memory block alignment.

Pascal

```
function GetMinimumBlockAlignment: TMinimumBlockAlignment;
```

C++

```
TMinimumBlockAlignment GetMinimumBlockAlignment();
```

File

System

Description

GetMinimumBlockAlignment fetches the Memory Manager's minimum block alignment.

See Also

Memory Management

Configuring the Memory Manager

38.3.379 System.GetModuleFileName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleFileName(Module: HMODULE; Buffer: PChar; BufLen: Integer): Integer;
```

C++

```
int GetModuleFileName(HMODULE Module, const char * Buffer, int BufLen);
```

File

System

Description

Call GetModuleFileName to obtain the fully qualified name of a module, given its handle. The version of GetModuleFileName defined in the System (see page 3165) unit is available only on Linux. On Windows, use the Windows API with the same name instead.

Module is the handle of the module whose file name is required.

Buffer is a buffer that receives the file name.

BufLen is the number of bytes in Buffer.

GetModuleFileName returns the number of bytes written to Buffer.

Warning: GetModuleFileName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleFileName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

See Also

GetModuleName

38.3.380 System.GetVariantManager Function

Returns the entry points of the routines that define variant behavior.

Pascal

```
procedure GetVariantManager(var VarMgr: TVariantManager); deprecated;
```

C++

```
GetVariantManager(TVariantManager VarMgr);
```

File

System

Description

GetVariantManager returns the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although GetVariantManager is defined in System.hpp, do not use this procedure in C++ code. Working directly with the Variant (see page 3430) manger is not guaranteed to work in C++.

See Also

SetVariantManager (see page 3759)

IsVariantManagerSet (see page 3701)

38.3.381 System.Halt Function

Initiates abnormal termination of a program.

Pascal

```
procedure Halt; overload;
procedure Halt(Exitcode: Integer); overload;
```

C++

```
Halt();
Halt(int Exitcode);
```

File

System

Description

Halt performs an abnormal termination of a program and returns to the operating system.

To perform a normal termination of a Delphi application, call the Terminate method on the global Application object. If the application does not use a unit that provides an Application object, call the Exit (see page 3665) procedure from the main Program block.

Exitcode is an optional expression that specifies an exit code for the program.

Delphi Examples:

```
{
This example halts before the TextOut and terminates the
application.  Do not expect to one step this procedure.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   if 1 = 1 then
     begin
       if 2 = 2 then
```

```
          begin
            if 3 = 3 then
              begin
                Halt(1); { Halt right here! }
              end;
          end;
      end;
    Canvas.TextOut(10, 10, 'This will not be executed');
  end;
```

38.3.382 System.Hi Function

Returns the high-order byte of X as an unsigned value.

Pascal

```
function Hi(X): Byte;
```

C++

```
Byte Hi( X);
```

File

System

Description

In Delphi code, use Hi to obtain the high-order byte of an expression of type Word or Integer.

Note: Hi treats Integer as a 16-bit value, even if it is 32-bit.

Delphi Examples:

```
{
This example demonstrate use of the System Hi function to
store the high byte of a word or integer into a byte.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Hi($1234);
  ShowMessage(Format('High byte of $1234 = $%x', [B]));
end;
```

See Also

Lo (see page 3704)

38.3.383 System.High Function

Returns the highest value in the range of an argument.

Pascal

```
procedure High(X);
```

C++

```
High( X);
```

File

System

Description

In Delphi code, call High to obtain the upper limit of an Ordinal, Array, or ShortString value. The result type is X, or the index type of X.

X is either a type identifier or a variable reference. The type denoted by X, or the type of the variable denoted by X, must be one of the following:

For this type	High returns
Ordinal type (includes Int64)	The highest value in the range of the type
Array type	The highest value within the range of the index type of the array. For empty arrays, High returns –1.
short string type	The declared size of the string
Open array	The value, of type Integer, giving the number of elements in the actual parameter minus one
short string parameter	The value, of type Integer, giving the number of elements in the actual parameter minus one

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;

function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0; { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, TempStr: string;
begin
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);

end;

procedure TForm1.FormCreate(Sender: TObject);
var
  X: Word;
begin
  for X := Low(List1) to High(List1) do
```

```
  begin
      List1[X] := X * 3.4;
      ListBox1.Items.Add(FloatToStr(List1[X]));
  end;
  for X := Low(List2) to High(List2) do
  begin
      List2[X] := X * 0.0123;
      ListBox2.Items.Add(FloatToStr(List2[X]));
  end;
end;
```

See Also

Low (see page 3706)

VarArrayHighBound

38.3.384 System.Inc Function

Increments an ordinal value by one or N.

Pascal

```
procedure Inc(var X); overload;
procedure Inc(var X; N: Longint); overload;
```

C++

```
Inc( X);
Inc( X, Longint N);
```

File

System

Description

In Delphi code, Inc adds one or N to the variable X.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X increments by 1, or by N if N is specified; that is, Inc(X) corresponds to the statement X := X + 1, and Inc(X, N) corresponds to the statement X := X + N. On some platforms, Inc may generate optimized code, especially useful in tight loops.

If X is a pointer type, it increments X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Inc(P)` increments `P` by `SizeOf (see page 3761)(TMyType)`.

Warning: You can't use Inc on properties because it modifies the parameter.

Note: Inc(S, I) where S is a ShortInt and I is a number greater than 127 will cause an EIntOverFlow exception to be raised if range and overflow checking are on. In Delphi 1.0, this did not raise an exception.

Delphi Examples:

```
{
```

```
This example requires two buttons and two text fields.  The
text fields can be set to any integer and long integer value.
Click the buttons to execute Inc.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  IntVar: Integer;
begin
  IntVar := StrToInt(Edit1.Text);
  Inc(IntVar);
  Edit1.Text := IntToStr(IntVar);       { IntVar := IntVar + 1 }
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  LongintVar: Longint;
begin
  LongintVar := StrToInt(Edit2.Text);
  Inc(LongintVar, 5);
  Edit2.Text := IntToStr(LongintVar);   { LongintVar := LongintVar + 5 }
end;
```

See Also

Dec (see page 3652)

Pred (see page 3721)

Succ (see page 3768)

38.3.385 System.Include Function

Adds an element to a Delphi set or adds an event handler in .NET applications.

Pascal

```
procedure Include(var S: set of T; I: T);
```

C++

```
Include(set of T S, T I);
```

File

System

Description

When the first argument of Include is a set type variable, Include adds an element to the set. The expression `Include(S,I)` corresponds to `S := S + [I]`, where S is a set type variable, and I is an expression of a type compatible with the base type of S. The Include procedure generates more efficient code than using the + operator.

On the .NET platform, Include is overloaded. When the first argument of Include is a multicast event, and the second argument is an event handler, Include adds the event handler as a listener to the event.

See Also

Exclude (see page 3664)

38.3.386 System.Initialize Function

Initializes a dynamically allocated variable.

Pascal

```
procedure Initialize(var V); overload;
procedure Initialize(var V; Count: Integer); overload;
```

C++

```
Initialize( V);
Initialize( V, int Count);
```

File

System

Description

Initialize should be used only in Delphi code where a variable is dynamically allocated by other means than the New (see page 3712) standard procedure.

For global variables, local variables, objects, and dynamic variables allocated using New (see page 3712), the compiler generates code that initializes all long strings and variants contained by a variable upon creation of the variable. A call to Initialize is required to initialize a variable before it can be used if:

- the dynamic variable is created by other means than the New (see page 3712) standard function (for example using GetMem (see page 3683) or ReallocMem (see page 3729)).
- the variable contains long strings, variants, or interfaces.
- the dynamic variable is created by other means than the New (see page 3712) standard function (for example using GetMem (see page 3683) or ReallocMem (see page 3729)).
- the variable contains long strings, variants, or interfaces.
- the memory allocated for the variable is not initialized to zeros.

Initialize simply zeros out the memory occupied by long strings, variants, and interfaces, causing long strings to be empty and variants and interfaces to be Unassigned.

In cases where several variables are allocated in a contiguous memory block, the additional Count parameter can be specified to initialize all variables in one operation.

If the variable specified in a call to Initialize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

See Also

Finalize (see page 3672)

New (see page 3712)

38.3.387 System.Insert Function

Inserts a substring into a string beginning at a specified point.

Pascal

```
procedure Insert(Source: string; var S: string; Index: Integer);
```

C++

```
Insert(UnicodeString Source, UnicodeString S, int Index);
```

File

System

Description

In Delphi code, Insert merges Source into S at the position S[index].

Source is a string-type expression. S is a string-type variable of any length. Index is an integer-type expression. It is a character index and not a byte index.

If Index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to the length of the string, turning the operation into an append.

If the Source parameter is an empty string, Insert does nothing.

Insert throws an EOutOfMemory exception if it is unable to allocate enough memory to accomodate the new returned string.

Delphi Examples:

```
{
This example uses the Insert function to edit a string.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'Honest Lincoln';
  Canvas.TextOut(10, 10, S);
  Insert('Abe ', S, 8);                    { 'Honest Abe Lincoln' }
  Canvas.TextOut(10, 40, S);
end;
```

See Also

Concat (see page 3646)

Copy (see page 3648)

Delete (see page 3653)

Length (see page 3702)

Pos (see page 3720)

38.3.388 System.Int Function

Returns the integer part of a real number.

Pascal

```
function Int(const X: Extended): Extended;
```

C++

```
Extended Int(const Extended X);
```

File

System

Description

In Delphi code, Int returns the integer part of X; that is, X rounded toward zero. X is a real-type expression.

Delphi Examples:

```
{
Click the button to display the integer part of any positive or
negative real number.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: Real;
begin
  R := StrToFloat(Edit1.Text);     { 123.456 }
  R := Int(R);                     { 123.0 }
  Edit2.Text := FloatToStr(R);
end;
```

See Also

Frac (see page 3675)

Round (see page 3739)

Trunc (see page 3773)

38.3.389 System.IOResult Function

Returns the status of the last I/O operation performed.

Pascal

```
function IOResult: Integer;
```

C++

```
int IOResult();
```

File

System

Description

IOResult returns the result of the last Delphi I/O routine when I/O-checking is off (that is, when using the {$I-} compiler option). If IOResult returns 0, the last I/O operation succeeded. Otherwise, IOResult returns an error code.

If an I/O error occurs and I/O-checking is off, all subsequent I/O operations are ignored until the internal error flag is cleared. Calling IOResult clears the internal error flag.

An alternative way to handle I/O errors is to use exception handling with the {$I+} compiler option.

Delphi Examples:

```
{
This example illustrates how to use IOResult to check the success of the Reset
method. If the call to Reset is successful, a message dialog displays the size
of the file that the user selects in an Open dialog. If the call to Reset fails,
a message warns the user that there was a file access error. To test the failure
part of this example, just misspell a file name in the open dialog.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File of Byte;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    {$I-}
    Reset(F);
    {$I+}
    if IOResult = 0 then
    begin
      MessageDlg('File size in bytes: ' + IntToStr(FileSize(F)),
        mtInformation, [mbOk], 0);
      CloseFile(F);
    end
    else
      MessageDlg('File access error', mtWarning, [mbOk], 0);
  end;
end;
```

38.3.390 System.IsMemoryManagerSet Function

Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3751) procedure.

Pascal

```
function IsMemoryManagerSet: Boolean;
```

C++

```
Boolean IsMemoryManagerSet();
```

File

System

Description

Call IsMemoryManagerSet to determine whether the default system memory management has been changed by a call to SetMemoryManager (see page 3751). IsMemoryManagerSet returns false if the default system memory management is in place, true otherwise.

See Also

GetMemoryManager (see page 3686)

SetMemoryManager (see page 3751)

38.3.391 System.IsVariantManagerSet Function

Indicates whether variant support is implemented in an application.

Pascal

```
function IsVariantManagerSet: Boolean; deprecated;
```

C++

```
Boolean IsVariantManagerSet();
```

File

System

Description

Use IsVariantManagerSet in Delphi code to determine whether the compiler knows the locations of functions that define variant behavior. In any program or unit that uses the Variants (see page 4997) unit (directly or indirectly), IsVariantManagerSet should return true.

Note: Although IsVariantManagerSet is defined in System.hpp, do not use this function in C++ code. Working directly with the Variant (see page 3430) manger is not guaranteed to work in C++.

See Also

GetVariantManager (see page 3692)

SetVariantManager (see page 3759)

38.3.392 System.Length Function

Returns the number of characters in a string or elements in an array.

Pascal

```
function Length(S): Integer;
```

C++

```
int Length( S);
```

File

System

Description

In Delphi code, Length returns the number of characters actually used in the string or the number of elements in the array. In C++ code, use the method of the same name on the AnsiString or DynamicArray (see page 3268) class.

For single-byte and multibyte strings, Length returns the number of bytes used by the string. For Unicode (WideString (see page 3476)) strings, Length returns the number of bytes divided by two.

S is a string- or array-valued expression.

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
    try
      Label1.Caption :=
        Label1.Caption +
```

```
          SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
      except
        Beep;
      end;
    end;
  end;
  {
  The following OnFind event handler searches a memo component
  for the text specified in the FindText property of a find
  dialog component. If found, the first occurrence of the text
  in Memo1 is selected. The code uses the Pos function to
  compare strings, and stores the number of characters to skip
  when determining the selection position in the SkipChars
  variable. Because there is no handling of case, whole word,
  or search direction in this algorithm, it is assumed that
  the Options property of FindDialog1 is set to
  [frHideMatchCase, frHideWholeWord, frHideUpDown].
  }
  procedure TForm1.FindDialog1Find(Sender: TObject);
  var
    I, J, PosReturn, SkipChars: Integer;
  begin
    for I := 0 to Memo1.Lines.Count do
    begin
      PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
      if PosReturn <> 0 then {found!}
      begin
        SkipChars := 0;
        for J := 0 to I - 1 do
          SkipChars := SkipChars + Length(Memo1.Lines[J]);
        SkipChars := SkipChars + (I*2);
        SkipChars := SkipChars + PosReturn - 1;
        Memo1.SetFocus;
        Memo1.SelStart := SkipChars;
        Memo1.SelLength := Length(FindDialog1.FindText);
        Break;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'country.list';
  begin
    Memo1.Lines.LoadFromFile(Path);
  end;
```

See Also

Concat (see page 3646)

Copy (see page 3648)

Delete (see page 3653)

Insert (see page 3698)

Pos (see page 3720)

38.3.393 System.Ln Function

Returns the natural log of a real expression.

Pascal

```
function Ln(const X: Extended): Extended;
```

C++

```
Extended Ln(const Extended X);
```

File

System

Description

In Delphi code, Ln returns the natural logarithm (Ln(e) = 1) of the real-type expression X.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Exp (see page 3667)

LnXP1

38.3.394 System.Lo Function

Returns the low order Byte of argument X.

Pascal

```
function Lo(X): Byte;
```

C++

```
Byte Lo( X);
```

File

System

Description

In Delphi code, Lo returns the low-order Byte of the argument X as an unsigned value. X is an expression of type Integer.

Delphi Examples:

```
{
This example demonstrate use of the System Lo function to
store the low byte of a word or integer into a byte.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Lo($1234);
  ShowMessage(Format('Low byte of $1234 = $%x', [B]));
end;
```

See Also

Hi (see page 3694)

38.3.395 System.LoadResourceModule Function

Pascal

```
function LoadResourceModule(ModuleName: PChar; CheckOwner: Boolean = True): LongWord;
```

C++

```
LongWord LoadResourceModule(const char * ModuleName, Boolean CheckOwner = True);
```

File

System

Description

This is function System.LoadResourceModule.

38.3.396 System::LoadResourceString Function

Pascal

```
function LoadResourceString(rsrc: ResourceString const * const): String;
```

C++

```
__fastcall String LoadResourceString(ResourceString const * const rsrc);
```

File

sysclass.h

Description

This is function System::LoadResourceString.

38.3.397 System.LoadResString Function

Pascal

```
function LoadResString(ResStringRec: PResStringRec): string;
```

C++

```
UnicodeString LoadResString(PResStringRec ResStringRec);
```

File

System

Description

This is function System.LoadResString.

38.3.398 System.Low Function

Returns the lowest value in a range.

Pascal

```
procedure Low(X);
```

C++

```
Low( X);
```

File

System

Description

In Delphi code, call Low to obtain the lowest value or first element of an Ordinal, Array or ShortString. Result type is X, or the index type of X where X is either a type identifier or a variable reference.

Type	Low returns
Ordinal type (includes Int64)	The lowest value in the range of the type
Array type	The lowest value within the range of the index type of the array
short string type	Returns 0
Open array	Returns 0
short string parameter	Returns 0

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;

function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0; { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, TempStr: string;
begin
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);

end;

procedure TForm1.FormCreate(Sender: TObject);
var
  X: Word;
begin
  for X := Low(List1) to High(List1) do
  begin
      List1[X] := X * 3.4;
      ListBox1.Items.Add(FloatToStr(List1[X]));
  end;
  for X := Low(List2) to High(List2) do
  begin
      List2[X] := X * 0.0123;
      ListBox2.Items.Add(FloatToStr(List2[X]));
  end;
end;
```

See Also

High (see page 3694)

VarArrayLowBound

38.3.399 System.Mark Function

Pascal

```
procedure Mark; deprecated;
```

C++

```
Mark();
```

File

System

Description

This is function System.Mark.

38.3.400 System.MemoryBarrier Function

Pascal

```
procedure MemoryBarrier;
```

C++

```
MemoryBarrier();
```

File

System

Description

This is function System.MemoryBarrier.

38.3.401 System.MkDir Function

Creates a new subdirectory.

Pascal

```
procedure MkDir(const S: string); overload;
procedure MkDir(P: PChar); overload;
```

C++

```
MkDir(const UnicodeString S);
MkDir(const char * P);
```

File

System

Description

MkDir creates a new subdirectory with the path specified by string S or P. The last item in the path cannot be an existing file name.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for an I/O error.

Delphi Examples:

```
{
The following sample code sets the directory of
DirectoryListBox1 to C:\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
mytemp is added to C:\WINDOWS, but note that it is not
updated in DirectoryListBox1 until Button2 is pressed
and Update is called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current path to see the magic happen.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir(Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  DirectoryListBox1.Directory := 'c:\WINDOWS';
end;
{
This example creates a directory using the path and directory name
entered in the text edit.  The path is relative to the project
directory.  Type in the name of an existing directory to induce the
error.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  {$I-}
  { Get directory name from TEdit control }
  MkDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot create directory', mtWarning, [mbOk], 0, mbOk)
  else
    MessageDlg('New directory created', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 3638)

GetDir (see page 3679)

RmDir (see page 3738)

ForceDirectories

38.3.402 System.MonitorEnter Function

Pascal

```
function MonitorEnter(AObject: TObject; Timeout: Cardinal = INFINITE): Boolean;
```

C++

```
Boolean MonitorEnter(TObject * AObject, unsigned Timeout = INFINITE);
```

File

System

Description

This is function System.MonitorEnter.

38.3.403 System.MonitorExit Function

Pascal

```
procedure MonitorExit(AObject: TObject);
```

C++

```
MonitorExit(TObject * AObject);
```

File

System

Description

This is function System.MonitorExit.

38.3.404 System.MonitorPulse Function

Pascal

```
procedure MonitorPulse(AObject: TObject);
```

C++

```
MonitorPulse(TObject * AObject);
```

File

System

Description

This is function System.MonitorPulse.

38.3.405 System.MonitorPulseAll Function

Pascal

```
procedure MonitorPulseAll(AObject: TObject);
```

C++

```
MonitorPulseAll(TObject * AObject);
```

File

System

Description

This is function System.MonitorPulseAll.

38.3.406 System.MonitorTryEnter Function

Pascal

```
function MonitorTryEnter(AObject: TObject): Boolean;
```

C++

```
Boolean MonitorTryEnter(TObject * AObject);
```

File

System

Description

This is function System.MonitorTryEnter.

38.3.407 System.MonitorWait Function

Pascal

```
function MonitorWait(AObject: TObject; Timeout: Cardinal): Boolean;
```

C++

```
Boolean MonitorWait(TObject * AObject, unsigned Timeout);
```

File

System

Description

This is function System.MonitorWait.

38.3.408 System.Move Function

Copies bytes from a source to a destination.

Pascal

```
procedure Move(const Source; var Dest; Count: Integer);
```

C++

```
Move(const  Source,  Dest, int Count);
```

File

System

Description

Move copies Count bytes from Source to Dest. No range checking is performed. Move compensates for overlaps between the source and destination blocks.

Whenever possible, use the global SizeOf (see page 3761) function (Delphi) or the sizeof operator (C++) to determine the count.

Delphi Examples:

```
{
This example moves characters from a Char array into an integer.
Displaying the integer as a hex, you can see that 'W' (0x57) is
stored in the least significant byte of the integer.
}
var
  A: array[1..4] of Char;
  B: Integer;

procedure DisplayAB;
begin
  Form1.ListBox1.Items[0] := A[1];
  Form1.ListBox1.Items[1] := A[2];
  Form1.ListBox1.Items[2] := A[3];
  Form1.ListBox1.Items[3] := A[4];
  Form1.Edit1.Text := IntToHex(B, 1);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Move(A, B, SizeOf(B));  { SizeOf = safety! }
  DisplayAB;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I : Integer;
begin
  A[1] := 'W';
  A[2] := 'H';
  A[3] := 'A';
  A[4] := 'T';
  B := 5;
  DisplayAB;
end;
```

38.3.409 System.MoveChars Function

Pascal

```
procedure MoveChars(const Source; var Dest; Length: Integer);
```

C++

```
MoveChars(const  Source,  Dest, int Length);
```

File

System

Description

This is function System.MoveChars.

38.3.410 System.New Function

Creates a new dynamic variable and sets P to point to it.

Pascal

```
procedure New(var P: Pointer);
```

C++

```
New(void * P);
```

File

System

Description

In Delphi code, the New procedure creates a new dynamic variable and sets a pointer variable to point to it. P is a variable of any pointer type. The size of the allocated memory block corresponds to the size of the type that P points to. The newly created variable can be referenced as P^. If there isn't enough memory available to allocate the dynamic variable, an EOutOfMemory exception is raised.

When an application is finished using a dynamic variable created with New, it should dispose of the memory allocated for the

variable using the Dispose (see page 3654) standard procedure.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  with TreeView1 do
  begin
    TreeViewIndex := Selected.AbsoluteIndex;
    if Items.Count = 0 then
      Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
    else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
      Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
  end;
end;
{
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
    Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                  PMyRec(TreeView1.Selected.Data)^.LName;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
end;
```

See Also

Dispose (see page 3654)

FreeMem (see page 3676)

GetMem (see page 3683)

38.3.411 System.Odd Function

Returns true if argument is an odd number.

Pascal

```
function Odd(X: Integer): Boolean;
```

C++

```
Boolean Odd(int X);
```

File

System

Description

In Delphi code, Odd tests if the argument is an odd number. It returns true if X is an odd number, false if X is even.

Delphi Examples:

```
{
This example requires a text edit and a button with some clear
```

```
  canvas in the top left ocrner of the form.  Enter the number to be
  tested in the text edit and click to button to test if the entry
  is odd.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     Canvas.TextOut(10, 10, '                                              ');
     if Odd(StrToInt(Edit1.Text)) then
       Canvas.TextOut(10, 10, Edit1.Text + ' is odd.')
     else
       Canvas.TextOut(10, 10, Edit1.Text + ' is even.');
  end;
```

38.3.412 System.OleStrToString Function

Copies data received from a COM interface to a string.

Pascal

```
function OleStrToString(Source: PWideChar): UnicodeString;
```

C++

```
UnicodeString OleStrToString(PWideChar Source);
```

File

System

Description

OleStrToString copies data from the format used by COM into a Delphi string.

See Also

OleStrToStrVar (see page 3715)

StringToOleStr (see page 3767)

38.3.413 System.OleStrToStrVar Function

Copies a string from its COM representation to an existing Delphi string.

Pascal

```
procedure OleStrToStrVar(Source: PWideChar; var Dest: AnsiString); overload;
procedure OleStrToStrVar(Source: PWideChar; var Dest: UnicodeString); overload;
```

C++

```
OleStrToStrVar(PWideChar Source, AnsiString Dest);
OleStrToStrVar(PWideChar Source, UnicodeString Dest);
```

File

System

Description

OleStrToStrVar converts a wide string (the format used by COM interfaces) into an existing string variable.

See Also

OleStrToString (see page 3715)

StringToOleStr (see page 3767)

38.3.414 System.Ord Function

Returns the ordinal value of an ordinal-type expression.

Pascal

```
procedure Ord(X);
```

C++

```
Ord( X);
```

File

System

Description

X is a Delphi ordinal-type expression. The result is the ordinal position of X; its type is the smallest standard integer type that can hold all values of X's type.

Ord cannot operate on Int64 values.

Delphi Examples:

```
{
This example displays the ordinal value for different enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
   Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'BLUE has an ordinal value of ' + IntToStr(Ord(BLUE)) + #13#10;
  S := S + 'The ASCII code for "c" is ' + IntToStr(Ord('c')) +  ' decimal';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Chr (see page 3640)

38.3.415 System.ParamCount Function

Returns the number of parameters passed on the command line.

Pascal

```
function ParamCount: Integer;
```

C++

```
int ParamCount();
```

File

System

Description

ParamCount returns the number of parameters passed to the program on the command line. Separate parameters with spaces or tabs. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

C++ Examples:

```
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamStr (see page 3718)

CmdLine (see page 3880)

38.3.416 System.ParamStr Function

Returns a specified parameter from the command-line.

Pascal

```
function ParamStr(Index: Integer): string;
```

C++

```
UnicodeString ParamStr(int Index);
```

File

System

Description

ParamStr returns the parameter from the command line that corresponds to Index, or an empty string if Index is greater than ParamCount (see page 3716). For example, an Index value of 2 returns the second command-line parameter.

Note: On Windows, ParamStr(0) returns the path and file name of the executing program (for example, C:\TEST\MYPROG.EXE).

Note: On Linux, ParamStr(0) returns the command used to execute the program, without parameters (for example, ./myprogram). This behavior is dependent on information returned by the shell program and may not be consistent among all shells.

Note: Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

C++ Examples:

```
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
```

```
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamCount (see page 3716)

CmdLine (see page 3880)

38.3.417 System.Pi Function

Returns 3.1415926535897932385.

Pascal

```
function Pi: Extended;
```

C++

```
Extended Pi();
```

File

System

Description

Use Pi in Delphi code to represent the mathematical value pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
```

```
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

38.3.418 System.Pos Function

Returns the index value of the first character in a specified substring that occurs in a given string.

Pascal

```
function Pos(const substr: UnicodeString; const str: UnicodeString): Integer; overload;
function Pos(const substr: RawByteString; const str: RawByteString): Integer; overload;
function Pos(const substr: WideString; const str: WideString): Integer; overload;
```

C++

```
int Pos(const UnicodeString substr, const UnicodeString str);
int Pos(const RawByteString substr, const RawByteString str);
int Pos(const BSTR substr, const BSTR str);
```

File

System

Description

In Delphi, Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions.

Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero.

The PosEx function is similar to Pos, but provides additional features and can be used in C++ code.

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
```

```
    begin
      if Pos ('=', Source[I]) > 1 then
      begin
        DI := Dest.IndexOfName(Source.Names[I]);
        if DI > -1 then Dest[DI] := Source[I];
      end;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    MergeStrings(ListBox1.Items, ListBox2.Items);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Plants = 10');
    ListBox1.Items.Add('Animals = 20');
    ListBox1.Items.Add('Minerals = 15');
    ListBox2.Items.Add('Animals = 4');
    ListBox2.Items.Add('Plants = 3');
    ListBox2.Items.Add('Minerals = 78');
  end;
  {
  This example demonstrates how to edit particular elements of a string.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var S: string;
  begin
    S := Edit1.Text;
    { Convert spaces to zeros }
    while Pos(' ', S) > 0 do
      S[Pos(' ', S)] := '0';
    Edit1.Text := S;
  end;
```

See Also

Concat (see page 3646)

Copy (see page 3648)

Delete (see page 3653)

Insert (see page 3698)

Length (see page 3702)

38.3.419 System.Pred Function

Returns the predecessor of the argument.

Pascal

```
procedure Pred(X);
```

C++

```
Pred( X);
```

File

System

Description

X is an expression of a Delphi ordinal type (including Int64). The result, of the same type as X, is the predecessor of X.

Delphi Examples:

```
{
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project. Build the executable in the IDE
by hitting "Run".  Then go to the debug directory for the project and
execute the command line "AssertErrorProc_proj /Debug".
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;
```

```
procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;

  ModifyStorage(nil, 'Ooops');
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Succ (see page 3768)

38.3.420 System.Ptr Function

Converts a specified address to a pointer.

Pascal

```
function Ptr(Address: Integer): Pointer;
```

C++

```
void * Ptr(int Address);
```

File

System

Description

In Delphi code, Ptr converts the given address to a pointer. A call to this function generates no code, but simply treats the 32-bit value given by Address as a pointer. Like nil, the result of Ptr is assignment compatible with all pointer types.

See Also

Addr (see page 3621)

38.3.421 System.PUCS4Chars Function

Converts a UCS4 string to a null-terminated array of UCS4 characters.

Pascal

```
function PUCS4Chars(const S: UCS4String): PUCS4Char;
```

C++

```
PUCS4Char PUCS4Chars(const UCS4String S);
```

File

System

Description

PUCS4Chars converts the UCS-4 string specified by S to a null-terminated sequence of UCS4Char (see page 3871) values.

See Also

UCS4StringToWideString (see page 3775)

Utf8ToUnicode (see page 3783)

WideStringToUCS4String (see page 3789)

38.3.422 System.RaiseList Function

Pascal

```
function RaiseList: Pointer; deprecated;
```

C++

```
void * RaiseList();
```

File

System

Description

This is function System.RaiseList.

38.3.423 System.Random Function

Generates random numbers within a specified range.

Pascal

```
function Random(const ARange: Integer): Integer; overload;
function Random: Extended; overload;
```

C++

```
int Random(const int ARange);
Extended Random();
```

File

System

Description

In Delphi code, Random returns a random number within the range 0 <= X < Range. If Range is not specified, the result is a real-type random number within the range

`0 <= X < 1.`

To initialize the random number generator, add a single call Randomize (see page 3726) or assign a value to the RandSeed (see page 3902) variable before making any calls to Random.

Note: Because the implementation of the Random function may change between compiler versions, we do not recommend using Random for encryption or other purposes that require reproducible sequences of pseudo-random numbers.

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
```

```
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandSeed (see page 3902)

38.3.424 System.Randomize Function

Initializes the random number generator with a random value.

Pascal

```
procedure Randomize;
```

C++

```
Randomize();
```

File

System

Description

Randomize initializes the built-in random number generator with a random value (obtained from the system clock). The random number generator should be initialized by making a call to Randomize, or by assigning a value to RandSeed (see page 3902).

Do not combine the call to Randomize in a loop with calls to the Random (see page 3725) function. Typically, Randomize is called only once, before all calls to Random (see page 3725).

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
```

```
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandG

RandSeed (see page 3902)

Random (see page 3725)

38.3.425 System.Read Function

Read reads data from a file.

Pascal

```
procedure Read(F; V1); overload;
procedure Read(F; V1; Vn); overload;
procedure Read(F: Text; V1); overload;
procedure Read(F: Text; V1; Vn); overload;
```

C++

```
Read( F,  V1);
Read( F,  V1,  Vn);
Read(Text F,  V1);
Read(Text F,  V1,  Vn);
```

File

System

Description

The Read procedure can be used in Delphi code in the following ways.

For typed files, it reads a file component into a variable.

For text files, it reads one or more values into one or more variables.

The syntax shown here for Read demonstrates that the procedure can take a variable number of arguments.

Read reads all characters up to, but not including, the next end-of-line marker or until Eof (see page 3659)(F) becomes true; it does not skip to the next line after reading. If the resulting string is longer than the maximum length of the string variable, it is truncated.

After the first Read, each subsequent Read sees the end-of-line marker and returns a zero-length string.

Use multiple Readln calls to read successive string values.

When the extended syntax is enabled, Read can read null-terminated strings into zero-based character arrays.

Read reads one character from the file and assigns it to the variable. If CRLF mode is enabled and Eof (see page 3659)(F) was true before Read was executed, the value Chr (see page 3640)(26) (a Ctrl-Z character) is assigned to the variable. (To enable CRLF mode, use SetLineBreakStyle (see page 3750).)

Read skips any blanks, tabs, or end-of-line markers preceding the numeric string.

If the numeric string does not conform to the expected format, an I/O error occurs; otherwise, the value is assigned to the variable.

The next Read starts with the blank, tab, or end-of-line marker that terminated the numeric string.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

Eof (see page 3659)

Readln

Write (see page 3789)

Writeln

SetLineBreakStyle (see page 3750)

38.3.426 System.ReadLn Function

Reads a line of text from a file.

Pascal

```
procedure ReadLn(V1); overload;
procedure ReadLn(V1; Vn); overload;
procedure ReadLn(var F: Text; V1); overload;
procedure ReadLn(var F: Text; V1; Vn); overload;
```

C++

```
ReadLn( V1);
ReadLn( V1,  Vn);
ReadLn(Text F,  V1);
ReadLn(Text F,  V1,  Vn);
```

File

System

Description

In Delphi code, the ReadLn procedure reads a line of text and then skips to the next line of the file.

The syntax shown here for ReadLn demonstrates that the procedure can take a variable number of arguments.

ReadLn(F) with no parameters causes the current file position to advance to the beginning of the next line if there is one; otherwise, it goes to the end of the file.

If F is omitted, the global variable Input (see page 3895) is used, which accesses the processes standard input file. Use of Input (see page 3895) in GUI applications raises special issues.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}

{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.  Click on the text edit to open the
open dialog box.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
```

See Also

Read (see page 3727)

Writeln

SetLineBreakStyle (see page 3750)

38.3.427 System.ReallocMem Function

ReallocMem reallocates a memory block.

Pascal

```
procedure ReallocMem(var P: Pointer; Size: Integer);
```

C++

```
ReallocMem(void * P, int Size);
```

File

System

Description

GetMem (see page 3683) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMem, passing the existing memory block pointer in **P**, and the revised block size in **Size**. If ReallocMem can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory (see page 3730) is the C++ compatible version of ReallocMem.

See Also

AllocMem (see page 3622)

FreeMem (see page 3676)

FreeMemory (see page 3678)

GetMem (see page 3683)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMemory (see page 3730)

38.3.428 System.ReallocMemory Function

ReallocMemory resizes a memory block.

Pascal

```
function ReallocMemory(P: Pointer; Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * ReallocMemory(void * P, int Size);
```

File

System

Description

GetMem (see page 3683) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMemory, passing the existing memory block pointer in **P**, and the revised block size in **Size**. A pointer to the resized memory block is returned. If ReallocMemory can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory is the C++ compatible version of ReallocMem (see page 3729).

See Also

AllocMem (see page 3622)

FreeMem (see page 3676)

FreeMemory (see page 3678)

GetMem (see page 3683)

GetMemory (see page 3684)

EOutOfMemory (see page 4008)

ReallocMem (see page 3729)

38.3.429 System.RegisterExpectedMemoryLeak Function

Registers an expected memory leak.

Pascal

```
function RegisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean RegisterExpectedMemoryLeak(void * P);
```

File

System

Description

RegisterExpectedMemoryLeak registers a memory location that an application has allocated and does not expect to free.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

UnregisterExpectedMemoryLeak (see page 3777)

ReportMemoryLeakOnShutdown

38.3.430 System.RegisterModule Function

Pascal

```
procedure RegisterModule(LibModule: PLibModule);
```

C++

```
RegisterModule(PLibModule LibModule);
```

File

System

Description

This is function System.RegisterModule.

38.3.431 System.Release Function

Pascal

```
procedure Release; deprecated;
```

C++

```
Release();
```

File

System

Description

This is function System.Release.

38.3.432 System.ReleaseExceptionObject Function

Releases exception object acquired by AcquireExceptionObject (see page 3619).

Pascal

```
procedure ReleaseExceptionObject;
```

C++

```
ReleaseExceptionObject();
```

File

System

Description

ReleaseExceptionObject releases an exception object previously acquired by AcquireExceptionObject (see page 3619).

See Also

AcquireExceptionObject (see page 3619)

ExceptObject (see page 3664)

38.3.433 System.RemoveModuleUnloadProc Function

RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.

Pascal

```
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
RemoveModuleUnloadProc(TModuleUnloadProc Proc);
RemoveModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc (see page 3620) adds your routine to the head of the list of internal routines that Delphi invokes to perform the unload. RemoveModuleUnloadProc reverses the effect of AddModuleUnloadProc (see page 3620).

See Also

AddModuleUnloadProc (see page 3620)

TModuleUnloadProc (see page 3863)

38.3.434 System.Rename Function

Changes the name of an external file.

Pascal

```
procedure Rename(var F; Newname: string); overload;
procedure Rename(var F; Newname: PChar); overload;
```

C++

```
Rename( F, UnicodeString Newname);
Rename( F, const char * Newname);
```

File

System

Description

The external file associated with F is renamed Newname. Further operations on F operate on the external file with the new name.

F is a variable of any file type. Newname is a string-type expression or an expression of type PChar if the extended syntax is enabled.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example requires a button, an open dialog and a save
dialog.  Click the button to rename a file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  f : file;
begin
  OpenDialog1.Title := 'Choose a file... ';
  if OpenDialog1.Execute then
  begin
    SaveDialog1.Title := 'Rename to...';
    if SaveDialog1.Execute then
    begin
```

```
        AssignFile(f, OpenDialog1.FileName);
        Canvas.TextOut(5, 10, 'Renaming ' + OpenDialog1.FileName +
                       ' to ' + SaveDialog1.FileName);
        Rename(f, SaveDialog1.FileName);
      end;
    end;
  end;
```

See Also

Rename File

38.3.435 System.Reset Function

Opens an existing file.

Pascal

```
procedure Reset(var F); overload;
procedure Reset(var F; RecSize: Word); overload;
procedure Reset(var F: File); overload;
procedure Reset(var F: File; RecSize: Word); overload;
```

C++

```
Reset( F);
Reset( F, Word RecSize);
Reset(File F);
Reset(File F, Word RecSize);
```

File

System

Description

In Delphi code, Reset opens the existing external file with the name assigned to F using the mode specified by the global FileMode (see page 3892) variable. An error results if no existing external file of the given name exists or if the file can't be opened with the current file mode. If F is already open, it is first closed and then reopened. The current file position is set to the beginning of the file.

Warning: The default value of FileMode (see page 3892) is 2 (Read (see page 3727)/Write (see page 3789) access). If this is not changed to a read-only file mode before calling Reset, attempts to open read-only files will fail.

F is a variable of any file type associated with an external file using AssignFile (see page 3629). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If F is assigned an empty name, such as AssignFile (see page 3629)(F, ''), then after the call to Reset, F refers to the standard input file.

If F is a text file, F becomes read-only.

After a call to Reset, Eof (see page 3659)(F) is true if the file is empty; otherwise, Eof (see page 3659)(F) is false.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
```

```
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example should be executed from a command line
formatted like this: "SystemReset_proj foo.txt".  The file
path is relative to the project directory.
}
function FileIsThere(FileName: string): Boolean;
{ Boolean function that returns True if the file exists; otherwise,
  it returns False. Closes the file if it exists. }
 var
  F: file;
begin
  {$I-}
  AssignFile(F, FileName);
  FileMode := 0;  {Set file access to read only }
  Reset(F);
  CloseFile(F);
  {$I+}
  FileIsThere := (IOResult = 0) and (FileName <> '');
end;  { FileIsThere }

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    if FileIsThere(ParamStr(1)) then {Get file name from command line}
      Canvas.TextOut(10, 10, 'File exists')
    else
      Canvas.TextOut(10, 10, 'File not found');
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Append (see page 3624)

AssignFile (see page 3629)

CloseFile (see page 3643)

Rewrite (see page 3736)

Truncate (see page 3774)

IDH

FileMode (see page 3892)

38.3.436 System.Rewrite Function

Creates a new file and opens it.

Pascal

```
procedure Rewrite(var F: File); overload;
procedure Rewrite(var F: File; Recsize: Word); overload;
```

C++

```
Rewrite(File F);
Rewrite(File F, Word Recsize);
```

File

System

Description

In Delphi code, Rewrite creates a new external file with the name assigned to F.

F is a variable of any file type associated with an external file using AssignFile (see page 3629). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If an external file with the same name already exists, it is deleted and a new empty file is created in its place.

If F is already open, it is first closed and then re-created. The current file position is set to the beginning of the empty file.

If F was assigned an empty name, such as AssignFile (see page 3629)(F,''), then after the call to Rewrite, F refers to the standard output file.

If F is a text file, F becomes write-only.

After calling Rewrite, Eof (see page 3659)(F) is always true.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
{
Rewrite example
}

procedure TForm1.Button1Click(Sender: TObject);
var F: TextFile;
```

```
begin
  AssignFile(F, 'NEWFILE.$$$');
  Rewrite(F);  // default record size is 128 bytes
  Writeln(F, 'Just created file with this text in it...');
  CloseFile(F);
  MessageDlg('NEWFILE.$$$ has been created in the ' + GetCurrentDir + ' directory.',
    mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Append (see page 3624)

AssignFile (see page 3629)

Reset (see page 3734)

Truncate (see page 3774)

38.3.437 System.RmDir Function

Deletes an empty subdirectory.

Pascal

```
procedure RmDir(const S: string); overload;
procedure RmDir(P: PChar); overload;
```

C++

```
RmDir(const UnicodeString S);
RmDir(const char * P);
```

File

System

Description

RmDir removes the subdirectory with the path specified by S or P. If the path does not exist, is non-empty, or is the currently logged directory, an I/O error occurs.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example requires a text edit and a button.  Enter the
folder to remove in the text edit and click the button.  The
path is relative to the project folder.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   {$I-}
   { Get directory name from TEdit control }
   RmDir(Edit1.Text);
   if IOResult <> 0 then
     MessageDlg('Cannot remove directory', mtWarning, [mbOk], 0, mbOk)
   else
     MessageDlg('Directory removed', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 3638)

GetDir (see page 3679)

MkDir (see page 3708)

RemoveDir

38.3.438 System.Round Function

Returns the value of X rounded to the nearest whole number.

Pascal

```
function Round(X: Extended): Int64;
```

C++

```
Int64 Round(Extended X);
```

File

System

Description

In Delphi, the Round function rounds a real-type value to an integer-type value.

X is a real-type expression. Round returns an Int64 value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is always the even number. This method of rounding is often called "Banker's Rounding".

If the rounded value of X is not within the Int64 range, a runtime error is generated, which can be handled using the EInvalidOp exception.

Note: The behavior of Round can be affected by the Set8087CW (see page 3744) procedure or SetRoundMode function.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
{
This example illustrates the rules for rounding positive and
negative real numbers to integers.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' rounds to ' + IntToStr(Round(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(1.5)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(-1.5));
   MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Int (see page 3699)

Trunc (see page 3773)

Set8087CW (see page 3744)

SetRoundMode

RoundTo

SimpleRoundTo

38.3.439 System.RunError Function

Stops execution and generates a runtime error.

Pascal

```
procedure RunError; overload;
procedure RunError(Errorcode: Byte); overload;
```

C++

```
RunError();
RunError(Byte Errorcode);
```

File

System

Description

In Delphi, the RunError procedure stops program execution by generating a runtime error with the given number at the current

statement. Errorcode is the runtime error number (0 if omitted).

Delphi Examples:

```
{
This example demonstrates how to set a run error, the behavior of
execution and the message that follows.
}
var P: Pointer;

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF Debug}
  if P = nil then
    RunError(204);
{$ENDIF}
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  P:= nil;
end;
```

See Also

Halt (see page 3693)

38.3.440 System.Seek Function

Moves the current position of a file to a specified component.

Pascal

```
procedure Seek(var F; N: Longint);
```

C++

```
Seek( F, Longint N);
```

File

System

Description

Use Seek in Delphi code to move to a specified position in open typed or untyped files. The current file position of F moves to component number N, where the number of the first component of a file is 0.

In the above syntax, F is a typed or untyped file variable. The file specified by F must be open. N is an expression of type Longint

To expand a file, seek one component beyond the last component; that is, the statement Seek(F, FileSize (see page 3669)(F)) moves the current file position to the end of the file.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
```

```
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
```

See Also

FilePos (see page 3668)

38.3.441 System.SeekEof Function

Returns the end-of-file status of a file, ignoring whitespace.

Pascal

```
function SeekEof: Boolean; overload;
function SeekEof(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEof();
Boolean SeekEof(Text F);
```

File

System

Description

In Delphi code, call SeekEof to determine whether there is only whitespace between the file pointer and the end of a file. SeekEof moves the file pointer past any whitespace, leaving it positioned at the end of a file or on the next non-whitespace character, whichever comes first. If it leaves the file pointer at the end of the file, SeekEof returns true. Otherwise, it returns false.

SeekEof can only be used on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
This example creates a file TEST.TXT, puts some text in it
and then reads the file to EOF.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f : System.TextFile;
   j, Y : Integer;
 begin
   AssignFile(f,'TEST.TXT');
   Rewrite(f);
   { Create a file with 8 numbers and some
     whitespace at the ends of the lines }
   Writeln(f,'1 2 3 4 ');
   Writeln(f,'5 6 7 8 ');
   Reset(f);
   { Read the numbers back. SeekEof returns TRUE if there is no
     more text (other than whitespace) in the file. }
   Y := 5;
   while not SeekEof(f) do
   begin
     Read(f,j);
     Canvas.TextOut(5, Y, IntToStr(j));
     Y := Y + Canvas.TextHeight(IntToStr(j)) + 5;
   end;
 end;
```

See Also

Eof (see page 3659)

SeekEoln (see page 3743)

SetLineBreakStyle (see page 3750)

38.3.442 System.SeekEoln Function

Returns the end-of-line status of a file, ignoring whitespace.

Pascal

```
function SeekEoln: Boolean; overload;
function SeekEoln(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEoln();
Boolean SeekEoln(Text F);
```

File

System

Description

In Delphi code, call SeekEoln to determine whether there is only whitespace between the file pointer and the end of the current line. SeekEoln moves the file pointer past any whitespace, leaving it positioned at the end of the current line, the end of the file, or on the next non-whitespace character (whichever comes first). If SeekEoln leaves the file pointer at the end of the current line or at the end of the file, it returns true. Otherwise, it returns false.

SeekEoln can be used only on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3700) to check for I/O errors.

Delphi Examples:

```
{
Create data with white space at beginning and end of line to
demonstrate that SeekEoln is immune to this.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: System.TextFile;
   j, Y: Integer;
   s: string;
begin
   AssignFile(f,'TEST.TXT');
   Rewrite(f);

   Writeln(f, '    1 2 3 4      ');
   Writeln(f, 'Some text');

   Reset(f);
   Y := 5;

   while not SeekEoln(f) do // Will skip white space
   begin
     Read(f, j);
     Canvas.TextOut(5, Y, IntToStr(j));
     Y := Y + Canvas.TextHeight(IntToStr(j)) + 5;
   end;
   Readln(f); // Read line-end marker

   Readln(f, s); // Read string with line-end marker
   Canvas.TextOut(5, Y, s);
end;
```

See Also

Eoln (see page 3660)

SeekEof (see page 3742)

SetLineBreakStyle (see page 3750)

38.3.443 System.Set8087CW Function

Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3901) declared in the system unit.

Pascal

```
procedure Set8087CW(NewCW: Word);
```

C++

```
Set8087CW(Word NewCW);
```

File

System

Description

The floating-point unit control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

This routine allows the programmer to have direct access to the CW. Be aware that using this routine to change the value of the 8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

It is recommended that you disable all floating-point exceptions when using OpenGL to render 3D graphics. To do this, call Set8087CW(0x133f) in your main form's OnCreate event before calling any OpenGL functions.

C++ Examples:

```
/*
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
*/
Word Saved8087CW;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit3->Text = FloatToStr(StrToFloat(Edit1->Text) / StrToFloat(Edit2->Text));
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "FPU Exceptions")
    System::Set8087CW(Saved8087CW);
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "No FPU Exceptions")
    System::Set8087CW(0x133f); // Disable all fpu exceptions
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("No FPU Exceptions");
  RadioGroup1->Items->Add("FPU Exceptions");
  RadioGroup1->ItemIndex = 2;
  Saved8087CW = Default8087CW;  // Save this because Set8087CW changes it!
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  System::Set8087CW(Saved8087CW); // Default value (with exceptions) is 0x1372
}
```

Delphi Examples:

```
{
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
}
var
  Form1: TForm1;
  Saved8087CW: Word;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit3.Text := FloatToStr(StrToFloat(Edit1.Text) / StrToFloat(Edit2.Text));
end;
```

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'FPU Exceptions' then
    System.Set8087CW(Saved8087CW);
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'No FPU Exceptions' then
    System.Set8087CW($133f); { Disable all fpu exceptions }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('No FPU Exceptions');
  RadioGroup1.Items.Add('FPU Exceptions');
  RadioGroup1.ItemIndex := 2;
  Saved8087CW := Default8087CW;  // Save this because Set8087CW changes it!
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  System.Set8087CW(Saved8087CW); // Default value (with exceptions) is $1372
end;
```

See Also

NoErrMsg (see page 3901)

SetExceptionMask

SetPrecisionMode

SetRoundMode

38.3.444 System.SetCodePage Function

Pascal

```
procedure SetCodePage(var S: RawByteString; CodePage: Word; Convert: Boolean = True);
```

C++

```
SetCodePage(RawByteString S, Word CodePage, Boolean Convert = True);
```

File

System

Description

This is function System.SetCodePage.

38.3.445 System.SetInOutRes Function

Pascal

```
procedure SetInOutRes(NewValue: Integer);
```

C++

```
SetInOutRes(int NewValue);
```

File

System

Description

This is function System.SetInOutRes.

38.3.446 System.SetLastError Function

SetLastError sets the last API error code.

Pascal

```
procedure SetLastError(ErrorCode: Integer); stdcall;
```

C++

```
__stdcall SetLastError(int ErrorCode);
```

File

System

Description

GetLastError (see page 3681) returns the last API error code encountered. It then resets it to 0. SetLastError forces the API error code to the specified **ErrorCode** value. A call to GetLastError (see page 3681) will then return this value before again resetting the error code to 0.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);

    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
    on Ex : EOSError do
       MessageDlg('Caught an OS error with code: ' +
          IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
  end;

  { Let the Delphi's Exception dialog pop-up }
  RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

  { Finally set the last error to none }
  System.SetLastError(ERROR_SUCCESS);

  if GetLastError() <> ERROR_SUCCESS then
     MessageDlg('Whoops, something went wrong in the ' +
                'mean time!', mtError, [mbOK], 0);

  { No exception should be thrown here }
  RaiseLastOSError();
end;
```

See Also

GetLastError (see page 3681)

38.3.447 System.SetLength Function

Sets the length of a string or dynamic-array variable.

Pascal

```
procedure SetLength(var S; NewLength: Integer);
```

C++

```
SetLength( S, int NewLength);
```

File

System

Description

S is a Delphi string or dynamic-array variable.

NewLength is the new number of characters or elements in S.

For a short-string variable, SetLength simply sets the length-indicator character (the character at S[0]) to the given value. In this case, NewLength must be a value between 0 and 255.

For a long-string or dynamic-array variable, SetLength reallocates the string or array referenced by S to the given length. Existing characters in the string or elements in the array are preserved, but the content of newly allocated space is undefined.

The one exception is when increasing the length of a dynamic array in which the elements are types that must be initialized (strings, Variants (see page 4997), Variant (see page 3430) arrays, or records that contain such types). When S is a dynamic array of types that must be initialized, newly allocated space is set to 0 or nil.

For dynamic arrays, SetLength may take more than one length parameter (up to the number of array dimensions). Each parameter specifies the number of elements along a particular dimension.

Following a call to SetLength, S is guaranteed to reference a unique string or array—that is, a string or array with a reference count of one. If there is not enough memory available to reallocate the variable, SetLength raises an EOutOfMemory exception.

Delphi Examples:

```
{
This example determines the encoding of a source file and then
write the file in any encoding chosen.  Use this project to
create files of any encoding.  This example requires two text edits
and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  LFileStream: TFileStream;
  EncodingArray: array[0..5] of TEncoding;
begin
  LEncoding:= nil;
  EncodingArray[0]:= TEncoding.UTF8;
  EncodingArray[1]:= TEncoding.UTF7;
  EncodingArray[2]:= TEncoding.Unicode;
  EncodingArray[3]:= TEncoding.Default;
  EncodingArray[4]:= TEncoding.BigEndianUnicode;
  EncodingArray[5]:= TEncoding.ASCII;
  DestEncoding := EncodingArray[ComboBox1.ItemIndex];
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
//    LFileStream.Read(LBuffer[0], Length(LBuffer));
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));

    // Identify encoding and convert buffer to UTF8
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
//    for I := 0 to Length(LBuffer) - 1 do
//      Memo1.Lines.Add((LBuffer[I]));
    LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
      LOffset, Length(LBuffer) - LOffset);
  finally
    LFileStream.Free;
  end;

  LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
  try
    // Write an encoding byte order mark and buffer to output file
    LByteOrderMark := DestEncoding.GetPreamble;
    LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
    LFileStream.Write(LBuffer[0], Length(LBuffer));
  finally
    LFileStream.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  StringList: TStrings;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('TEncoding.UTF8');
      Add('TEncoding.UTF7');
      Add('TEncoding.Unicode');
      Add('TEncoding.Default');
      Add('TEncoding.BigEndianUnicode');
      Add('TEncoding.ASCII');
    end;
    with ComboBox1 do begin
      Items.Assign(StringList);
      ItemIndex := 0;
    end;
  finally
    StringList.free;
  end;
end;
```

See Also

EOutOfMemory

string

IDH

38.3.448 System.SetLineBreakStyle Function

Determines the end-of-line and end-of-file conventions for text file I/O.

Pascal

```
procedure SetLineBreakStyle(var T: Text; Style: TTextLineBreakStyle);
```

C++

```
SetLineBreakStyle(Text T, TTextLineBreakStyle Style);
```

File

System

Description

In Delphi, call SetLineBreakStyle to determine how text file routines find or mark the end of a line and the end of a file. Affected routines include Read (see page 3727), Write (see page 3789), Readln, Writeln, Eof (see page 3659), Eoln (see page 3660), SeekEof (see page 3742), and SeekEoln (see page 3743).

T is the text file whose line break style is altered. It must be closed, or SetLineBreakStyle generates a runtime error.

Style indicates the style to use. When Style is tlbsLF, end-of-line is marked using only a linefeed character. When Style is tlbsCRLF, end-of-line is marked using a carriage return and linefeed.

See Also

Readln

Writeln

Read (see page 3727)

Write (see page 3789)

Eof (see page 3659)

Eoln (see page 3660)

SeekEof (see page 3742)

SeekEoln (see page 3743)

Text

DefaultTextLineBreakStyle (see page 3883)

38.3.449 System.SetMemoryManager Function

Sets entry points of the memory manager.

Pascal

```
procedure SetMemoryManager(const MemMgr: TMemoryManager); overload; deprecated;
procedure SetMemoryManager(const MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
SetMemoryManager(const TMemoryManager MemMgr);
SetMemoryManager(const TMemoryManagerEx MemMgrEx);
```

File

System

Description

SetMemoryManager sets the entry points of the memory manager to the three functions encapsulated by the MemMgr parameter. Use SetMemoryManager to change the memory allocation routines that are used for:

Allocation and deallocation of dynamic strings and dynamic arrays.

The GetMem (see page 3683), ReallocMem (see page 3729), and FreeMem (see page 3676) standard procedures (Delphi only)

The New (see page 3712) and Dispose (see page 3654) standard procedures (Delphi only).

C++ Examples:

```
/*
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
*/

bool isMyMemMgr;
TMemoryManagerEx oldMemMgr;
volatile long getMemCalls;
volatile long freeMemCalls;
volatile long reallocMemCalls;
volatile long allocMemCalls;

void* __fastcall myGetMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&getMemCalls);

    //route the call
    return oldMemMgr.GetMem(size);
```

```
}

int __fastcall myFreeMem(void* p)
{
    //safely increment the counter
    InterlockedIncrement(&freeMemCalls);

    //route the call
    return oldMemMgr.FreeMem(p);
}

void* __fastcall myReallocMem(void* p, int size)
{
    //safely increment the counter
    InterlockedIncrement(&reallocMemCalls);

    //route the call
    return oldMemMgr.ReallocMem(p, size);
}

void* __fastcall myAllocMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&allocMemCalls);

    //route the call
    return oldMemMgr.AllocMem(size);
}

void __fastcall TForm3::btUseSysMemMgrClick(TObject *Sender)
{
    //switch button states
    btUseSysMemMgr->Enabled = false;
    btUseMyMemMgr->Enabled = true;

    //set the old memory manager back
    SetMemoryManager(oldMemMgr);

    isMyMemMgr = false;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;
}

void __fastcall TForm3::FormDestroy(TObject *Sender)
{
    //set the old memory manager back
    if (isMyMemMgr)
    {
        SetMemoryManager(oldMemMgr);
    }
}

void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
    //Note that IntToStr calls will also Allocate/Free
    //memory so at each timer tick the counts will be +4;

    Form3->edGetMem->Text = IntToStr((int)getMemCalls);
    Form3->edFreeMem->Text = IntToStr((int)freeMemCalls);
    Form3->edReallocMem->Text = IntToStr((int)reallocMemCalls);
    Form3->edAllocMem->Text = IntToStr((int)allocMemCalls);
}
```

```
void __fastcall TForm3::btUseMyMemMgrClick(TObject *Sender)
{
    TMemoryManagerEx myMemMgr;

    //switch button states
    btUseSysMemMgr->Enabled = true;
    btUseMyMemMgr->Enabled = false;

    //get the old memory manager
    GetMemoryManager(oldMemMgr);

    //create out instance
    myMemMgr.GetMem = myGetMem;
    myMemMgr.FreeMem = myFreeMem;
    myMemMgr.AllocMem = myAllocMem;
    myMemMgr.ReallocMem = myReallocMem;

    //use defaults for this - not important
    myMemMgr.RegisterExpectedMemoryLeak = oldMemMgr.RegisterExpectedMemoryLeak;
    myMemMgr.UnregisterExpectedMemoryLeak = oldMemMgr.UnregisterExpectedMemoryLeak;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;

    //install the new memory manager
    SetMemoryManager(myMemMgr);

    isMyMemMgr = true;
}

void __fastcall TForm3::btDoSomethingClick(TObject *Sender)
{
    void *ptr;
    //perform some new and delete
    for (int i=0; i < 1000; ++i)
    {
        ptr = GetMemory(4);
        FreeMemory(ptr);
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  IsMyMemMgr      : Boolean;
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
```

```
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;

function MyReallocMem(P: Pointer; Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;

function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Swith button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
```

```
    ReallocMemCalls := 0;
    AllocMemCalls := 0;

    { Install the new memory manager }
    SetMemoryManager(MyMemMgr);

    IsMyMemMgr := true;
  end;

  procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
  begin
    { Swith button states }
    btUseSystemMemMgr.Enabled := false;
    btUseMyMemMgr.Enabled := true;

    { Set the old memory manager back! }
    SetMemoryManager(OldMemMgr);

    IsMyMemMgr := false;

    GetMemCalls := 0;
    FreeMemCalls := 0;
    ReallocMemCalls := 0;
    AllocMemCalls := 0;
  end;

  procedure TForm2.FormDestroy(Sender: TObject);
  begin
    { Set the old memory manager back }
    if IsMyMemMgr then
       SetMemoryManager(OldMemMgr);
  end;

  procedure TForm2.Timer1Timer(Sender: TObject);
  begin
    {
    Note that IntToStr calls will also Allocate/Free
    memory so at each timer tick the counts will be +4;
    }
    Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
    Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
    Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
    Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
  end;
```

See Also

GetMemoryManager (see page 3686)

GetMem (see page 3683)

ReallocMem (see page 3729)

FreeMem (see page 3676)

New (see page 3712)

Dispose (see page 3654)

IsMemorymanagerSet

38.3.450 System.SetMinimumBlockAlignment Function

Sets the minimum memory block alignment.

Pascal

```
procedure SetMinimumBlockAlignment(AMinimumBlockAlignment: TMinimumBlockAlignment);
```

C++

```
SetMinimumBlockAlignment(TMinimumBlockAlignment AMinimumBlockAlignment);
```

File

System

Description

SetMinimumBlockAlignment sets the Memory Manager's minimum block alignment. Existing allocations are not affected if this setting is changed. The memory manager will enforce the new setting for all allocations made after the change.

See Also

Memory Management

Configuring the Memory Manager

38.3.451 System.SetMultiByteConversionCodePage Function

SetMultiByteConversionCodePage overrides the default multi byte code conversion page.

Pascal

```
procedure SetMultiByteConversionCodePage(CodePage: Integer);
```

C++

```
SetMultiByteConversionCodePage(int CodePage);
```

File

System

Description

RAD Studion textual elements such as captions are stored as WideStrings (see page 6767). When running under 9x, they are automatically converted to AnsiStrings. This conversion uses, by default, the System (see page 3165) Code Page. SetMultiByteConversionCodePage allows you to specify the **CodePage** that is used in such conversions.

See Also

GetLastError (see page 3681)

38.3.452 System.SetRaiseList Function

Pascal

```
function SetRaiseList(NewPtr: Pointer): Pointer; deprecated;
```

C++

```
void * SetRaiseList(void * NewPtr);
```

File

System

Description

This is function System.SetRaiseList.

38.3.453 System.SetString Function

Sets the contents and length of the given string.

Pascal

```
procedure SetString(var s: string; buffer: PChar; len: Integer);
```

C++

```
SetString(UnicodeString s, const char * buffer, int len);
```

File

System

Description

In Delphi code, SetString sets the contents and length of the given string variable to the block of characters given by the Buffer and Len parameters.

For a short string variable, SetString sets the length indicator character (the character at S[0]) to the value given by Len and then, if the Buffer parameter is not nil, copies Len characters from Buffer into the string starting at S[1]. For a short string variable, the Len parameter must be a value between 0 and 255.

For a long string variable, SetString sets S to reference a newly allocated string of the given length. If the Buffer parameter is not nil, SetString then copies Len characters from Buffer into the string; otherwise, the contents of the new string is left uninitialized. If there is not enough memory available to create the string, an EOutOfMemory exception is raised. Following a call to SetString, S is guaranteed to reference a unique string, that is a string with a reference count of one.

See Also

EOutOfMemory

string

38.3.454 System.SetTextBuf Function

Assigns an I/O buffer to a text file.

Pascal

```
procedure SetTextBuf(var F: Text; var Buf); overload;
procedure SetTextBuf(var F: Text; var Buf; Size: Integer); overload;
```

C++

```
SetTextBuf(Text F,  Buf);
SetTextBuf(Text F,  Buf, int Size);
```

File

System

Description

In Delphi code, SetTextBuf changes the text file F to use the buffer specified by Buf instead of F's internal buffer. F is a text file variable, Buf is any variable, and Size is an optional expression.

Each Text file variable has an internal 128-byte buffer that buffers Read (see page 3727) and Write (see page 3789) operations. This buffer is adequate for most operations. However, heavily I/O-bound programs benefit from a larger buffer to reduce disk head movement and file system overhead.

Size specifies the size of the buffer in bytes. If Size is omitted, SizeOf (see page 3761)(Buf) is assumed. The new buffer remains in effect until F is next passed to AssignFile (see page 3629).

SetTextBuf can be called immediately after Reset (see page 3734), Rewrite (see page 3736), and Append (see page 3624), but never apply it to an open file.

When SetTextBuf is called on an open file once I/O operations have taken place, data could be lost because of the change of buffer.

The Delphi runtime library does not ensure that the buffer exists for the entire duration of I/O operations on the file. A common error is to install a local variable as a buffer, then use the file outside the procedure that declared the buffer.

Delphi Examples:

```
{
This example reads a file into a buffer and then dumps
the buffer into a new file WOOF.DOG
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F, FTwo: System.TextFile;
  Ch: Char;
  Buf: array[1..4096] of Char;  { 4K buffer }
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    { Larger buffer for faster reads }
    System.SetTextBuf(F, Buf);
    Reset(F);
    { Dump text file into another file }
    AssignFile(FTwo, 'WOOF.DOG');
    Rewrite(FTwo);
    while not Eof(f) do
    begin
```

```
      Read(F, Ch);
      Write(FTwo, Ch);
    end;
    System.CloseFile(F);
    System.CloseFile(FTwo);
  end;
```

See Also

Append (see page 3624)

AssignFile (see page 3629)

Read (see page 3727)

Reset (see page 3734)

Rewrite (see page 3736)

SizeOf (see page 3761)

Write (see page 3789)

38.3.455 System.SetVariantManager Function

Sets the entry points of routines that define variant behavior.

Pascal

```
procedure SetVariantManager(const VarMgr: TVariantManager); deprecated;
```

C++

```
SetVariantManager(const TVariantManager VarMgr);
```

File

System

Description

SetVariantManager sets the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although System.hpp defines the SetVariantManager procedure, do not use this routine in C++ code. Customized Variant (see page 3430) support is not fully supported in C++.

See Also

GetVariantManager (see page 3692)

IsVariantManagerSet (see page 3701)

38.3.456 System.ShareMemoryManager Function

Shares the module's memory manager.

Pascal

```
function ShareMemoryManager: Boolean;
```

C++

```
Boolean ShareMemoryManager();
```

File

System

Description

ShareMemoryManager makes the module's memory manager available for sharing to other modules in the same process. Only one memory manager may be shared per process. It returns **False** if another module in the process is already sharing its memory manager.

Note: This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

38.3.457 System.Sin Function

Returns the sine of the angle in radians.

Pascal

```
function Sin(const X: Extended): Extended;
```

C++

```
Extended Sin(const Extended X);
```

File

System

Description

In Delphi code, the Sin function returns the sine of the argument.

X is a real-type expression. Sin returns the sine of the angle X in radians.

Delphi Examples:

```
{
This example uses the Sine trig function to compute
the sine of Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: Extended;
  S: string;
begin
  R := Sin(Pi);
  Str(R:5:3, S);
  Canvas.TextOut(10, 10, 'The Sin of Pi is ' + S);
end;
```

See Also

ArcTan (see page 3625)

Cos (see page 3650)

38.3.458 System.SizeOf Function

Returns the number of bytes occupied by a variable or type.

Pascal

```
function SizeOf(X): Integer;
```

C++

```
int SizeOf( X);
```

File

System

Description

Pass a Delphi variable reference to SizeOf to determine the number of bytes used to represent the variable. Pass a type identifier to SizeOf to determine the number of bytes used to represent instances of that type. SizeOf is useful for determining the amount of memory to specify for the FillChar (see page 3670), Move (see page 3711), or GetMem (see page 3683) procedures.

SizeOf returns 0 when the argument is an untyped variable.

Delphi Examples:

```
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..79] of AnsiChar;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
{
This example makes a record.  Reports the size of the reocrd,
fills it and reports the contents.
}
type
  CustRec = record
    Name: string[30];
    Phone: string[14];
  end;

procedure TForm1.Button1Click(Sender: TObject);
 var
  P: ^CustRec;
  size: Integer;
begin
  GetMem(P, SizeOf(CustRec));
  Canvas.TextOut(10, 10, 'The size of the record is ' + IntToStr(SizeOf(CustRec)));
  P.Name := 'a12345678901234567890123456789';
  P.Phone := '9,1,4085551212';
  Canvas.TextOut(10, 40, 'Name was assigned this string: ' + P.Name);
  Canvas.TextOut(10, 70, 'Phone was assigned this string: ' + P.Phone);
  size := SizeOf(CustRec);
```

```
  FreeMem (P, size);
end;
```

See Also

FillChar (see page 3670)

GetMem (see page 3683)

Move (see page 3711)

38.3.459 System.Slice Function

Returns a sub-section of an array.

Pascal

```
function Slice(var A: array; Count: Integer): array;
```

C++

```
array Slice(array A, int Count);
```

File

System

Description

In Delphi code, use Slice to select a sub-section (slice) of an array to be passed as an open array parameter. Slice is only allowed as a parameter in a call to a procedure or function that expects an open array parameter. Slice selects the first Count elements of the array given by A to be passed as an open array parameter. A can be an open array parameter itself, allowing dynamically allocated arrays to be passed to open array parameters.

See Also

SetLength (see page 3748)

IDH

38.3.460 System.Sqr Function

Returns the square of a number.

Pascal

```
function Sqr(X: Extended): Extended; overload;
function Sqr(X: Integer): Integer; overload;
```

C++

```
Extended Sqr(Extended X);
int Sqr(int X);
```

File

System

Description

In Delphi code, the Sqr function returns the square of the argument.

X is a floating-point expression. The result, of the same type as X, is the square of X, or X*X.

Delphi Examples:

```
{
This example uses the System Sqr and Sqrt functions to calculate
the some mathematical values.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqr(5.5):3:1, Temp);
   S := '5.5 squared is ' + Temp + #13#10;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqrt(2.0):5:4, Temp);
   S := S + 'The square root of 2 is ' + Temp;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

See Also

Sqrt (see page 3763)

Power

38.3.461 System.Sqrt Function

Returns the square root of X.

Pascal

```
function Sqrt(const X: Extended): Extended;
```

C++

```
Extended Sqrt(const Extended X);
```

File

System

Description

X is a Delphi floating-point expression. The result is the square root of X.

Delphi Examples:

```
{
This example uses the System Sqr and Sqrt functions to calculate
the some mathematical values.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqr(5.5):3:1, Temp);
```

```
    S := '5.5 squared is ' + Temp + #13#10;
    MessageDlg(S, mtInformation, [mbOk], 0);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  S, Temp: string;
begin
    Str(Sqrt(2.0):5:4, Temp);
    S := S + 'The square root of 2 is ' + Temp;
    MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

See Also

Sqr (see page 3762)

38.3.462 System.Str Function

Formats a string and returns it to a variable.

Pascal

```
procedure Str(X: Width; var S); overload;
procedure Str(X: Decimals; var S); overload;
```

C++

```
Str(Width X,  S);
Str(Decimals X,  S);
```

File

System

Description

In Delphi code, Str converts X to a string representation according to the Width and Decimals formatting parameters. The effect is like a call to Write (see page 3789) except the resulting string is stored in S instead of being written to a text file.

X is an integer-type or real-type expression. Width and Decimals are integer-type expressions. S is a string-type variable or a zero-based character array variable if extended syntax is enabled.

Delphi Examples:

```
{
This example makes a simple string conversion routine
and then prints an integer to the screen.
}

function MakeItAString(I: Longint): string;
{ Convert any integer type to a string }
var
  S: string[11];
begin
  Str(I, S);
  Result:= S;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.TextOut(10, 10, MakeItAString(-5322));
end;
```

See Also

Write (see page 3789)

38.3.463 System.StringCodePage Function

Pascal

```
function StringCodePage(const S: RawByteString): Word; overload;
```

C++

```
Word StringCodePage(const RawByteString S);
```

File

System

Description

This is function System.StringCodePage.

38.3.464 System.StringCodePage Function

Pascal

```
function StringCodePage(const S: UnicodeString): Word; overload;
```

C++

```
Word StringCodePage(const UnicodeString S);
```

File

System

Description

This is function System.StringCodePage.

38.3.465 System.StringElementSize Function

Pascal

```
function StringElementSize(const S: RawByteString): Word; overload;
```

C++

```
Word StringElementSize(const RawByteString S);
```

File

System

Description

This is function System.StringElementSize.

38.3.466 System.StringElementSize Function

Pascal

```
function StringElementSize(const S: UnicodeString): Word; overload;
```

C++

```
Word StringElementSize(const UnicodeString S);
```

File

System

Description

This is function System.StringElementSize.

38.3.467 System.StringOfChar Function

Returns a string with a specified number of repeating characters.

Pascal

```
function StringOfChar(ch: AnsiChar; Count: Integer): AnsiString; overload;
function StringOfChar(ch: WideChar; Count: Integer): UnicodeString; overload;
```

C++

```
AnsiString StringOfChar(AnsiChar ch, int Count);
UnicodeString StringOfChar(WideChar ch, int Count);
```

File

System

Description

In Delphi code, StringOfChar returns a string containing Count characters with the character value given by Ch. For example,

S := StringOfChar('A', 10);

sets S to the string 'AAAAAAAAAA'.

See Also

FillChar (see page 3670)

DupeString

38.3.468 System.StringRefCount Function

Pascal

```
function StringRefCount(const S: RawByteString): Longint; overload;
```

C++

```
Longint StringRefCount(const RawByteString S);
```

File

System

Description

This is function System.StringRefCount.

38.3.469 System.StringRefCount Function

Pascal

```
function StringRefCount(const S: UnicodeString): Longint; overload;
```

C++

```
Longint StringRefCount(const UnicodeString S);
```

File

System

Description

This is function System.StringRefCount.

38.3.470 System.StringToOleStr Function

Allocates memory and copies an AnsiStringa string to the OLE format.

Pascal

```
function StringToOleStr(const Source: AnsiString): PWideChar; overload;
function StringToOleStr(const Source: UnicodeString): PWideChar; overload;
```

C++

```
PWideChar StringToOleStr(const AnsiString Source);
PWideChar StringToOleStr(const UnicodeString Source);
```

File

System

Description

StringToOleStr allocates memory and copies string data from a native AnsiString into a format that can be passed to OLE functions.

See Also

OleStrToString (see page 3715)

OleStrToStrVar (see page 3715)

StringToWideChar (see page 3768)

38.3.471 System.StringToWideChar Function

Returns a UNICODE string from an AnsiString.

Pascal

```
function StringToWideChar(const Source: UnicodeString; Dest: PWideChar; DestSize: Integer):
PWideChar;
```

C++

```
PWideChar StringToWideChar(const UnicodeString Source, PWideChar Dest, int DestSize);
```

File

System

Description

StringToWideChar converts a string from ANSI to UNICODE and stores the result in a specified buffer.

Source is the string to convert. It must include at most DestSize – 1 characters.

Dest is the buffer into which StringToWideChar writes the corresponding UNICODE string. The caller allocates this memory to include at least DestSize wide characters. Following the call, Dest contains at most DestSize - 1 characters, terminated by a NULL wide character.

DestSize is the size of the buffer Dest.

StringToWideChar returns a pointer to Dest.

See Also

WideCharToString (see page 3788)

WideCharLenToString (see page 3787)

WideCharToStrVar (see page 3788)

WideCharlenToStrVar

38.3.472 System.Succ Function

Returns the successor of an argument.

Pascal

```
procedure Succ(X);
```

C++

```
Succ( X);
```

File

System

Description

In Delphi code, Succ returns the successor of the argument, X.

X is an expression of an ordinal type (including Int64). The result, of the same type as X, is the successor of X.

Delphi Examples:

```
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Pred (see page 3721)

38.3.473 System.Swap Function

Exchanges high order byte with the low order byte of an integer or word.

Pascal

```
procedure Swap(X);
```

C++

```
Swap( X);
```

File

System

Description

In Delphi code, Swap exchanges the high-order bytes with the low-order bytes of the argument. X is an expression of type SmallInt, as a 16-bit value, or Word. This is provided for backward compatibility only.

See Also

Hi (see page 3694)

Lo (see page 3704)

38.3.474 System.SysAllocMem Function

Pascal

```
function SysAllocMem(Size: Cardinal): Pointer;
```

C++

```
void * SysAllocMem(unsigned Size);
```

File

System

Description

This is function System.SysAllocMem.

38.3.475 System.SysFreeMem Function

Frees the memory pointed to by a specified pointer.

Pascal

```
function SysFreeMem(P: Pointer): Integer;
```

C++

```
int SysFreeMem(void * P);
```

File

System

Description

When implementing a custom memory manager, use SysFreeMem to free the memory pointed to by the P parameter.

Note: On Windows, the System (see page 3165) unit provides a limited implementation of SysFreeMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysFreeMem provided by the ShareMem (see page 359) unit. When used, ShareMem (see page 359) must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3165) unit provides an implementation of SysFreeMem that supports reference counting and shared objects. The ShareMem (see page 359) unit is not provided for Linux.

See Also

GetHeapStatus (see page 3681)

SysGetMem (see page 3770)

SetMemoryManager (see page 3751)

38.3.476 System.SysGetMem Function

Allocates a specified number of bytes and returns a pointer to them.

Pascal

```
function SysGetMem(Size: Integer): Pointer;
```

C++

```
void * SysGetMem(int Size);
```

File

System

Description

When implementing a custom memory manager, use SysGetMem to dynamically allocate Size bytes of memory.

Note: On Windows, the System (see page 3165) unit provides a limited implementation of SysGetMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysGetMem provided by the ShareMem (see page 359) unit. When used, ShareMem (see page 359) must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3165) unit provides an implementation of SysGetMem that supports reference counting and shared objects. The ShareMem (see page 359) unit is not provided for Linux.

See Also

SysFreeMem (see page 3770)

SysReallocMem (see page 3771)

SetMemoryManager (see page 3751)

38.3.477 System.SysReallocMem Function

Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.

Pascal

```
function SysReallocMem(P: Pointer; Size: Integer): Pointer;
```

C++

```
void * SysReallocMem(void * P, int Size);
```

File

System

Description

When implementing a custom memory manager, use SysReallocMem to change the size of a dynamically allocated chunk of memory. If SysReallocMem can't expand the memory block point to by P, it frees the referenced memory and copies the values to the newly allocated memory that is returned. If Size is less than the number of bytes pointed to by P, only the first Size bytes are preserved. Otherwise, the entire value of the memory pointed to by P is preserved in the newly allocated memory.

Note: On Windows, the System (see page 3165) unit provides a limited implementation of SysReallocMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysReallocMem provided by the ShareMem (see page 359) unit. When used, ShareMem (see page 359) must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3165) unit provides an implementation of SysReallocMem that supports reference counting and shared objects. The ShareMem (see page 359) unit is not provided for Linux.

See Also

SysFreeMem (see page 3770)

SysGetMem (see page 3770)

SetMemoryManager (see page 3751)

38.3.478 System.SysRegisterExpectedMemoryLeak Function

Pascal

```
function SysRegisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysRegisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysRegisterExpectedMemoryLeak.

38.3.479 System.SysUnregisterExpectedMemoryLeak Function

Pascal

```
function SysUnregisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysUnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysUnregisterExpectedMemoryLeak.

38.3.480 System.TextStart Function

Pascal

```
procedure TextStart; deprecated;
```

C++

```
TextStart();
```

File

System

Description

This is function System.TextStart.

38.3.481 System.Trunc Function

Truncates a real number to an integer.

Pascal

```
function Trunc(X: Extended): Int64;
```

C++

```
Int64 Trunc(Extended X);
```

File

System

Description

In Delphi code, the Trunc function truncates a real-type value to an integer-type value. X is a real-type expression. Trunc returns an Int64 value that is the value of X rounded toward zero.

If the truncated value of X is not within the Int64 range, an EInvalidOp exception is raised.

Delphi Examples:

```
{
This example demonstrates how the Trunc function operates
on various real numbers.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' Truncs to ' + IntToStr(Trunc(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.5)) + #13#10;
   Str(1.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.6)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.5)) + #13#10;
   Str(-1.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.6)) + #13#10;
   Str(2.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(2.5)) + #13#10;
   Str(-2.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-2.5)) + #13#10;
   Str(-2.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-2.6));
   MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Int (see page 3699)

Round (see page 3739)

Floor

Ceil

38.3.482 System.Truncate Function

Deletes all the records after the current file position.

Pascal

```
procedure Truncate(var F);
```

C++

```
Truncate( F);
```

File

System

Description

In Delphi code, Call Truncate to cause the current file position to become end-of-file (Eof (see page 3659)(F) is true).

F is a file variable of any type except a text file. Truncate does not work on text files. F must be open.

Delphi Examples:

```
{
This example shows how to truncate a file after the position
set by the last read statement.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Integer;
   i,j: Integer;
 begin
   AssignFile(f,'TEST.INT');
   Rewrite(f);
   for i := 1 to 6 do
     Write(f,i);
   Writeln('File before truncation:');
   Reset(f);
   while not Eof(f) do
   begin
     Read(f,i);
     Writeln(i);
   end;
   Reset(f);
   for i := 1 to 3 do
     Read(f,j); { Read ahead 3 records }
   Truncate(f); { Cut file off here }
   Writeln;
   Writeln('File after truncation:');
   Reset(f);
   while not Eof(f) do
   begin
     Read(f,i);
     Writeln(i);
   end;
   CloseFile(f);
   Erase(f);
 end;
```

See Also

Reset (see page 3734)

Rewrite (see page 3736)

Seek (see page 3741)

38.3.483 System.UCS4StringToUnicodeString Function

Pascal

```
function UCS4StringToUnicodeString(const S: UCS4String): UnicodeString;
```

C++

```
UnicodeString UCS4StringToUnicodeString(const UCS4String S);
```

File

System

Description

This is function System.UCS4StringToUnicodeString.

38.3.484 System.UCS4StringToWideString Function

Converts a string of 4-byte Unicode characters into a WideString (see page 3476) value.

Pascal

```
function UCS4StringToWideString(const S: UCS4String): WideString;
```

C++

```
BSTR UCS4StringToWideString(const UCS4String S);
```

File

System

Description

Call UCS4StringToWideString to convert a string of UCS4 (4-byte Unicode) characters into a WideString (see page 3476) value. UCS4 strings are returned by some Linux operating system calls.

See Also

WideStringToUCS4String (see page 3789)

PUCS4Chars (see page 3724)

UnicodeToUtf8 (see page 3776)

Utf8ToUnicode (see page 3783)

38.3.485 System.UnicodeStringToUCS4String Function

Pascal

```
function UnicodeStringToUCS4String(const S: UnicodeString): UCS4String;
```

C++

```
UCS4String UnicodeStringToUCS4String(const UnicodeString S);
```

File

System

Description

This is function System.UnicodeStringToUCS4String.

38.3.486 System.UnicodeToUtf8 Function

Converts a string of Unicode characters into a UTF-8 string.

Pascal

```
function UnicodeToUtf8(Dest: PChar; MaxDestBytes: Cardinal; Source: PWideChar; SourceChars:
Cardinal): Cardinal; overload;
function UnicodeToUtf8(Dest: PChar; Source: PWideChar; MaxBytes: Integer): Integer; overload;
deprecated;
function UnicodeToUtf8(Dest: PAnsiChar; Source: PWideChar; MaxBytes: Integer): Integer;
overload; deprecated;
function UnicodeToUtf8(Dest: PAnsiChar; MaxDestBytes: Cardinal; Source: PWideChar;
SourceChars: Cardinal): Cardinal; overload;
```

C++

```
unsigned UnicodeToUtf8(const char * Dest, unsigned MaxDestBytes, PWideChar Source, unsigned
SourceChars);
int UnicodeToUtf8(const char * Dest, PWideChar Source, int MaxBytes);
int UnicodeToUtf8(PAnsiChar Dest, PWideChar Source, int MaxBytes);
unsigned UnicodeToUtf8(PAnsiChar Dest, unsigned MaxDestBytes, PWideChar Source, unsigned
SourceChars);
```

File

System

Description

Call UnicodeToUtf8 to convert a Unicode string to a null-terminated sequence of UTF-8 characters. Use the second syntax. The deprecated syntax is included only for backward compatibility.

Dest receives the resulting null-terminated array of UTF-8 characters.

MaxDestBytes or MaxBytes indicates the number of bytes available in the buffer specified by Dest, including the byte for the null terminator.

Source is an array of Unicode characters.

SourceChars is the number of characters in Source. If SourceChars is not specified, Source must be null-terminated.

UnicodeToUtf8 returns the number of bytes written to Dest.

See Also

UCS4StringToWideString (see page 3775)

Utf8ToUnicode (see page 3783)

Utf8Encode

Utf8Decode

UTF8String (see page 3873)

38.3.487 System.UniqueString Function

Ensures that a given string has a reference count of one.

Pascal

```
procedure UniqueString(var str: AnsiString); overload;
procedure UniqueString(var str: WideString); overload;
procedure UniqueString(var str: UnicodeString); overload;
```

C++

```
UniqueString(AnsiString str);
UniqueString(BSTR str);
UniqueString(UnicodeString str);
```

File

System

Description

UniqueString forces the reference count on an AnsiString to one, copying the string in memory if necessary. For normal string handling, there is no need to call UniqueString. UniqueString is used only in cases where an application modifies the contents of a string after

Note: casting the string to a PChar or PWideChar in Delphi.

Note: using the c_str() method of AnsiString or the c_bstr() method of WideString (see page 3476) in C++.

See Also

SetString (see page 3757)

SetLength (see page 3748)

38.3.488 System.UnregisterExpectedMemoryLeak Function

Unregisters an expected memory leak.

Pascal

```
function UnregisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean UnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

UnregisterExpectedMemoryLeak removes a memory location from the Memory Manager's list of expected memory leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak (see page 3731)

ReportMemoryLeakOnShutdown

38.3.489 System.UnregisterModule Function

Pascal

```
procedure UnregisterModule(LibModule: PLibModule);
```

C++

```
UnregisterModule(PLibModule LibModule);
```

File

System

Description

This is function System.UnregisterModule.

38.3.490 System.UpCase Function

Converts a character to uppercase.

Pascal

```
function UpCase(Ch: AnsiChar): AnsiChar; overload;
function UpCase(Ch: WideChar): WideChar; overload;
```

C++

```
AnsiChar UpCase(AnsiChar Ch);
WideChar UpCase(WideChar Ch);
```

File

System

Description

UpCase converts a character to uppercase. Ch is an expression of type Char. Character values not in the range a..z are unaffected.

C++ Examples:

```
/*
The following code uppercases every other character in the
text of an edit control when the user clicks the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  // Get string from TEdit control
  UnicodeString s = Edit1->Text;
  for (int i = 1; i <= s.Length(); i += 2)
    s[i] = System::UpCase(s[i]);
  Edit1->Text = s;
}
```

Delphi Examples:

```
{
The following code uppercases every other character in the
text of an edit control when the user clicks the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
  i : Integer;
begin
  { Get string from TEdit control }
  s := Edit1.Text;
  for i := 1 to Length(s) do
    if i mod 2 = 0 then s[i] := System.UpCase(s[i]);
  Edit1.Text := s;
end;
```

See Also

AnsiUpperCase

StrUpper

UpperCase

38.3.491 System.UTF8Decode Function

Converts a UTF8 string to a Unicode string (WideString (see page 3476)).

Pascal

```
function UTF8Decode(const S: RawByteString): WideString; deprecated;
```

C++

```
BSTR UTF8Decode(const RawByteString S);
```

File

System

Description

Call UTF8Decode to convert a UTF-8 string to Unicode. S is a string encoded in UTF-8. UTF8Decode returns the corresponding WideString (see page 3476) value that represents the string in Unicode.

See Also

UnicodeToUtf8 (see page 3776)

UTF8Encode (see page 3780)

Utf8ToUnicode (see page 3783)

Utf8ToAnsi (see page 3781)

AnsiToUtf8 (see page 3624)

38.3.492 System.UTF8Encode Function

Converts a WideString (see page 3476) value to a UTF8 string.

Pascal

```
function UTF8Encode(const WS: WideString): RawByteString; overload;
function UTF8Encode(const US: UnicodeString): RawByteString; overload;
function UTF8Encode(const A: RawByteString): RawByteString; overload;
```

C++

```
RawByteString UTF8Encode(const BSTR WS);
RawByteString UTF8Encode(const UnicodeString US);
RawByteString UTF8Encode(const RawByteString A);
```

File

System

Description

Call Utf8Encode to convert a Unicode string to UTF-8. WS is the Unicode string to convert. Utf8Encode returns the corresponding UTF-8 string.

See Also

UnicodeToUtf8 (see page 3776)

UTF8Decode (see page 3779)

Utf8ToUnicode (see page 3783)

38.3.493 System.UTF8EncodeToShortString Function

Pascal

```
function UTF8EncodeToShortString(const A: RawByteString): ShortString; overload;
```

C++

```
ShortString UTF8EncodeToShortString(const RawByteString A);
```

File

System

Description

This is function System.UTF8EncodeToShortString.

38.3.494 System.UTF8EncodeToShortString Function

Pascal

```
function UTF8EncodeToShortString(const US: UnicodeString): ShortString; overload;
```

C++

```
ShortString UTF8EncodeToShortString(const UnicodeString US);
```

File

System

Description

This is function System.UTF8EncodeToShortString.

38.3.495 System.UTF8EncodeToShortString Function

Pascal

```
function UTF8EncodeToShortString(const WS: WideString): ShortString; overload;
```

C++

```
ShortString UTF8EncodeToShortString(const BSTR WS);
```

File

System

Description

This is function System.UTF8EncodeToShortString.

38.3.496 System.Utf8ToAnsi Function

Converts a UTF8 string to a string of Ansi characters.

Pascal

```
function Utf8ToAnsi(const S: RawByteString): string;
```

C++

```
UnicodeString Utf8ToAnsi(const RawByteString S);
```

File

System

Description

Call Utf8ToAnsi to convert a UTF-8 string to Ansi. S is a string encoded in UTF-8. Utf8ToAnsi returns the corresponding string that uses the Ansi character set.

See Also

UnicodeToUtf8 (see page 3776)

Utf8Encode

Utf8ToUnicode (see page 3783)

Utf8Decode

AnsiToUtf8 (see page 3624)

38.3.497 System.UTF8ToString Function

Pascal

```
function UTF8ToString(const S: PAnsiChar): string;
```

C++

```
UnicodeString UTF8ToString(const PAnsiChar S);
```

File

System

Description

This is function System.UTF8ToString.

38.3.498 System.UTF8ToString Function

Pascal

```
function UTF8ToString(const S: RawByteString): string;
```

C++

```
UnicodeString UTF8ToString(const RawByteString S);
```

File

System

Description

This is function System.UTF8ToString.

38.3.499 System.UTF8ToString Function

Pascal

```
function UTF8ToString(const S: ShortString): string;
```

C++

```
UnicodeString UTF8ToString(const ShortString S);
```

File

System

Description

This is function System.UTF8ToString.

38.3.500 System.UTF8ToString Function

Pascal

```
function UTF8ToString(const S: array of AnsiChar): string; overload;
```

C++

```
UnicodeString UTF8ToString(const array of AnsiChar S);
```

File

System

Description

This is function System.UTF8ToString.

38.3.501 System.Utf8ToUnicode Function

Converts a string of UTF-8 characters to a Unicode string.

Pascal

```
function Utf8ToUnicode(Dest: PWideChar; Source: PAnsiChar; MaxChars: Integer): Integer;
overload; deprecated;
function Utf8ToUnicode(Dest: PWideChar; MaxDestChars: Cardinal; Source: PAnsiChar;
SourceBytes: Cardinal): Cardinal; overload;
```

C++

```
int Utf8ToUnicode(PWideChar Dest, PAnsiChar Source, int MaxChars);
unsigned Utf8ToUnicode(PWideChar Dest, unsigned MaxDestChars, PAnsiChar Source, unsigned
SourceBytes);
```

File

System

Description

Call Utf8ToUnicode to convert a UTF-8 string to Unicode. Use the second syntax. The deprecated syntax is provided for backward compatibility only.

Dest receives the resulting null-terminated array of Unicode characters.

MaxDestChars or MaxChars indicates the maximum number of Unicode characters that the buffer specified by Dest can hold, not counting the null terminator.

Source is an array of UTF-8 characters.

SourceBytes is the number of bytes in Source. If SourceBytes is not specified, Source must be null-terminated.

Utf8ToUnicode returns the number of characters written to Dest.

In case of errors, the Utf8ToUnicode ... : Cardinal variant returns `Cardinal(-1) == $FFFFFFFF` and the Utf8ToUnicode ... : Integer returns `-1`.

See Also

UCS4StringToWideString (see page 3775)

UnicodeToUtf8 (see page 3776)

Utf8Encode

Utf8Decode

UTF8String (see page 3873)

38.3.502 System.UTF8ToUnicodeString Function

Pascal

```
function UTF8ToUnicodeString(const S: PAnsiChar): UnicodeString; overload;
```

C++

```
UnicodeString UTF8ToUnicodeString(const PAnsiChar S);
```

File

System

Description

This is function System.UTF8ToUnicodeString.

38.3.503 System.UTF8ToUnicodeString Function

Pascal

```
function UTF8ToUnicodeString(const S: RawByteString): UnicodeString; overload;
```

C++

```
UnicodeString UTF8ToUnicodeString(const RawByteString S);
```

File

System

Description

This is function System.UTF8ToUnicodeString.

38.3.504 System.UTF8ToUnicodeString Function

Pascal

```
function UTF8ToUnicodeString(const S: ShortString): UnicodeString; overload;
```

C++

```
UnicodeString UTF8ToUnicodeString(const ShortString S);
```

File

System

Description

This is function System.UTF8ToUnicodeString.

38.3.505 System.UTF8ToWideString Function

Pascal

```
function UTF8ToWideString(const S: RawByteString): WideString;
```

C++

```
BSTR UTF8ToWideString(const RawByteString S);
```

File

System

Description

This is function System.UTF8ToWideString.

38.3.506 System.Val Function

Converts a string to a numeric representation.

Pascal

```
procedure Val(S; var V; var Code: Integer);
```

C++

```
Val( S,  V, int Code);
```

File

System

Description

In Delphi code, Val converts the string value S to its numeric representation, as if it were read from a text file with Read (see page 3727).

S is a string-type expression; it must be a sequence of characters that form a signed real number.

V is an integer-type or real-type variable. If V is an integer-type variable, S must form a whole number.

Code is a variable of type Integer.

If the string is invalid, the index of the offending character is stored in Code; otherwise, Code is set to zero. For a null-terminated string, the error position returned in Code is one larger than the actual zero-based index of the character in error.

Delphi Examples:

```
{
This example uses the System Val function to convert a string to
an integer.  Enter an integer in the text edit and then click the
button.  An error will occur if the value entered is not an integer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    I, Code: Integer;
    mystr: string;
  begin
    { Get text from TEdit control }
    mystr := Edit1.Text;
    Val(mystr, I, Code);
    { Error during conversion to integer? }
    if Code <> 0 then
      MessageDlg('Error at position: ' + IntToStr(Code), mtWarning, [mbOk], 0, mbOk)
    else
      Canvas.TextOut(10, 10, 'Value = ' + IntToStr(I));
  end;
```

See Also

Str (see page 3764)

38.3.507 System.VarArrayRedim Function

Resizes a Variant (see page 3430) array.

Pascal

```
procedure VarArrayRedim(A: Variant; HighBound: Integer);
```

C++

```
VarArrayRedim(Variant A, int HighBound);
```

File

System

Description

VarArrayRedim resizes the given Variant (see page 3430) array by changing the high bound of the rightmost dimension to the given value. The A parameter can be a Variant (see page 3430) or an OleVariant (see page 3281), and it must be possible to assign a value to A (A must be an lvalue).

Existing elements of the array are preserved, and new elements are set to zero or empty. An EVariantArrayCreateError exception is thrown if the Variant (see page 3430) given by A is not an array, or if there is not enough memory to resize the array.

Note: A Variant (see page 3430) array can't be resized while it is locked.

See Also

VarArrayCreate

VarArrayLock

EVariantArrayCreateError

38.3.508 System.VarClear Function

Empties a Variant (see page 3430), so that it is unassigned.

Pascal

```
procedure VarClear(V: Variant);
```

C++

```
VarClear(Variant V);
```

File

System

Description

Calling VarClear is equivalent to assigning the Unassigned constant to the Variant (see page 3430). V can be either a Variant (see page 3430) or an OleVariant (see page 3281), but it must be possible to assign a value to it (it must be an lvalue).

After calling VarClear, the VarIsEmpty function returns true, and the VarType function returns varEmpty (see page 3933). Using an unassigned variant in an expression causes an exception to be thrown. Likewise, if you attempt to convert an unassigned Variant (see page 3430) to another type (using VarAsType), an exception is thrown.

Note: Do not confuse clearing a Variant (see page 3430), which leaves it unassigned, with assigning a Null value. A Null Variant (see page 3430) is still assigned, but has the value Null. Unlike unassigned Variants (see page 4997), Null Variants (see page 4997) can be used in expressions and can be converted to other types of Variants (see page 4997).

See Also

Unassigned

VarIsEmpty

VarAsType

VarType

38.3.509 System.WideCharLenToString Function

Converts Unicode characters to single or multi byte character data.

Pascal

```
function WideCharLenToString(Source: PWideChar; SourceLen: Integer): UnicodeString;
```

C++

```
UnicodeString WideCharLenToString(PWideChar Source, int SourceLen);
```

File

System

Description

WideCharLenToString converts SourceLen Unicode characters from the buffer given by Source to a string made up of single or multi byte characters.

See Also

StringToWideChar (see page 3768)

WideCharToString (see page 3788)

38.3.510 System.WideCharLenToStrVar Function

WideCharLenToStrVar converts Unicode characters to single or multi byte character data.

Pascal

```
procedure WideCharLenToStrVar(Source: PWideChar; SourceLen: Integer; var Dest: UnicodeString);
overload;
procedure WideCharLenToStrVar(Source: PWideChar; SourceLen: Integer; var Dest: AnsiString);
overload;
```

C++

```
WideCharLenToStrVar(PWideChar Source, int SourceLen, UnicodeString Dest);
WideCharLenToStrVar(PWideChar Source, int SourceLen, AnsiString Dest);
```

File

System

Description

The WideCharToStrVar (see page 3788) procedure converts SourceLen WideChar (Unicode) characters from Source into single or multi byte character data, and copies the result into Dest.

See Also

WideCharToString (see page 3788)

38.3.511 System.WideCharToString Function

Converts null-terminated Unicode string to single or multi byte character data.

Pascal

```
function WideCharToString(Source: PWideChar): UnicodeString;
```

C++

```
UnicodeString WideCharToString(PWideChar Source);
```

File

System

Description

WideCharToString converts the given null-terminated Unicode string to a string made up of single- or multi-byte characters.

See Also

StringToWideChar (see page 3768)

WideCharLenToString (see page 3787)

38.3.512 System.WideCharToStrVar Function

Converts Unicode string to a single or multi byte character data.

Pascal

```
procedure WideCharToStrVar(Source: PWideChar; var Dest: UnicodeString);
```

C++

```
WideCharToStrVar(PWideChar Source, UnicodeString Dest);
```

File

System

Description

WideCharToStrVar converts UNICODE string Source into single or multi byte character data, and copies the result into Dest. The source must be null-terminated.

See Also

WideCharToString (see page 3788)

38.3.513 System.WideStringToUCS4String Function

Converts a WideString (see page 3476) value to a string of 4-byte Unicode characters.

Pascal

```
function WideStringToUCS4String(const S: WideString): UCS4String;
```

C++

```
UCS4String WideStringToUCS4String(const BSTR S);
```

File

System

Description

Call WideStringToUCS4String to convert a WideString (see page 3476) value to a string of UCS4 (4-byte Unicode) characters. UCS4 strings are used by some Linux operating system calls.

38.3.514 System.Write Function

Writes to either a typed file or a text file

Pascal

```
procedure Write(P1); overload;
procedure Write(P1; Pn); overload;
procedure Write(var F: Text; P1); overload;
procedure Write(var F: Text; P1; Pn); overload;
```

C++

```
Write( P1);
Write( P1,  Pn);
Write(Text F,  P1);
Write(Text F,  P1,  Pn);
```

File

System

Description

The syntax shown here for the Write procedure is an example that illustrates that Write can take a variable number of arguments.

Write procedure for typed files:

In Delphi code, Write writes a file to a file component. F is a file variable, and each V is a variable of the same type as the component type of F. For each variable written, the current file position is advanced to the next component. If the current file position is at the end of the file (that is, if Eof (see page 3659)(F) is true), the file is expanded.

Write procedure for text files:

In Delphi code, Write writes one or more values to a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Output (see page 3901) is assumed. Each P is a write parameter. Each write parameter includes an output expression whose value is to be written to the file. A write parameter can also contain the specifications of a field width and a number of decimal places. Each output expression must be of a type Char, one of the Integer types (Byte, Shortint, Word, Longint, Cardinal), one of the floating-point types (Single, Real, Double, Extended, Currency (see page 3242)), one of the string types (PChar, AnsiString, ShortString), a packed string, or one of the Boolean types (Boolean, Bool).

A write parameter has the form:

```
OutExpr [: MinWidth [: DecPlaces ] ]
```

where OutExpr is an output expression. MinWidth and DecPlaces are type integer expressions.

MinWidth specifies the minimum field width, which must be greater than 0. Exactly MinWidth characters are written (using leading blanks if necessary) except when OutExpr has a value that must be represented in more than MinWidth characters. In that case, enough characters are written to represent the value of OutExpr. Likewise, if MinWidth is omitted, then the necessary number of characters is written to represent the value of OutExpr.

DecPlaces specifies the number of decimal places in a fixed-point representation of one of the Real types. It can be specified only if OutExpr is one of the Real types, and if MinWidth is also specified. When MinWidth is specified, it must be greater than or equal to 0.

Write with a character-type value: If MinWidth is omitted, the character value of OutExpr is written to the file. Otherwise, MinWidth - 1 blanks followed by the character value of OutExpr is written.

Write with one of the integer type values: If MinWidth is omitted, the decimal representation of OutExpr is written to the file with no preceding blanks. If MinWidth is specified and its value is larger than the length of the decimal string, enough blanks are written before the decimal string to make the field width MinWidth.

Write with one of the real type values: If OutExpr has one of the real type values, its decimal representation is written to the file. The format of the representation depends on the presence or absence of DecPlaces. If DecPlaces is omitted (or if it's present but has a negative value), a floating-point decimal string is written. If MinWidth is also omitted, a default MinWidth of 17 is assumed; otherwise, if MinWidth is less than 8, it's assumed to be 8. The format of the floating-point string is:

```
[   | - ] <digit> . <decimals> E [ + | - ] <exponent>
```

The following table lists the components of the output string.

Component	Meaning
[| —]	" “ or “-”|, according to the sign of OutExpr
<digit>	Single digit, “0” only if OutExpr is 0
<decimals>	Digit string of MinWidth-7 (but at most 10) digits
E	Uppercase [E] character

[+ | —]	According to the sign of the exponent
<exponent>	Two-digit decimal exponent

If DecPlaces is present, a fixed-point decimal string is written. If DecPlaces is larger than 11, it's assumed to be 11. The format of the fixed-point string follows:

```
[<blanks>] [ - ] <digits> [.<decimals>]
```

The following table lists the components of the fixed-point string:

Component	Meaning
[<blanks>	Blanks to satisfy MinWidth
[-]	If OutExpr is negative
<digits>	At least one digit, but no leading zeros
[.<decimals>]	Decimals if DecPlaces > 0

Write with one of the string-type values: If MinWidth is omitted, the string value of OutExpr is written to the file with no leading blanks. If MinWidth is specified, and its value is larger than the length of OutExpr, enough blanks are written before the decimal string to make the field width MinWidth.

Write with a packed string-type value: If OutExpr is of packed string type, the effect is the same as writing a string whose length is the number of elements in the packed string type.

Write with one of the Boolean type values: If OutExpr is of type Boolean, the effect is the same as writing the strings true or false, depending on the value of OutExpr.

Note: When using Write the file must be open for output.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

WriteLn (see page 3792)

38.3.515 System.WriteLn Function

Writes to a text file and adds an end-of-line marker.

Pascal

```
procedure WriteLn(P1); overload;
procedure WriteLn(P1; Pn); overload;
procedure WriteLn(var F: Text; P1); overload;
procedure WriteLn(var F: Text; P1; Pn); overload;
```

C++

```
WriteLn( P1);
WriteLn( P1,  Pn);
WriteLn(Text F,  P1);
WriteLn(Text F,  P1,  Pn);
```

File

System

Description

WriteLn is an extension to the Write (see page 3789) procedure, as it is defined for text files.

The syntax shown here for the WriteLn procedure is an example that illustrates that WriteLn can take a variable number of arguments.

After executing Write (see page 3789), WriteLn writes an end-of-line marker (line feed or carriage return/line feed) to the file.

If F is omitted, the global variable Output (see page 3901) is used, which accesses the processes standard input file. Use of Output (see page 3901) in GUI applications raises special issues.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}

{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
```

See Also

Write (see page 3789)

SetLineBreakStyle (see page 3750)

Output (see page 3901)

38.3.516 System.YieldProcessor Function

Pascal

```
procedure YieldProcessor;
```

C++

```
YieldProcessor();
```

File

System

Description

This is function System.YieldProcessor.

38.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TChunkStatus (see page 3796)	TChunkStatus enumerates the possible states of a chunk of memory.
	TMinimumBlockAlignment (see page 3807)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TRuntimeError (see page 3809)	This is record System.TRuntimeError.
	TTextLineBreakStyle (see page 3811)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Records

	Name	Description
	PackageInfoTable (see page 3795)	This is record System.PackageInfoTable.
	PackageUnitEntry (see page 3795)	This is record System.PackageUnitEntry.
	TCallDesc (see page 3795)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3830) is a pointer to this descriptior.
	TCVModInfo (see page 3797)	This is record System.TCVModInfo.
	TDispatchMessage (see page 3797)	This is record System.TDispatchMessage.
	TDispDesc (see page 3798)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 3798)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 3799)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 3800)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapStatus (see page 3802)	This is record System::THeapStatus.
	TInitContext (see page 3803)	This is record System.TInitContext.
	TInterfaceEntry (see page 3803)	TInterfaceTable (see page 3804) lists the interfaces that a class supports, and PInterfaceTable (see page 3839) points to this list.
	TInterfaceTable (see page 3804)	This is record System.TInterfaceTable.
	TLibModule (see page 3804)	This is record System::TLibModule.
	TMemoryManager (see page 3805)	This is record System::TMemoryManager.
	TMemoryManagerEx (see page 3805)	This is record System::TMemoryManagerEx.
	TMemoryManagerState (see page 3806)	This is record System::TMemoryManagerState.
	TMethod (see page 3806)	TMethod stores Code and Data fields to represent a method.
	TModuleUnloadRec (see page 3808)	This is record System::TModuleUnloadRec.
	TMonitorSupport (see page 3808)	This is record System.TMonitorSupport.
	TResStringRec (see page 3809)	This is record System::TResStringRec.
	TSmallBlockTypeState (see page 3810)	This is record System::TSmallBlockTypeState.
	TTextRec (see page 3811)	TTextRec is the internal format for text files.
	TVarArray (see page 3812)	This is record System::TVarArray.
	TVarArrayBound (see page 3813)	This is record System::TVarArrayBound.
	TVarData (see page 3813)	TVarData represents Variant (see page 3430) information.
	TVariantManager (see page 3817)	This is record System::TVariantManager.
	TVarRec (see page 3819)	This is class System::TVarRec.

Structures

	Name	Description
	THeapBlock (see page 3802)	This is record System::THeapBlock.
	TCallDesc (see page 3820)	This is type System::TCallDesc.
	TDispDesc (see page 3820)	This is type System::TDispDesc.

38.4.1 System.PackageInfoTable Record

Pascal

```
PackageInfoTable = packed record
  UnitCount: Integer;
  UnitInfo: PUnitEntryTable;
end;
```

C++

```
struct PackageInfoTable {
  int UnitCount;
  PUnitEntryTable UnitInfo;
};
```

File

System

Description

This is record System.PackageInfoTable.

38.4.2 System.PackageUnitEntry Record

Pascal

```
PackageUnitEntry = packed record
  FInit: Pointer;
  Init: Pointer;
end;
```

C++

```
struct PackageUnitEntry {
  void * FInit;
  void * Init;
};
```

File

System

Description

This is record System.PackageUnitEntry.

38.4.3 System.TCallDesc Record

TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3830) is a pointer to this description.

Pascal

```
TCallDesc = packed record
  CallType: Byte;
  ArgCount: Byte;
```

```
    NamedArgCount: Byte;
    ArgTypes: array[0..255] of Byte;
  end;
```

C++

```
  struct TCallDesc {
    Byte CallType;
    Byte ArgCount;
    Byte NamedArgCount;
    array[0..255] of Byte ArgTypes;
  };
```

File

System

Description

TCallDesc identifies a method call that was made using a Variant (see page 3430).

CallType identifies the method that was called.

ArgCount indicates the total number of arguments passed to the method.

NamedArgCount

ArgTypes is an array giving the type codes for each argument.

See Also

PCallDesc (see page 3830)

38.4.4 System.TChunkStatus Enumeration

TChunkStatus enumerates the possible states of a chunk of memory.

Pascal

```
  TChunkStatus = (
    csUnallocated,
    csAllocated,
    csReserved,
    csSysAllocated,
    csSysReserved
  );
```

C++

```
  enum TChunkStatus {
    csUnallocated,
    csAllocated,
    csReserved,
    csSysAllocated,
    csSysReserved
  };
```

File

System

Description

GetMemoryMap (see page 3691) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap (see page 3862) obtains a TChunkStatus status value for each of them in the TMemoryMap (see page 3862) array. TChunkStatus defines the

following values :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 3691)

TMemoryMap (see page 3862)

38.4.5 System.TCVModInfo Record

Pascal

```
TCVModInfo = packed record
  ModName: PAnsiChar;
  LibName: PAnsiChar;
  UserData: Pointer;
end;
```

C++

```
struct TCVModInfo {
  PAnsiChar ModName;
  PAnsiChar LibName;
  void * UserData;
};
```

File

System

Description

This is record System.TCVModInfo.

38.4.6 System.TDispatchMessage Record

Pascal

```
TDispatchMessage = record
  MsgID: Word;
end;
```

C++

```
struct TDispatchMessage {
  Word MsgID;
};
```

File

System

Description

This is record System.TDispatchMessage.

38.4.7 System.TDispDesc Record

Pascal

```
TDispDesc = packed record
  DispID: Integer;
  ResType: Byte;
  CallDesc: TCallDesc;
end;
```

C++

```
struct TDispDesc {
  int DispID;
  Byte ResType;
  TCallDesc CallDesc;
};
```

File

System

Description

This is record System.TDispDesc.

38.4.8 System.TDynArrayTypeInfo Record

Pascal

```
TDynArrayTypeInfo = packed record
  kind: Byte;
  name: string[0];
  elSize: Longint;
  elType: ^PDynArrayTypeInfo;
  varType: Integer;
end;
```

C++

```
struct TDynArrayTypeInfo {
  Byte kind;
  string[0] name;
  Longint elSize;
  ^PDynArrayTypeInfo elType;
  int varType;
};
```

File

System

Description

This is record System.TDynArrayTypeInfo.

38.4.9 System.TFileRec Record

TFileRec is the internal format for typed and untyped files in the Delphi language.

Pascal

```
TFileRec = packed record
  Handle: Integer;
  Mode: Word;
  Flags: Word;
  case Byte of
    0: (RecSize: Cardinal;);
    1: (BufSize: Cardinal;
        BufPos: Cardinal;
        BufEnd: Cardinal;
        BufPtr: PAnsiChar;
        OpenFunc: Pointer;
        InOutFunc: Pointer;
        FlushFunc: Pointer;
        CloseFunc: Pointer;
        UserData: array[1..32] of Byte;
        Name: array[0..259] of WideChar;);
end;
```

C++

```
struct TFileRec {
  int Handle;
  Word Mode;
  Word Flags;
  unsigned RecSize;
  unsigned BufSize;
  unsigned BufPos;
  unsigned BufEnd;
  PAnsiChar BufPtr;
  void * OpenFunc;
  void * InOutFunc;
  void * FlushFunc;
  void * CloseFunc;
  array[1..32] of Byte UserData;
  array[0..259] of WideChar Name;
};
```

File

System

Description

TFileRec is the internal format for typed and untyped files. TFileRec enables you to typecast a Delphi file variable to access its internal fields.

Note: You would normally never declare a variable of this type.

38.4.10 System.TGUID Record

TGUID is a structured form of the value that uniquely identifies an interface.

Pascal

```
TGUID = packed record
  D1: LongWord;
  D2: Word;
  D3: Word;
  D4: array[0..7] of Byte;
end;
```

C++

```
struct TGUID {
  LongWord D1;
  Word D2;
  Word D3;
  array[0..7] of Byte D4;
};
```

File

System

Description

An interface declaration can specify a globally unique identifier (GUID), represented by a string literal enclosed in brackets immediately preceding the member list. The GUID part of the declaration must have the form :

['{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}']

where each x is a hexadecimal digit (0 through 9 or A through F). On Windows, the Type Library editor automatically generates GUIDs for new interfaces. You can also generate GUIDs by pressing Ctrl+Shift+G in the Code editor.

TGUID provides a structured access to this GUID :

D1 maps to the first 8 hexadecimal digits.

D2 maps to the next 4 hexadecimal digits.

D3 maps to the next 4 hexadecimal digits.

D4 maps to the final 8 hexadecimal digits.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
```

```
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
  if IsEqualGUID(MyGuid0, MyGuid1) then
     Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                     '2 randomly generated GUIDs cannot be equal!');
end;
```

38.4.11 System::THeapBlock Structure

Pascal

```
THeapBlock = record
  Start: Pointer;
  Size: unsigned int;
end;
```

C++

```
struct THeapBlock {
  void * Start;
  unsigned int Size;
};
```

File

sysclass.h

Description

This is record System::THeapBlock.

38.4.12 System::THeapStatus Record

Pascal

```
THeapStatus = record
  TotalAddrSpace: Cardinal;
  TotalUncommitted: Cardinal;
  TotalCommitted: Cardinal;
  TotalAllocated: Cardinal;
  TotalFree: Cardinal;
  FreeSmall: Cardinal;
  FreeBig: Cardinal;
  Unused: Cardinal;
  Overhead: Cardinal;
  HeapErrorCode: Cardinal;
end;
```

C++

```
struct THeapStatus {
  Cardinal TotalAddrSpace;
  Cardinal TotalUncommitted;
  Cardinal TotalCommitted;
  Cardinal TotalAllocated;
  Cardinal TotalFree;
  Cardinal FreeSmall;
  Cardinal FreeBig;
  Cardinal Unused;
  Cardinal Overhead;
  Cardinal HeapErrorCode;
};
```

File

System.pas

Description

This is record System::THeapStatus.

38.4.13 System.TInitContext Record

Pascal

```
TInitContext = record
  OuterContext: PInitContext;
  ExcFrame: Pointer;
  InitTable: PackageInfo;
  InitCount: Integer;
  Module: PLibModule;
  DLLSaveEBP: Pointer;
  DLLSaveEBX: Pointer;
  DLLSaveESI: Pointer;
  DLLSaveEDI: Pointer;
  ExitProcessTLS: procedure;
  DLLInitState: Byte;
  PkgLoaderLock: Pointer;
end;
```

C++

```
struct TInitContext {
  PInitContext OuterContext;
  void * ExcFrame;
  PackageInfo InitTable;
  int InitCount;
  PLibModule Module;
  void * DLLSaveEBP;
  void * DLLSaveEBX;
  void * DLLSaveESI;
  void * DLLSaveEDI;
  procedure ExitProcessTLS;
  Byte DLLInitState;
  void * PkgLoaderLock;
};
```

File

System

Description

This is record System.TInitContext.

38.4.14 System.TInterfaceEntry Record

TInterfaceTable (see page 3804) lists the interfaces that a class supports, and PInterfaceTable (see page 3839) points to this list.

Pascal

```
TInterfaceEntry = packed record
  IID: TGUID;
  VTable: Pointer;
  IOffset: Integer;
```

```
    ImplGetter: Integer;
  end;
```

C++

```
  struct TInterfaceEntry {
    TGUID IID;
    void * VTable;
    int IOffset;
    int ImplGetter;
  };
```

File

System

Description

PInterfaceTable (see page 3839) is a pointer to a TInterfaceTable (see page 3804) value. TInterfaceTable (see page 3804) contains the following fields:

Field	Meaning
EntryCount	The number of interface descriptors in the table.
Entries	An array of interface descriptors.

See Also

PInterfaceTable (see page 3839)

38.4.15 System.TInterfaceTable Record

Pascal

```
  TInterfaceTable = packed record
    EntryCount: Integer;
    Entries: array[0..9999] of TInterfaceEntry;
  end;
```

C++

```
  struct TInterfaceTable {
    int EntryCount;
    array[0..9999] of TInterfaceEntry Entries;
  };
```

File

System

Description

This is record System.TInterfaceTable.

38.4.16 System::TLibModule Record

Pascal

```
  TLibModule = record
    Next: PLibModule;
```

```
    Instance: LongWord;
    CodeInstance: LongWord;
    DataInstance: LongWord;
    ResInstance: LongWord;
    Reserved: Integer;
  end;
```

C++

```
  struct TLibModule {
    PLibModule Next;
    LongWord Instance;
    LongWord CodeInstance;
    LongWord DataInstance;
    LongWord ResInstance;
    Integer Reserved;
  };
```

File

System.pas

Description

This is record System::TLibModule.

38.4.17 System::TMemoryManager Record

Pascal

```
  TMemoryManager = record
    GetMem: function(Size: Integer): Pointer;
    FreeMem: function(P: Pointer): Integer;
    ReallocMem: function(P: Pointer; Size: Integer): Pointer;
  end;
```

C++

```
  struct TMemoryManager {
    function(Size: Integer): Pointer GetMem;
    function(P: Pointer): Integer FreeMem;
    function(P: Pointer; Size: Integer): Pointer ReallocMem;
  };
```

File

System.pas

Description

This is record System::TMemoryManager.

38.4.18 System::TMemoryManagerEx Record

Pascal

```
  TMemoryManagerEx = record
    GetMem: function(Size: Integer): Pointer;
    FreeMem: function(P: Pointer): Integer;
    ReallocMem: function(P: Pointer; Size: Integer): Pointer;
    AllocMem: function(Size: Cardinal): Pointer;
    RegisterExpectedMemoryLeak: function(P: Pointer): Boolean;
```

```
    UnregisterExpectedMemoryLeak: function(P: Pointer): Boolean;
  end;
```

C++

```
  struct TMemoryManagerEx {
    function(Size: Integer): Pointer GetMem;
    function(P: Pointer): Integer FreeMem;
    function(P: Pointer; Size: Integer): Pointer ReallocMem;
    function(Size: Cardinal): Pointer AllocMem;
    function(P: Pointer): Boolean RegisterExpectedMemoryLeak;
    function(P: Pointer): Boolean UnregisterExpectedMemoryLeak;
  };
```

File

System.pas

Description

This is record System::TMemoryManagerEx.

38.4.19 System::TMemoryManagerState Record

Pascal

```
  TMemoryManagerState = packed record
    SmallBlockTypeStates: TSmallBlockTypeStates;
    AllocatedMediumBlockCount: Cardinal;
    TotalAllocatedMediumBlockSize: Cardinal;
    ReservedMediumBlockAddressSpace: Cardinal;
    AllocatedLargeBlockCount: Cardinal;
    TotalAllocatedLargeBlockSize: Cardinal;
    ReservedLargeBlockAddressSpace: Cardinal;
  end;
```

C++

```
  struct TMemoryManagerState {
    TSmallBlockTypeStates SmallBlockTypeStates;
    Cardinal AllocatedMediumBlockCount;
    Cardinal TotalAllocatedMediumBlockSize;
    Cardinal ReservedMediumBlockAddressSpace;
    Cardinal AllocatedLargeBlockCount;
    Cardinal TotalAllocatedLargeBlockSize;
    Cardinal ReservedLargeBlockAddressSpace;
  };
```

File

System.pas

Description

This is record System::TMemoryManagerState.

38.4.20 System.TMethod Record

TMethod stores Code and Data fields to represent a method.

Pascal

```
TMethod = record
  Code: Pointer;
  Data: Pointer;
end;
```

C++

```
struct TMethod {
  void * Code;
  void * Data;
};
```

File

System

Description

The TMethod stores the Code and Data pointers for a method. This type can be used in a type cast of a method pointer to access the code and data parts of the method pointer.

You can also furnish a TMethod variable by assigning Data a pointer to an object, and assigning Code using System::TObject::MethodAddress (see page 3390), specifying the method name as a string parameter to that method.

See Also

System::TObject::MethodAddress (see page 3390)

38.4.21 System.TMinimumBlockAlignment Enumeration

TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.

Pascal

```
TMinimumBlockAlignment = (
  mba8Byte,
  mba16Byte
);
```

C++

```
enum TMinimumBlockAlignment {
  mba8Byte,
  mba16Byte
};
```

File

System

Description

The TMinimumBlockAlignment type defines either 8 byte (**mba8Byte**) or 16 byte (**mba16Byte**) block alignments. Namely, that the memory address is divisible exactly by 8 or 16 respectively.

See Also

AllocMem (see page 3622)

GetMemory (see page 3684)

38.4.22 System::TModuleUnloadRec Record

Pascal

```
TModuleUnloadRec = record
  Next: PModuleUnloadRec;
  Proc: TModuleUnloadProcLW;
end;
```

C++

```
struct TModuleUnloadRec {
  PModuleUnloadRec Next;
  TModuleUnloadProcLW Proc;
};
```

File

System.pas

Description

This is record System::TModuleUnloadRec.

38.4.23 System.TMonitorSupport Record

Pascal

```
TMonitorSupport = record
  NewSyncObject: function: Pointer;
  FreeSyncObject: procedure (SyncObject: Pointer);
  NewWaitObject: function: Pointer;
  FreeWaitObject: procedure (WaitObject: Pointer);
  WaitAndOrSignalObject: function (SignalObject, WaitObject: Pointer; Timeout: Cardinal):
Cardinal;
end;
```

C++

```
struct TMonitorSupport {
  function: Pointer NewSyncObject;
  procedure (SyncObject: Pointer) FreeSyncObject;
  function: Pointer NewWaitObject;
  procedure (WaitObject: Pointer) FreeWaitObject;
  function (SignalObject, WaitObject: Pointer; Timeout: Cardinal): Cardinal
WaitAndOrSignalObject;
};
```

File

System

Description

This is record System.TMonitorSupport.

38.4.24 System::TResStringRec Record

Pascal

```
TResStringRec = packed record
  Module: ^Cardinal;
  Identifier: Integer;
end;
```

C++

```
struct TResStringRec {
  ^Cardinal Module;
  Integer Identifier;
};
```

File

System.pas

Description

This is record System::TResStringRec.

38.4.25 System.TRuntimeError Enumeration

Pascal

```
TRuntimeError = (
  reNone,
  reOutOfMemory,
  reInvalidPtr,
  reDivByZero,
  reRangeError,
  reIntOverflow,
  reInvalidOp,
  reZeroDivide,
  reOverflow,
  reUnderflow,
  reInvalidCast,
  reAccessViolation,
  rePrivInstruction,
  reControlBreak,
  reStackOverflow,
  reVarTypeCast,
  reVarInvalidOp,
  reVarDispatch,
  reVarArrayCreate,
  reVarNotArray,
  reVarArrayBounds,
  reAssertionFailed,
  reExternalException,
  reIntfCastError,
  reSafeCallError,
  reMonitorNotLocked,
  reNoMonitorSupport
);
```

C++

```
enum TRuntimeError {
  reNone,
  reOutOfMemory,
  reInvalidPtr,
  reDivByZero,
  reRangeError,
  reIntOverflow,
  reInvalidOp,
  reZeroDivide,
  reOverflow,
  reUnderflow,
  reInvalidCast,
  reAccessViolation,
  rePrivInstruction,
  reControlBreak,
  reStackOverflow,
  reVarTypeCast,
  reVarInvalidOp,
  reVarDispatch,
  reVarArrayCreate,
  reVarNotArray,
  reVarArrayBounds,
  reAssertionFailed,
  reExternalException,
  reIntfCastError,
  reSafeCallError,
  reMonitorNotLocked,
  reNoMonitorSupport
};
```

File

System

Description

This is record System.TRuntimeError.

38.4.26 System::TSmallBlockTypeState Record

Pascal

```
TSmallBlockTypeState = packed record
  InternalBlockSize: Cardinal;
  UseableBlockSize: Cardinal;
  AllocatedBlockCount: Cardinal;
  ReservedAddressSpace: Cardinal;
end;
```

C++

```
struct TSmallBlockTypeState {
  Cardinal InternalBlockSize;
  Cardinal UseableBlockSize;
  Cardinal AllocatedBlockCount;
  Cardinal ReservedAddressSpace;
};
```

File

System.pas

Description

This is record System::TSmallBlockTypeState.

38.4.27 System.TTextLineBreakStyle Enumeration

TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Pascal

```
TTextLineBreakStyle = (
  tlbsLF,
  tlbsCRLF
);
```

C++

```
enum TTextLineBreakStyle {
  tlbsLF,
  tlbsCRLF
};
```

File

System

Description

TTextLineBreakStyle can have one of the following values:

Value	Meaning
tlbsLF	Linefeed only
tlbsCRLF	Carriage return followed by Linefeed

38.4.28 System.TTextRec Record

TTextRec is the internal format for text files.

Pascal

```
TTextRec = packed record
  Handle: Integer;
  Mode: Word;
  Flags: Word;
  BufSize: Cardinal;
  BufPos: Cardinal;
  BufEnd: Cardinal;
  BufPtr: PAnsiChar;
  OpenFunc: Pointer;
  InOutFunc: Pointer;
  FlushFunc: Pointer;
  CloseFunc: Pointer;
  UserData: array[1..32] of Byte;
  Name: array[0..259] of WideChar;
  Buffer: TTextBuf;
end;
```

C++

```
struct TTextRec {
  int Handle;
  Word Mode;
  Word Flags;
  unsigned BufSize;
  unsigned BufPos;
  unsigned BufEnd;
  PAnsiChar BufPtr;
  void * OpenFunc;
  void * InOutFunc;
  void * FlushFunc;
  void * CloseFunc;
  array[1..32] of Byte UserData;
  array[0..259] of WideChar Name;
  TTextBuf Buffer;
};
```

File

System

Description

TTextRec is the internal format of a Delphi variable of type text. You would never declare a variable of this type. However, you can use TTextRec to typecast a text file variable to access the internal data fields, such as the file name.

TextBuf is the type of the Buffer entry in the TTextRec record, and indicates the size of the buffer used.

Note: Do not use this unless you are familiar with writing text file device drivers for the Delphi language.

38.4.29 System::TVarArray Record

Pascal

```
TVarArray = packed record
  DimCount: Word;
  Flags: Word;
  ElementSize: Integer;
  LockCount: Integer;
  Data: Pointer;
  Bounds: TVarArrayBoundArray;
end;
```

C++

```
struct TVarArray {
  Word DimCount;
  Word Flags;
  Integer ElementSize;
  Integer LockCount;
  Pointer Data;
  TVarArrayBoundArray Bounds;
};
```

File

System.pas

Description

This is record System::TVarArray.

38.4.30 System::TVarArrayBound Record

Pascal

```
TVarArrayBound = packed record
  ElementCount: Integer;
  LowBound: Integer;
end;
```

C++

```
struct TVarArrayBound {
  Integer ElementCount;
  Integer LowBound;
};
```

File

System.pas

Description

This is record System::TVarArrayBound.

38.4.31 System.TVarData Record

TVarData represents Variant (see page 3430) information.

Pascal

```
TVarData = packed record
  case Integer of
    0: (VType: TVarType;
        Reserved1: Word;
        Reserved2: Word;
        Reserved3: Word;);
    1: (VLongs: array[0..2] of LongInt);
        RawData: array [0..3] of LongInt););
    2: (VWords: array [0..6] of Word);
        VBytes: array [0..13] of Byte););
    varAny: (VAny: Pointer;);
    varArray: (VArray: PVarArray;);
    varBoolean: (VBoolean: WordBool;);
    varByRef: (VPointer: Pointer;);
    varByte: (VByte: Byte;);
    varCurrency: (VCurrency: Currency;);
    varDate: (VDate: TDateTime;);
    varDispatch: (VDispatch: Pointer;);
    varDouble: (VDouble: Double;);
    varError: (VError: HRESULT;);
    varInt64: (VInt64: Int64;);
    varInteger: (VInteger: Integer;);
    varLongWord: (VLongWord: LongWord;);
    varOleStr: (VOleStr: PWideChar;);
    varShortInt: (VShortInt: ShortInt;);
    varSingle: (VSingle: Single;);
    varSmallInt: (VSmallInt: SmallInt;);
    varString: (VString: Pointer;);
    varUInt64: (VUInt64: UInt64;);
    varUnknown: (VUnknown: Pointer;);
```

```
    varUString: (VUString: Pointer;);
    varWord: (VWord: Word;);
end;
```

C++

```
struct TVarData {
  TVarType VType;
  Word Reserved1;
  Word Reserved2;
  Word Reserved3;
  SmallInt VSmallInt;
  int VInteger;
  Single VSingle;
  Double VDouble;
  Currency VCurrency;
  TDateTime VDate;
  PWideChar VOleStr;
  void * VDispatch;
  HRESULT VError;
  WordBool VBoolean;
  void * VUnknown;
  ShortInt VShortInt;
  Byte VByte;
  Word VWord;
  LongWord VLongWord;
  Int64 VInt64;
  UInt64 VUInt64;
  void * VString;
  void * VAny;
  PVarArray VArray;
  void * VPointer;
  void * VUString;
  array[0..2] of LongInt) VLongs;
  array [0..6] of Word) VWords;
  array [0..13] of Byte) VBytes;
  array [0..3] of LongInt) RawData;
};
```

File

System

Description

TVarData is the base class for Variant (see page 3430).

Note: The internal structure of a Variant (see page 3430) as designated by TVarData is identical to the Variant (see page 3430) type used by OLE and the Win32 APIs.

struct TVarData

```
  {
union
    {
      Word     VType;    // Delphi-compatible - Variant Type member
      VARTYPE  vt;       // tagVARIANT compatible member
    };
    Word Reserved1;
    Word Reserved2;
    Word Reserved3;
union
    {
      // Delphi-compatible Variant members
      Smallint      VSmallint;    //  iVal
      Integer       VInteger;     //  lVal
      Single        VSingle;      //  fltVal
      Double        VDouble;      //  dblVal
```

```
      CurrencyBase  VCurrency;    //  cyVal
      TDateTimeBase VDate;        //  date
      PWideChar     VOleStr;      //  bstrVal
      IDispatch*    VDispatch;    //  pdispVal
      HResult       VError;       //  scode
      WordBool      VBoolean;     //  boolVal
      IUnknown*     VUnknown;     //  punkVal
      Byte          VByte;        //  bVal
      Pointer       VString;      //  ??????
      PVarArray     VArray;       //  parray
      Pointer       VPointer;     //  byref
      // tagVARIANT compatible members (from OAIDL.H)
      // Allowing all types marked as [V] (may appear in a VARIANT to be initialized
      LONG          lVal;
      BYTE          bVal;
      SHORT         iVal;
      FLOAT         fltVal;
      DOUBLE        dblVal;
      VARIANT_BOOL  boolVal;
      SCODE         scode;
      CY            cyVal;
      DATE          date;
      BSTR          bstrVal;
      IUnknown     *punkVal;
      IDispatch    *pdispVal;
      SAFEARRAY    *parray;
      BYTE         *pbVal;SHORT        *piVal;
      LONG         *plVal;
      FLOAT        *pfltVal;
      DOUBLE       *pdblVal;
      VARIANT_BOOL *pboolVal;
      SCODE        *pscode;
      CY           *pcyVal;
      DATE         *pdate;
      BSTR         *pbstrVal;
      IUnknown    **ppunkVal;
      IDispatch   **ppdispVal;
      SAFEARRAY   **pparray;
      VARIANT      *pvarVal;
      PVOID         byref;
      CHAR          cVal;
      USHORT        uiVal;
      ULONG         ulVal;
      INT           intVal;
      UINT          uintVal;
      DECIMAL      *pdecVal;
      CHAR         *pcVal;
      USHORT       *puiVal;
      ULONG        *pulVal;
      INT          *pintVal;
      UINT         *puintVal;
    };
  };
```

The VType of TVarData contains the type code of the variant in the lower twelve bits (the bits defined by varTypeMask (see page 3939)).

The following values are used for VType:

#define varEmpty (see page 3933) (Byte)(0)

```
#define varNull (ByteByte_Type)(1)
#define varSmallint (ByteByte_Type)(2)
#define varInteger (ByteByte_Type)(3)
#define varSingle (ByteByte_Type)(4)
#define varDouble (ByteByte_Type)(5)
#define varCurrency (ByteByte_Type)(6)
```

```
#define varDate (ByteByte_Type)(7)
#define varOleStr (ByteByte_Type)(8)
#define varDispatch (ByteByte_Type)(9)
#define varError (ByteByte_Type)(10)
#define varBoolean (ByteByte_Type)(11)
#define varVariant (ByteByte_Type)(12)
#define varUnknown (ByteByte_Type)(13)
#define varByte (ByteByte_Type)(17)
#define varString (WordWord_Type)(256)
#define varTypeMask (WordWord_Type)(4095)
#define varArray (WordWord_Type)(8192)
#define varByRef (WordWord_Type)(16384)
#define varStrArg (ByteByte_Type)(72)
#define vtInteger (ByteByte_Type)(0)
#define vtBoolean (ByteByte_Type)(1)
#define vtChar (ByteByte_Type)(2)
#define vtExtended (ByteByte_Type)(3)
#define vtString (ByteByte_Type)(4)
#define vtPointer (ByteByte_Type)(5)
#define vtPChar (ByteByte_Type)(6)
#define vtObject (ByteByte_Type)(7)
#define vtClass (ByteByte_Type)(8)
#define vtWideChar (ByteByte_Type)(9)
#define vtPWideChar (ByteByte_Type)(10)
#define vtAnsiString (ByteByte_Type)(11)
#define vtCurrency (ByteByte_Type)(12)
#define vtVariant (ByteByte_Type)(13)
```

In addition, the varArray (see page 3929) bit may be set to indicate that the variant is an array, and the varByRef (see page 3930) bit may be set to indicate that the variant contains a reference as opposed to a value.

Word Reserved1;

```
    Word Reserved2;
    Word Reserved3;
```

The Reserved1, Reserved2, and Reserved3 members of TVarData are unused.

union { Smallint VSmallint;

```
      Integer VInteger;
      Single VSingle;
      Double VDouble;
      CurrencyBase VCurrency;
      TDateTimeBase VDate;
      PWideChar VOleStr;
      Ole2::IDispatch* VDispatch;
      Integer VError;
      WordBool VBoolean;
      Ole2::IUnknown* VUnknown;
      Byte VByte;
      Pointer VString;
      PVarArray VArray;
      Pointer VPointer;
    };
};
```

The contents of the remaining eight bytes of TVarData depend on the VType field. If neither the varArray (see page 3929) nor the varByRef (see page 3930) bits are set, the variant contains a value of the given type.

If the varArray (see page 3929) bit is set, the variant contains a pointer to a TVarArray (see page 3812) structure that defines the specifics of the array. The type of each array element is given by the varTypeMask (see page 3939) bits in VType.

If the varByRef (see page 3930) bit is set, the variant contains a reference to a value of the type given by the varTypeMask (see page 3939) and varArray (see page 3929) bits in VType.

C++ Examples:

```
/*
This example shows how to use the TVarData type in typecasts
of Variant variables to access the internals of a variable.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Variant V;
  AnsiString mystr;
  if (TVarData(V).VType == varEmpty)
    ListBox1->Items->Add("Variants start empty.");
  VarClear(V);
  TVarData(V).VType = varString;
  mystr = "Here is my string";
  std::auto_ptr<char> MyBuffer(new char[mystr.Length() + 1]);
  StrCopy(MyBuffer.get(), mystr.t_str());
  TVarData(V).VString  = MyBuffer.get();
  mystr = PAnsiChar(TVarData(V).VString);
  ListBox1->Items->Add("This variant is now a string: " + mystr);
  VarClear(V);
  TVarData(V).VType = varInteger;
  TVarData(V).VInteger = 1234567;
  ListBox1->Items->Add("This variant is now an integer: " + IntToStr(TVarData(V).VInteger));
}
```

Delphi Examples:

```
{{
This example shows how to use the TVarData type in typecasts
of Variant variables to access the internals of a variable.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  V: Variant;
  mystr: string;
  MyBuffer: PChar;
begin
  if TVarData(V).VType = varEmpty then
    ListBox1.Items.Add('Variants start empty.');
  VarClear(V);
  TVarData(V).VType := varString;
  mystr := 'Here is my string';
  MyBuffer:= GetMemory(Length(mystr));
  StrCopy(MyBuffer, PChar(mystr));
  TVarData(V).VString := MyBuffer;
  ListBox1.Items.Add('This variant is now a string: ' + PChar(TVarData(V).VString));
  VarClear(V);
  TVarData(V).VType := varInteger;
  TVarData(V).VInteger := 1234567;
  ListBox1.Items.Add('This variant is now an integer: ' + IntToStr(TVarData(V).VInteger));
end;
```

38.4.32 System::TVariantManager Record

Pascal

```
TVariantManager = record
  VarClear: procedure(var V : Variant);
  VarCopy: procedure(var Dest: Variant; const Source: Variant);
  VarCopyNoInd: procedure;
  VarCast: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarCastOle: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarToInt: function(const V: Variant): Integer;
```

```
    VarToInt64: function(const V: Variant): Int64;
    VarToBool: function(const V: Variant): Boolean;
    VarToReal: function(const V: Variant): Extended;
    VarToCurr: function(const V: Variant): Currency;
    VarToPStr: procedure(var S; const V: Variant);
    VarToLStr: procedure(var S: string; const V: Variant);
    VarToWStr: procedure(var S: WideString; const V: Variant);
    VarToIntf: procedure(var Unknown: IInterface; const V: Variant);
    VarToDisp: procedure(var Dispatch: IDispatch; const V: Variant);
    VarToDynArray: procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
    VarFromInt: procedure(var V: Variant; const Value: Integer; const Range: ShortInt);
    VarFromInt64: procedure(var V: Variant; const Value: Int64);
    VarFromBool: procedure(var V: Variant; const Value: Boolean);
    VarFromReal: procedure;
    VarFromTDateTime: procedure;
    VarFromCurr: procedure;
    VarFromPStr: procedure(var V: Variant; const Value: ShortString);
    VarFromLStr: procedure(var V: Variant; const Value: string);
    VarFromWStr: procedure(var V: Variant; const Value: WideString);
    VarFromIntf: procedure(var V: Variant; const Value: IInterface);
    VarFromDisp: procedure(var V: Variant; const Value: IDispatch);
    VarFromDynArray: procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
    OleVarFromPStr: procedure(var V: OleVariant; const Value: ShortString);
    OleVarFromLStr: procedure(var V: OleVariant; const Value: string);
    OleVarFromVar: procedure(var V: OleVariant; const Value: Variant);
    OleVarFromInt: procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt);
    OleVarFromInt64: procedure(var V: OleVariant; const Value: Int64);
    VarOp: procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp);
    VarCmp: procedure(const Left, Right: TVarData; const OpCode: TVarOp);
    VarNeg: procedure(var V: Variant);
    VarNot: procedure(var V: Variant);
    DispInvoke: procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
  Pointer);
    VarAddRef: procedure(var V: Variant);
    VarArrayRedim: procedure(var A : Variant; HighBound: Integer);
    VarArrayGet: function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant;
    VarArrayPut: procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices:
  Integer);
    WriteVariant: function(var T: Text; const V: Variant; Width: Integer): Pointer;
    Write0Variant: function(var T: Text; const V: Variant): Pointer;
  end;
```

C++

```
  struct TVariantManager {
    procedure(var V : Variant) VarClear;
    procedure(var Dest: Variant; const Source: Variant) VarCopy;
    procedure VarCopyNoInd;
    procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCast;
    procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCastOle;
    function(const V: Variant): Integer VarToInt;
    function(const V: Variant): Int64 VarToInt64;
    function(const V: Variant): Boolean VarToBool;
    function(const V: Variant): Extended VarToReal;
    function(const V: Variant): Currency VarToCurr;
    procedure(var S; const V: Variant) VarToPStr;
    procedure(var S: string; const V: Variant) VarToLStr;
    procedure(var S: WideString; const V: Variant) VarToWStr;
    procedure(var Unknown: IInterface; const V: Variant) VarToIntf;
    procedure(var Dispatch: IDispatch; const V: Variant) VarToDisp;
    procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer) VarToDynArray;
    procedure(var V: Variant; const Value: Integer; const Range: ShortInt) VarFromInt;
    procedure(var V: Variant; const Value: Int64) VarFromInt64;
    procedure(var V: Variant; const Value: Boolean) VarFromBool;
    procedure VarFromReal;
    procedure VarFromTDateTime;
    procedure VarFromCurr;
```

```
    procedure(var V: Variant; const Value: ShortString) VarFromPStr;
    procedure(var V: Variant; const Value: string) VarFromLStr;
    procedure(var V: Variant; const Value: WideString) VarFromWStr;
    procedure(var V: Variant; const Value: IInterface) VarFromIntf;
    procedure(var V: Variant; const Value: IDispatch) VarFromDisp;
    procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer) VarFromDynArray;
    procedure(var V: OleVariant; const Value: ShortString) OleVarFromPStr;
    procedure(var V: OleVariant; const Value: string) OleVarFromLStr;
    procedure(var V: OleVariant; const Value: Variant) OleVarFromVar;
    procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt) OleVarFromInt;
    procedure(var V: OleVariant; const Value: Int64) OleVarFromInt64;
    procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp) VarOp;
    procedure(const Left, Right: TVarData; const OpCode: TVarOp) VarCmp;
    procedure(var V: Variant) VarNeg;
    procedure(var V: Variant) VarNot;
    procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params: Pointer)
  DispInvoke;
    procedure(var V: Variant) VarAddRef;
    procedure(var A : Variant; HighBound: Integer) VarArrayRedim;
    function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant VarArrayGet;
    procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices: Integer)
  VarArrayPut;
    function(var T: Text; const V: Variant; Width: Integer): Pointer WriteVariant;
    function(var T: Text; const V: Variant): Pointer Write0Variant;
  };
```

File

System.pas

Description

This is record System::TVariantManager.

38.4.33 System::TVarRec Record

Pascal

```
  TVarRec = record
    case Byte of
      vtAnsiString: (VAnsiString: Pointer;);
      vtBoolean: (VBoolean: Boolean;);
      vtChar: (VChar: AnsiChar;);
      vtClass: (VClass: TClass;);
      vtCurrency: (VCurrency: PCurrency;);
      vtExtended: (VExtended: PExtended;);
      vtInt64: (VInt64: PInt64;);
      vtInteger: (VInteger: Integer;
                  VType: Byte;);
      vtInterface: (VInterface: Pointer;);
      vtObject: (VObject: TObject;);
      vtPChar: (VPChar: PAnsiChar;);
      vtPointer: (VPointer: Pointer;);
      vtPWideChar: (VPWideChar: PWideChar;);
      vtString: (VString: PShortString;);
      vtUnicodeString: (VUnicodeString: Pointer;);
      vtVariant: (VVariant: PVariant;);
      vtWideChar: (VWideChar: WideChar;);
      vtWideString: (VWideString: Pointer;);
  end;
```

C++

```
  struct TVarRec {
```

```
    Integer VInteger;
    Byte VType;
    Boolean VBoolean;
    AnsiChar VChar;
    PExtended VExtended;
    PShortString VString;
    Pointer VPointer;
    PAnsiChar VPChar;
    TObject VObject;
    TClass VClass;
    WideChar VWideChar;
    PWideChar VPWideChar;
    Pointer VAnsiString;
    PCurrency VCurrency;
    PVariant VVariant;
    Pointer VInterface;
    Pointer VWideString;
    PInt64 VInt64;
    Pointer VUnicodeString;
  };
```

File

System.pas

Description

This is class System::TVarRec.

38.4.34 System::TCallDesc Structure

Pascal

```
  TCallDesc = struct TCallDesc@1;
```

C++

```
  typedef struct {
    unsigned char CallType;
    unsigned char ArgCount;
    unsigned char NamedArgCount;
    unsigned char ArgTypes[256];
  } TCallDesc, * PCallDesc;
```

File

systvar.h

Description

This is type System::TCallDesc.

38.4.35 System::TDispDesc Structure

Pascal

```
  TDispDesc = struct TDispDesc@1;
```

C++

```
  typedef struct {
    int DispID;
```

```
    unsigned char ResType;
    TCallDesc CallDesc;
} TDispDesc, * PDispDesc;
```

File

systvar.h

Description

This is type System::TDispDesc.

38.5 Types

The following table lists types in this documentation.

Types

Name	Description
CppLongInt (see page 3824)	This is type System.CppLongInt.
CppULongInt (see page 3825)	This is type System.CppULongInt.
GetPackageInfoTable (see page 3825)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 3825)	This is type System.HGLOBAL.
HINST (see page 3826)	This is type System.HINST.
HMODULE (see page 3826)	This is type System.HMODULE.
HRESULT (see page 3826)	This is type System.HRESULT.
HRSRC (see page 3827)	This is type System.HRSRC.
Int16 (see page 3827)	This is type System.Int16.
Int32 (see page 3827)	This is type System.Int32.
Int8 (see page 3828)	This is type System.Int8.
IntegerArray (see page 3828)	This is type System.IntegerArray.
IUnknown (see page 3828)	IUnknown is a special base interface.
PackageInfo (see page 3829)	This is type System.PackageInfo.
PAnsiString (see page 3829)	PAnsiString is a pointer to an AnsiString variable.
PBoolean (see page 3829)	PBoolean is a pointer to a Boolean variable.
PByte (see page 3830)	PByte is a pointer to a Byte variable.
PCallDesc (see page 3830)	This is type System::PCallDesc.
PCardinal (see page 3831)	PCardinal is a pointer to a Cardinal variable.
PComp (see page 3831)	PComp is a pointer to a Comp variable.
PCurrency (see page 3831)	PCurrency is a pointer to a Currency (see page 3242) variable.
PCVModInfo (see page 3832)	This is type System.PCVModInfo.
PDate (see page 3832)	PDate is a pointer to a Date variable.
PDateTime (see page 3833)	PDateTime is a pointer to a TDateTime (see page 3335) variable.
PDispatch (see page 3833)	This is type System.PDispatch.
PDispDesc (see page 3833)	This is type System::PDispDesc.
PDouble (see page 3834)	PDouble is a pointer to a Double variable.
PDynArrayTypeInfo (see page 3834)	This is type System.PDynArrayTypeInfo.
PError (see page 3834)	This is type System.PError.
PExtended (see page 3835)	PExtended is a pointer to an Extended variable.
PGUID (see page 3835)	PGUID is a pointer to a TGUID (see page 3800) variable.
PInitContext (see page 3837)	This is type System.PInitContext.
PInt64 (see page 3837)	PInt64 is a pointer to an Int64 variable.
PInteger (see page 3837)	PInteger is a pointer to an Integer variable.
PIntegerArray (see page 3838)	PIntegerArray is a pointer to an IntegerArray (see page 3828) variable.
PInterfaceEntry (see page 3838)	PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3803) variable.
PInterfaceTable (see page 3839)	PInterfaceTable is a pointer to a TInterfaceTable (see page 3804) variable.
PLibModule (see page 3839)	PLibModule is a pointer to a TLibModule (see page 3804) variable.
PLongint (see page 3839)	This is type System.PLongint.
PLongWord (see page 3840)	PLongWord is a pointer to a LongWord variable.
PMemoryManager (see page 3840)	This is type System::PMemoryManager.
PMemoryManagerEx (see page 3840)	This is type System::PMemoryManagerEx.
PModuleUnloadRec (see page 3841)	This is type System::PModuleUnloadRec.
PMonitor (see page 3841)	This is type System.PMonitor.

PMonitorSupport (see page 3841)	This is type System.PMonitorSupport.
PointerArray (see page 3842)	This is type System.PointerArray.
POleVariant (see page 3842)	This is type System.POleVariant.
PPAnsiChar (see page 3842)	This is type System.PPAnsiChar.
PPCharArray (see page 3843)	This is type System.PPCharArray.
PPDispatch (see page 3843)	This is type System.PPDispatch.
PPMonitor (see page 3843)	This is type System.PPMonitor.
PPointer (see page 3844)	PPointer is a pointer to a Pointer variable.
PPointerArray (see page 3844)	PPointerArray is a pointer to a PointerArray (see page 3842) variable.
PPUnknown (see page 3844)	This is type System.PPUnknown.
PPWideChar (see page 3845)	This is type System.PPWideChar.
PRawByteString (see page 3845)	This is type System.PRawByteString.
PResStringRec (see page 3845)	This is type System::PResStringRec.
PShortInt (see page 3846)	PShortInt is a pointer to a ShortInt variable.
PShortString (see page 3846)	PShortString is a pointer to a ShortString variable.
PSingle (see page 3846)	PSingle is a pointer to a Single variable.
PSmallBlockTypeState (see page 3847)	This is type System::PSmallBlockTypeState.
PSmallInt (see page 3847)	PSmallInt is a pointer to a SmallInt variable.
PTextBuf (see page 3848)	This is type System.PTextBuf.
PUCS4Char (see page 3848)	This is type System.PUCS4Char.
PUCS4CharArray (see page 3848)	This is type System.PUCS4CharArray.
PUInt64 (see page 3849)	This is type System.PUInt64.
PUnicodeString (see page 3849)	This is type System.PUnicodeString.
PUnitEntryTable (see page 3849)	This is type System.PUnitEntryTable.
PUnknown (see page 3850)	PUnknown is a pointer to an IUnknown (see page 3828) variable.
PUTF8String (see page 3850)	This is type System.PUTF8String.
PVarArray (see page 3850)	This is type System.PVarArray.
PVarArrayBoundArray (see page 3851)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 3851)	This is type System.PVarArrayCoorArray.
PVarData (see page 3851)	This is type System.PVarData.
PVariant (see page 3852)	PVariant is a pointer to a Variant (see page 3430) variable.
PVariantManager (see page 3852)	This is type System::PVariantManager.
PVarRec (see page 3852)	This is type System::PVarRec.
PWideString (see page 3853)	PWideString is a pointer to a WideString (see page 3476) variable.
PWord (see page 3853)	PWord is a pointer to a Word variable.
PWordBool (see page 3853)	PWordBool is a pointer to a WordBool variable.
RawByteString (see page 3854)	RawByteString enables the passing of string data of any code page without doing any codepage conversions.
ResourceString (see page 3854)	This is type System::ResourceString.
TAssertErrorProc (see page 3855)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 3855)	This is type System::TBooleanDynArray.
TBoundArray (see page 3855)	This is type System.TBoundArray.
TByteDynArray (see page 3856)	This is type System::TByteDynArray.
TCardinalDynArray (see page 3856)	This is type System::TCardinalDynArray.
TClass (see page 3856)	TClass defines the metaclass for System::TObject (see page 3364).
TClassHelperBaseClass (see page 3857)	This is type System.TClassHelperBaseClass.
TDate (see page 3857)	This is type System.TDate.
TDateTime (see page 3858)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 3859)	This is type System.TDLLProc.
TDLLProcEx (see page 3859)	This is type System.TDLLProcEx.
TDoubleDynArray (see page 3859)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 3860)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 3860)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 3860)	This is type System.TFileIOFunc.
TInt64DynArray (see page 3861)	This is type System::TInt64DynArray.

TIntegerDynArray (see page 3861)	This is type System::TIntegerDynArray.
TInterfacedClass (see page 3861)	TInterfacedClass defines the metaclass for TInterfacedObject (see page 3355).
TLongWordDynArray (see page 3862)	This is type System::TLongWordDynArray.
TMemoryMap (see page 3862)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3796) values.
TModuleUnloadProc (see page 3863)	TModuleUnloadProc dehines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 3863)	This is type System.TModuleUnloadProcLW.
TPCharArray (see page 3863)	This is type System.TPCharArray.
TResourceHandle (see page 3864)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 3864)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 3864)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 3865)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 3865)	This is type System::TSmallBlockTypeStates.
TSmallIntDynArray (see page 3865)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 3866)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 3866)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 3866)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 3867)	This is type System.TTextBuf.
TTextIOFunc (see page 3867)	This is type System.TTextIOFunc.
TThreadFunc (see page 3867)	This is type System::TThreadFunc.
TTime (see page 3868)	This is type System.TTime.
TUCS4CharArray (see page 3868)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 3868)	This is type System.TVarArrayBoundArray.
TVarArrayCoorArray (see page 3869)	This is type System.TVarArrayCoorArray.
TVarOp (see page 3869)	TVarOp defines a Variant (see page 3430) operation.
TWideStringDynArray (see page 3870)	This is type System::TWideStringDynArray.
TWordDynArray (see page 3870)	This is type System::TWordDynArray.
UCS4Char (see page 3871)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 3871)	UCS4String is used for working with 4–byte Unicode characters.
UInt16 (see page 3872)	This is type System.UInt16.
UInt32 (see page 3872)	This is type System.UInt32.
UInt8 (see page 3872)	This is type System.UInt8.
UnitEntryTable (see page 3873)	This is type System.UnitEntryTable.
UTF8String (see page 3873)	UTF8String represents UTF-8 encoded Unicode strings.

38.5.1 System.CppLongInt Type

Pascal

```
CppLongInt = type LongInt;
```

C++

```
type LongInt CppLongInt;
```

File

System

Description

This is type System.CppLongInt.

38.5.2 System.CppULongInt Type

Pascal

```
CppULongInt = type LongWord;
```

C++

```
type LongWord CppULongInt;
```

File

System

Description

This is type System.CppULongInt.

38.5.3 System.GetPackageInfoTable Type

Pascal

```
GetPackageInfoTable = function : PackageInfo;
```

C++

```
: PackageInfo ( GetPackageInfoTable)();
```

File

System

Description

This is type System.GetPackageInfoTable.

38.5.4 System.HGLOBAL Type

Pascal

```
HGLOBAL = THandle;
```

C++

```
THandle HGLOBAL;
```

File

System

Description

This is type System.HGLOBAL.

38.5.5 System.HINST Type

Pascal

```
HINST = THandle;
```

C++

```
THandle HINST;
```

File

System

Description

This is type System.HINST.

38.5.6 System.HMODULE Type

Pascal

```
HMODULE = HINST;
```

C++

```
HINST HMODULE;
```

File

System

Description

This is type System.HMODULE.

38.5.7 System.HRESULT Type

Pascal

```
HRESULT = type Longint;
```

C++

```
type Longint HRESULT;
```

File

System

Description

This is type System.HRESULT.

38.5.8 System.HRSRC Type

Pascal

```
HRSRC = THandle;
```

C++

```
THandle HRSRC;
```

File

System

Description

This is type System.HRSRC.

38.5.9 System.Int16 Type

Pascal

```
Int16 = SmallInt;
```

C++

```
SmallInt Int16;
```

File

System

Description

This is type System.Int16.

38.5.10 System.Int32 Type

Pascal

```
Int32 = Integer;
```

C++

```
int Int32;
```

File

System

Description

This is type System.Int32.

38.5.11 System.Int8 Type

Pascal

```
Int8 = ShortInt;
```

C++

```
ShortInt Int8;
```

File

System

Description

This is type System.Int8.

38.5.12 System.IntegerArray Type

Pascal

```
IntegerArray = array[0..$efffffff] of Integer;
```

C++

```
array[0..$efffffff] of Integer IntegerArray;
```

File

System

Description

This is type System.IntegerArray.

38.5.13 System.IUnknown Type

IUnknown is a special base interface.

Pascal

```
IUnknown = IInterface;
```

C++

```
IInterface IUnknown;
```

File

System

Description

IUnknown is a base interface with features that are important to C++ programmers and also programmers who utilize COM objects.

Note: In Delphi code, IUnknown is simply another name for IInterface (see page 3503). Descending from IUnknown instead of IInterface (see page 3503) informs the Delphi compiler that the interface must be compatible with COM objects.

Cross-platform applications can use interfaces descended from IUnknown, but must not use them to access COM objects, which are only available under Windows.

See Also

IInterface (see page 3503)

System::DelphiInterface (see page 3263)

38.5.14 System.PackageInfo Type

Pascal

```
PackageInfo = ^PackageInfoTable;
```

C++

```
^PackageInfoTable PackageInfo;
```

File

System

Description

This is type System.PackageInfo.

38.5.15 System.PAnsiString Type

PAnsiString is a pointer to an AnsiString variable.

Pascal

```
PAnsiString = ^AnsiString;
```

C++

```
^AnsiString PAnsiString;
```

File

System

Description

PAnsiString is a pointer to an AnsiString variable.

See Also

AnsiString

38.5.16 System.PBoolean Type

PBoolean is a pointer to a Boolean variable.

Pascal

```
PBoolean = ^Boolean;
```

C++

```
^Boolean PBoolean;
```

File

System

Description

PBoolean is a pointer to a Boolean variable.

See Also

Boolean

38.5.17 System.PByte Type

PByte is a pointer to a Byte variable.

Pascal

```
PByte = ^Byte;
```

C++

```
^Byte PByte;
```

File

System

Description

PByte is a pointer to a Byte variable.

See Also

Byte

38.5.18 System::PCallDesc Type

Pascal

```
PCallDesc = ^TCallDesc;
```

C++

```
^TCallDesc PCallDesc;
```

File

System.pas

Description

This is type System::PCallDesc.

38.5.19 System.PCardinal Type

PCardinal is a pointer to a Cardinal variable.

Pascal

```
PCardinal = ^Cardinal;
```

C++

```
^Cardinal PCardinal;
```

File

System

Description

PCardinal is a pointer to a Cardinal variable.

See Also

Cardinal

38.5.20 System.PComp Type

PComp is a pointer to a Comp variable.

Pascal

```
PComp = ^Comp;
```

C++

```
^Comp PComp;
```

File

System

Description

PComp is a pointer to a Comp variable.

Note: The Comp data type is deprecated.

See Also

Comp

38.5.21 System.PCurrency Type

PCurrency is a pointer to a Currency (see page 3242) variable.

Pascal

```
PCurrency = ^Currency;
```

C++

```
^Currency PCurrency;
```

File

System

Description

PCurrency is a pointer to a Currency (see page 3242) variable.

See Also

Currency (see page 3242)

38.5.22 System.PCVModInfo Type

Pascal

```
PCVModInfo = ^TCVModInfo;
```

C++

```
^TCVModInfo PCVModInfo;
```

File

System

Description

This is type System.PCVModInfo.

38.5.23 System.PDate Type

PDate is a pointer to a Date variable.

Pascal

```
PDate = ^Double;
```

C++

```
^Double PDate;
```

File

System

Description

PDate is a pointer to a Date variable.

See Also

Date

38.5.24 System.PDateTime Type

PDateTime is a pointer to a TDateTime (see page 3335) variable.

Pascal

```
PDateTime = ^TDateTime;
```

C++

```
^TDateTime PDateTime;
```

File

System

Description

PDateTime is a pointer to a TDateTime (see page 3335) variable.

See Also

TDateTime (see page 3858)

38.5.25 System.PDispatch Type

Pascal

```
PDispatch = ^IDispatch;
```

C++

```
^IDispatch PDispatch;
```

File

System

Description

This is type System.PDispatch.

38.5.26 System::PDispDesc Type

Pascal

```
PDispDesc = ^TDispDesc;
```

C++

```
^TDispDesc PDispDesc;
```

File

System.pas

Description

This is type System::PDispDesc.

38.5.27 System.PDouble Type

PDouble is a pointer to a Double variable.

Pascal

```
PDouble = ^Double;
```

C++

```
^Double PDouble;
```

File

System

Description

PDouble is a pointer to a Double variable.

See Also

Double

38.5.28 System.PDynArrayTypeInfo Type

Pascal

```
PDynArrayTypeInfo = ^TDynArrayTypeInfo;
```

C++

```
^TDynArrayTypeInfo PDynArrayTypeInfo;
```

File

System

Description

This is type System.PDynArrayTypeInfo.

38.5.29 System.PError Type

Pascal

```
PError = ^LongWord;
```

C++

```
^LongWord PError;
```

File

System

Description

This is type System.PError.

38.5.30 System.PExtended Type

PExtended is a pointer to an Extended variable.

Pascal

```
PExtended = ^Extended;
```

C++

```
^Extended PExtended;
```

File

System

Description

PExtended is a pointer to an Extended variable.

See Also

Extended

38.5.31 System.PGUID Type

PGUID is a pointer to a TGUID (see page 3800) variable.

Pascal

```
PGUID = ^TGUID;
```

C++

```
^TGUID PGUID;
```

File

System

Description

PGUID is a pointer to a TGUID (see page 3800) variable.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
```

```
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
  if IsEqualGUID(MyGuid0, MyGuid1) then
     Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                     '2 randomly generated GUIDs cannot be equal!');
end;
```

See Also

TGUID (see page 3800)

38.5.32 System.PInitContext Type

Pascal

```
PInitContext = ^TInitContext;
```

C++

```
^TInitContext PInitContext;
```

File

System

Description

This is type System.PInitContext.

38.5.33 System.PInt64 Type

PInt64 is a pointer to an Int64 variable.

Pascal

```
PInt64 = ^Int64;
```

C++

```
^Int64 PInt64;
```

File

System

Description

PInt64 is a pointer to an Int64 variable.

See Also

Int64

38.5.34 System.PInteger Type

PInteger is a pointer to an Integer variable.

Pascal

```
PInteger = ^Integer;
```

C++

```
^Integer PInteger;
```

File

System

Description

PInteger is a pointer to an Integer variable.

See Also

Integer

38.5.35 System.PIntegerArray Type

PIntegerArray is a pointer to an IntegerArray (see page 3828) variable.

Pascal

```
PIntegerArray = ^IntegerArray;
```

C++

```
^IntegerArray PIntegerArray;
```

File

System

Description

PIntegerArray is a pointer to an IntegerArray (see page 3828) variable.

See Also

IntegerArray (see page 3828)

38.5.36 System.PInterfaceEntry Type

PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3803) variable.

Pascal

```
PInterfaceEntry = ^TInterfaceEntry;
```

C++

```
^TInterfaceEntry PInterfaceEntry;
```

File

System

Description

PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3803) variable.

See Also

TInterfaceEntry (see page 3803)

38.5.37 System.PInterfaceTable Type

PInterfaceTable is a pointer to a TInterfaceTable (see page 3804) variable.

Pascal

```
PInterfaceTable = ^TInterfaceTable;
```

C++

```
^TInterfaceTable PInterfaceTable;
```

File

System

Description

PInterfaceTable is a pointer to a TInterfaceTable (see page 3804) variable.

See Also

TInterfaceTable (see page 3804)

38.5.38 System.PLibModule Type

PLibModule is a pointer to a TLibModule (see page 3804) variable.

Pascal

```
PLibModule = ^TLibModule;
```

C++

```
^TLibModule PLibModule;
```

File

System

Description

PLibModule is a pointer to a TLibModule (see page 3804) variable.

See Also

TLibModule (see page 3804)

38.5.39 System.PLongint Type

Pascal

```
PLongint = ^Longint;
```

C++

```
^Longint PLongint;
```

File

System

Description

This is type System.PLongint.

38.5.40 System.PLongWord Type

PLongWord is a pointer to a LongWord variable.

Pascal

```
PLongWord = ^LongWord;
```

C++

```
^LongWord PLongWord;
```

File

System

Description

PLongWord is a pointer to a LongWord variable.

See Also

LongWord

38.5.41 System::PMemoryManager Type

Pascal

```
PMemoryManager = ^TMemoryManager;
```

C++

```
^TMemoryManager PMemoryManager;
```

File

System.pas

Description

This is type System::PMemoryManager.

38.5.42 System::PMemoryManagerEx Type

Pascal

```
PMemoryManagerEx = ^TMemoryManagerEx;
```

C++

```
^TMemoryManagerEx PMemoryManagerEx;
```

File

System.pas

Description

This is type System::PMemoryManagerEx.

38.5.43 System::PModuleUnloadRec Type

Pascal

```
PModuleUnloadRec = ^TModuleUnloadRec;
```

C++

```
^TModuleUnloadRec PModuleUnloadRec;
```

File

System.pas

Description

This is type System::PModuleUnloadRec.

38.5.44 System.PMonitor Type

Pascal

```
PMonitor = ^TMonitor;
```

C++

```
^TMonitor PMonitor;
```

File

System

Description

This is type System.PMonitor.

38.5.45 System.PMonitorSupport Type

Pascal

```
PMonitorSupport = ^TMonitorSupport;
```

C++

```
^TMonitorSupport PMonitorSupport;
```

File

System

Description

This is type System.PMonitorSupport.

38.5.46 System.PointerArray Type

Pascal

```
PointerArray = array [0..512*1024*1024 - 2] of Pointer;
```

C++

```
array [0..512*1024*1024 - 2] of Pointer PointerArray;
```

File

System

Description

This is type System.PointerArray.

38.5.47 System.POleVariant Type

Pascal

```
POleVariant = ^OleVariant;
```

C++

```
^OleVariant POleVariant;
```

File

System

Description

This is type System.POleVariant.

38.5.48 System.PPAnsiChar Type

Pascal

```
PPAnsiChar = ^PAnsiChar;
```

C++

```
^PAnsiChar PPAnsiChar;
```

File

System

Description

This is type System.PPAnsiChar.

38.5.49 System.PPCharArray Type

Pascal

```
PPCharArray = ^TPCharArray;
```

C++

```
^TPCharArray PPCharArray;
```

File

System

Description

This is type System.PPCharArray.

38.5.50 System.PPDispatch Type

Pascal

```
PPDispatch = ^PDispatch;
```

C++

```
^PDispatch PPDispatch;
```

File

System

Description

This is type System.PPDispatch.

38.5.51 System.PPMonitor Type

Pascal

```
PPMonitor = ^PMonitor;
```

C++

```
^PMonitor PPMonitor;
```

File

System

Description

This is type System.PPMonitor.

38.5.52 System.PPointer Type

PPointer is a pointer to a Pointer variable.

Pascal

```
PPointer = ^Pointer;
```

C++

```
^Pointer PPointer;
```

File

System

Description

PPointer is a pointer to a Pointer variable.

See Also

Pointer

38.5.53 System.PPointerArray Type

PPointerArray is a pointer to a PointerArray (see page 3842) variable.

Pascal

```
PPointerArray = ^PointerArray;
```

C++

```
^PointerArray PPointerArray;
```

File

System

Description

PPointerArray is a pointer to a PointerArray (see page 3842) variable.

See Also

PointerArray (see page 3842)

38.5.54 System.PPUnknown Type

Pascal

```
PPUnknown = ^PUnknown;
```

C++

```
^PUnknown PPUnknown;
```

File

System

Description

This is type System.PPUnknown.

38.5.55 System.PPWideChar Type

Pascal

```
PPWideChar = ^PWideChar;
```

C++

```
^PWideChar PPWideChar;
```

File

System

Description

This is type System.PPWideChar.

38.5.56 System.PRawByteString Type

Pascal

```
PRawByteString = ^RawByteString;
```

C++

```
^RawByteString PRawByteString;
```

File

System

Description

This is type System.PRawByteString.

38.5.57 System::PResStringRec Type

Pascal

```
PResStringRec = ^TResStringRec;
```

C++

```
^TResStringRec PResStringRec;
```

File

System.pas

Description

This is type System::PResStringRec.

38.5.58 System.PShortInt Type

PShortInt is a pointer to a ShortInt variable.

Pascal

```
PShortInt = ^ShortInt;
```

C++

```
^ShortInt PShortInt;
```

File

System

Description

PShortInt is a pointer to a ShortInt variable.

See Also

ShortInt

38.5.59 System.PShortString Type

PShortString is a pointer to a ShortString variable.

Pascal

```
PShortString = ^ShortString;
```

C++

```
^ShortString PShortString;
```

File

System

Description

PShortString is a pointer to a ShortString variable.

See Also

ShortString

38.5.60 System.PSingle Type

PSingle is a pointer to a Single variable.

Pascal

```
PSingle = ^Single;
```

C++

```
^Single PSingle;
```

File

System

Description

PSingle is a pointer to a Single variable.

See Also

Single

38.5.61 System::PSmallBlockTypeState Type

Pascal

```
PSmallBlockTypeState = TSmallBlockTypeState *;
```

C++

```
typedef TSmallBlockTypeState * PSmallBlockTypeState;
```

File

sysclass.h

Description

This is type System::PSmallBlockTypeState.

38.5.62 System.PSmallInt Type

PSmallInt is a pointer to a SmallInt variable.

Pascal

```
PSmallInt = ^SmallInt;
```

C++

```
^SmallInt PSmallInt;
```

File

System

Description

PSmallInt is a pointer to a SmallInt variable.

See Also

SmallInt

38.5.63 System.PTextBuf Type

Pascal

```
PTextBuf = ^TTextBuf;
```

C++

```
^TTextBuf PTextBuf;
```

File

System

Description

This is type System.PTextBuf.

38.5.64 System.PUCS4Char Type

Pascal

```
PUCS4Char = ^UCS4Char;
```

C++

```
^UCS4Char PUCS4Char;
```

File

System

Description

This is type System.PUCS4Char.

38.5.65 System.PUCS4CharArray Type

Pascal

```
PUCS4CharArray = ^TUCS4CharArray;
```

C++

```
^TUCS4CharArray PUCS4CharArray;
```

File

System

Description

This is type System.PUCS4CharArray.

38.5.66 System.PUInt64 Type

Pascal

```
PUInt64 = ^UInt64;
```

C++

```
^UInt64 PUInt64;
```

File

System

Description

This is type System.PUInt64.

38.5.67 System.PUnicodeString Type

Pascal

```
PUnicodeString = ^UnicodeString;
```

C++

```
^UnicodeString PUnicodeString;
```

File

System

Description

This is type System.PUnicodeString.

38.5.68 System.PUnitEntryTable Type

Pascal

```
PUnitEntryTable = ^UnitEntryTable;
```

C++

```
^UnitEntryTable PUnitEntryTable;
```

File

System

Description

This is type System.PUnitEntryTable.

38.5.69 System.PUnknown Type

PUnknown is a pointer to an IUnknown (see page 3828) variable.

Pascal

```
PUnknown = ^IUnknown;
```

C++

```
^IUnknown PUnknown;
```

File

System

Description

PUnknown is a pointer to an IUnknown (see page 3828) variable.

See Also

IUnknown (see page 3828)

38.5.70 System.PUTF8String Type

Pascal

```
PUTF8String = ^UTF8String;
```

C++

```
^UTF8String PUTF8String;
```

File

System

Description

This is type System.PUTF8String.

38.5.71 System.PVarArray Type

Pascal

```
PVarArray = ^TVarArray;
```

C++

```
^TVarArray PVarArray;
```

File

System

Description

This is type System.PVarArray.

38.5.72 System.PVarArrayBoundArray Type

Pascal

```
PVarArrayBoundArray = ^TVarArrayBoundArray;
```

C++

```
^TVarArrayBoundArray PVarArrayBoundArray;
```

File

System

Description

This is type System.PVarArrayBoundArray.

38.5.73 System.PVarArrayCoorArray Type

Pascal

```
PVarArrayCoorArray = ^TVarArrayCoorArray;
```

C++

```
^TVarArrayCoorArray PVarArrayCoorArray;
```

File

System

Description

This is type System.PVarArrayCoorArray.

38.5.74 System.PVarData Type

Pascal

```
PVarData = ^TVarData;
```

C++

```
^TVarData PVarData;
```

File

System

Description

This is type System.PVarData.

38.5.75 System.PVariant Type

PVariant is a pointer to a Variant (see page 3430) variable.

Pascal

```
PVariant = ^Variant;
```

C++

```
^Variant PVariant;
```

File

System

Description

PVariant is a pointer to a Variant (see page 3430) variable.

See Also

Variant (see page 3430)

38.5.76 System::PVariantManager Type

Pascal

```
PVariantManager = ^TVariantManager;
```

C++

```
^TVariantManager PVariantManager;
```

File

System.pas

Description

This is type System::PVariantManager.

38.5.77 System::PVarRec Type

Pascal

```
PVarRec = ^TVarRec;
```

C++

```
^TVarRec PVarRec;
```

File

System.pas

Description

This is type System::PVarRec.

38.5.78 System.PWideString Type

PWideString is a pointer to a WideString (see page 3476) variable.

Pascal

```
PWideString = ^WideString;
```

C++

```
^WideString PWideString;
```

File

System

Description

PWideString is a pointer to a WideString (see page 3476) variable.

See Also

WideString (see page 3476)

38.5.79 System.PWord Type

PWord is a pointer to a Word variable.

Pascal

```
PWord = ^Word;
```

C++

```
^Word PWord;
```

File

System

Description

PWord is a pointer to a Word variable.

See Also

Word

38.5.80 System.PWordBool Type

PWordBool is a pointer to a WordBool variable.

Pascal

```
PWordBool = ^WordBool;
```

C++

```
^WordBool PWordBool;
```

File

System

Description

PWordBool is a pointer to a WordBool variable.

See Also

WordBool

38.5.81 System.RawByteString Type

RawByteString enables the passing of string data of any code page without doing any codepage conversions.

Pascal

```
RawByteString = type AnsiString($ffff);
```

C++

```
type AnsiString($ffff) RawByteString;
```

File

System

Description

RawByteString enables the passing of string data of any code page without doing any codepage conversions. Normally, this means that parameters of routines that process strings without regard for the string's code page should be of type RawByteString. Declaring variables of type RawByteString should rarely, if ever, be done, because this can lead to undefined behavior and potential data loss.

38.5.82 System::ResourceString Type

Pascal

```
ResourceString = TResStringRec;
```

C++

```
typedef TResStringRec ResourceString;
```

File

sysclass.h

Description

This is type System::ResourceString.

38.5.83 System.TAssertErrorProc Type

Pascal

```
TAssertErrorProc = procedure (const Message, Filename: string; LineNumber: Integer; ErrorAddr:
Pointer);
```

C++

```
(const Message, Filename: string; LineNumber: Integer; ErrorAddr: Pointer) (
TAssertErrorProc)();
```

File

System

Description

This is type System.TAssertErrorProc.

38.5.84 System::TBooleanDynArray Type

Pascal

```
TBooleanDynArray = DynamicArray<Boolean>;
```

C++

```
typedef DynamicArray<Boolean> TBooleanDynArray;
```

File

sysdyn.h

Description

This is type System::TBooleanDynArray.

38.5.85 System.TBoundArray Type

Pascal

```
TBoundArray = array of Integer;
```

C++

```
array of Integer TBoundArray;
```

File

System

Description

This is type System.TBoundArray.

38.5.86 System::TByteDynArray Type

Pascal

```
TByteDynArray = DynamicArray<Byte>;
```

C++

```
typedef DynamicArray<Byte> TByteDynArray;
```

File

sysdyn.h

Description

This is type System::TByteDynArray.

38.5.87 System::TCardinalDynArray Type

Pascal

```
TCardinalDynArray = DynamicArray<Cardinal>;
```

C++

```
typedef DynamicArray<Cardinal> TCardinalDynArray;
```

File

sysdyn.h

Description

This is type System::TCardinalDynArray.

38.5.88 System.TClass Type

TClass defines the metaclass for System::TObject (see page 3364).

Pascal

```
TClass = class of TObject;
```

C++

```
TObject * TClass;
```

File

System

Description

TClass is the metaclass for System::TObject (see page 3364). Its value is the class reference for System::TObject (see page 3364) or for one of its descendants.

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject (see page 3364)

38.5.89 System.TClassHelperBaseClass Type

Pascal

```
TClassHelperBaseClass = class of TClassHelperBase;
```

C++

```
TClassHelperBase TClassHelperBaseClass;
```

File

System

Description

This is type System.TClassHelperBaseClass.

38.5.90 System.TDate Type

Pascal

```
TDate = type TDateTime;
```

C++

```
type TDateTime TDate;
```

File

System

Description

This is type System.TDate.

38.5.91 System.TDateTime Type

TDateTime represents a date-and-time value in the Delphi language.

Pascal

```
TDateTime = type Double;
```

C++

```
type Double TDateTime;
```

File

System

Description

In Delphi, TDateTime is a type that maps to a Double. In C++, the TDateTime class corresponds to the Delphi TDateTime type.

The integral part of a Delphi TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of the TDateTime value is fraction of a 24 hour day that has elapsed.

Following are some examples of TDateTime values and their corresponding dates and times:

0	12/30/1899 12:00 am
2.75	1/1/1900 6:00 pm
-1.25	12/29/1899 6:00 am
35065	1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the TDateTime value is positive.

When working with negative TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the TDateTime value. For example, 6:00 am on 12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no TDateTime values between –1 and 0.

Note: Delphi 1.0 calculated the date from year 1 instead of from 1899. To convert a Delphi 1.0 date to a TDateTime value in later versions of the Delphi language, subtract 693594.0 from the Delphi 1.0 date.

See Also

TTimeStamp

TDate (see page 3857)

TSQLTimeStamp

38.5.92 System.TDLLProc Type

Pascal

```
TDLLProc = procedure (Reason: Integer);
```

C++

```
(Reason: Integer) ( TDLLProc)();
```

File

System

Description

This is type System.TDLLProc.

38.5.93 System.TDLLProcEx Type

Pascal

```
TDLLProcEx = procedure (Reason: Integer; Reserved: Integer);
```

C++

```
(Reason: Integer; Reserved: Integer) ( TDLLProcEx)();
```

File

System

Description

This is type System.TDLLProcEx.

38.5.94 System::TDoubleDynArray Type

Pascal

```
TDoubleDynArray = DynamicArray<Double>;
```

C++

```
typedef DynamicArray<Double> TDoubleDynArray;
```

File

sysdyn.h

Description

This is type System::TDoubleDynArray.

38.5.95 System.TEnumModuleFunc Type

Pascal

```
TEnumModuleFunc = function (HInstance: Integer; Data: Pointer): Boolean;
```

C++

```
(HInstance: Integer; Data: Pointer): Boolean ( TEnumModuleFunc)();
```

File

System

Description

This is type System.TEnumModuleFunc.

38.5.96 System.TEnumModuleFuncLW Type

Pascal

```
TEnumModuleFuncLW = function (HInstance: LongWord; Data: Pointer): Boolean;
```

C++

```
(HInstance: LongWord; Data: Pointer): Boolean ( TEnumModuleFuncLW)();
```

File

System

Description

This is type System.TEnumModuleFuncLW.

38.5.97 System.TFileIOFunc Type

Pascal

```
TFileIOFunc = function (var F: TFileRec): Integer;
```

C++

```
(var F: TFileRec): Integer ( TFileIOFunc)();
```

File

System

Description

This is type System.TFileIOFunc.

38.5.98 System::TInt64DynArray Type

Pascal

```
TInt64DynArray = DynamicArray<__int64>;
```

C++

```
typedef DynamicArray<__int64> TInt64DynArray;
```

File

sysdyn.h

Description

This is type System::TInt64DynArray.

38.5.99 System::TIntegerDynArray Type

Pascal

```
TIntegerDynArray = DynamicArray<Integer>;
```

C++

```
typedef DynamicArray<Integer> TIntegerDynArray;
```

File

sysdyn.h

Description

This is type System::TIntegerDynArray.

38.5.100 System.TInterfacedClass Type

TInterfacedClass defines the metaclass for TInterfacedObject (see page 3355).

Pascal

```
TInterfacedClass = class of TInterfacedObject;
```

C++

```
TInterfacedObject TInterfacedClass;
```

File

System

Description

TInterfacedClass is the metaclass for TInterfacedObject (see page 3355). Its value is the class reference for TInterfacedObject (see page 3355) or for one of its descendants.

See Also

TInterfacedObject (see page 3355)

38.5.101 System::TLongWordDynArray Type

Pascal

```
TLongWordDynArray = DynamicArray<unsigned>;
```

C++

```
typedef DynamicArray<unsigned> TLongWordDynArray;
```

File

sysdyn.h

Description

This is type System::TLongWordDynArray.

38.5.102 System.TMemoryMap Type

TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3796) values.

Pascal

```
TMemoryMap = array[0..65535] of TChunkStatus;
```

C++

```
array[0..65535] of TChunkStatus TMemoryMap;
```

File

System

Description

GetMemoryMap (see page 3691) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap obtains a status value for each of them in the TMemoryMap array. Each TMemoryMap array element may have the following value :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 3691)

38.5.103 System.TModuleUnloadProc Type

TModuleUnloadProc dehines a procedure that can be called during package unload.

Pascal

```
TModuleUnloadProc = procedure (HInstance: Integer);
```

C++

```
(HInstance: Integer) ( TModuleUnloadProc)();
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your TModuleUnloadProcLW (see page 3863) routine. In essence, AddModuleUnloadProc (see page 3620) adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Proc is the Handle to the DLL Package instance being unloaded.

The LW part of the name refers to the Long Word type of package handle. This is in line with Windows handles, and the THandle type.

See Also

AddModuleUnloadProcLW

TModuleUnloadProcLW (see page 3863)

38.5.104 System.TModuleUnloadProcLW Type

Pascal

```
TModuleUnloadProcLW = procedure (HInstance: LongWord);
```

C++

```
(HInstance: LongWord) ( TModuleUnloadProcLW)();
```

File

System

Description

This is type System.TModuleUnloadProcLW.

38.5.105 System.TPCharArray Type

Pascal

```
TPCharArray = array[0..(MaxLongint div SizeOf(PChar))-1] of PChar;
```

C++

```
array[0..(MaxLongint div SizeOf(PChar))-1] of PChar TPCharArray;
```

File

System

Description

This is type System.TPCharArray.

38.5.106 System.TResourceHandle Type

Pascal

```
TResourceHandle = HRSRC;
```

C++

```
HRSRC TResourceHandle;
```

File

System

Description

This is type System.TResourceHandle.

38.5.107 System.TSafeCallErrorProc Type

Pascal

```
TSafeCallErrorProc = procedure (ErrorCode: HResult; ErrorAddr: Pointer);
```

C++

```
(ErrorCode: HResult; ErrorAddr: Pointer) ( TSafeCallErrorProc)();
```

File

System

Description

This is type System.TSafeCallErrorProc.

38.5.108 System::TShortIntDynArray Type

Pascal

```
TShortIntDynArray = DynamicArray<Shortint>;
```

C++

```
typedef DynamicArray<Shortint> TShortIntDynArray;
```

File

sysdyn.h

Description

This is type System::TShortIntDynArray.

38.5.109 System::TSingleDynArray Type

Pascal

```
TSingleDynArray = DynamicArray<Single>;
```

C++

```
typedef DynamicArray<Single> TSingleDynArray;
```

File

sysdyn.h

Description

This is type System::TSingleDynArray.

38.5.110 System::TSmallBlockTypeStates Type

Pascal

```
TSmallBlockTypeStates = array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState;
```

C++

```
array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState
TSmallBlockTypeStates[NumSmallBlockTypes];
```

File

System.pas

Description

This is type System::TSmallBlockTypeStates.

38.5.111 System::TSmallIntDynArray Type

Pascal

```
TSmallIntDynArray = DynamicArray<Smallint>;
```

C++

```
typedef DynamicArray<Smallint> TSmallIntDynArray;
```

File

sysdyn.h

Description

This is type System::TSmallIntDynArray.

38.5.112 System::TStringDynArray Type

Pascal

```
TStringDynArray = DynamicArray<String>;
```

C++

```
typedef DynamicArray<String> TStringDynArray;
```

File

sysdyn.h

Description

This is type System::TStringDynArray.

38.5.113 System.TSystemThreadEndProc Type

Pascal

```
TSystemThreadEndProc = procedure (ExitCode: Integer);
```

C++

```
(ExitCode: Integer) ( TSystemThreadEndProc)();
```

File

System

Description

This is type System.TSystemThreadEndProc.

38.5.114 System.TSystemThreadFuncProc Type

Pascal

```
TSystemThreadFuncProc = function (ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer;
```

C++

```
(ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer ( TSystemThreadFuncProc)();
```

File

System

Description

This is type System.TSystemThreadFuncProc.

38.5.115 System.TTextBuf Type

Pascal

```
TTextBuf = array[0..127] of AnsiChar;
```

C++

```
array[0..127] of AnsiChar TTextBuf;
```

File

System

Description

This is type System.TTextBuf.

38.5.116 System.TTextIOFunc Type

Pascal

```
TTextIOFunc = function (var F: TTextRec): Integer;
```

C++

```
(var F: TTextRec): Integer ( TTextIOFunc)();
```

File

System

Description

This is type System.TTextIOFunc.

38.5.117 System::TThreadFunc Type

Pascal

```
TThreadFunc = function (Parameter: Pointer): Integer;
```

C++

```
(Parameter: Pointer): Integer (* TThreadFunc)(Pointer Parameter);
```

File

System.pas

Description

This is type System::TThreadFunc.

38.5.118 System.TTime Type

Pascal

```
TTime = type TDateTime;
```

C++

```
type TDateTime TTime;
```

File

System

Description

This is type System.TTime.

38.5.119 System.TUCS4CharArray Type

Pascal

```
TUCS4CharArray = array [0..$effffff] of UCS4Char;
```

C++

```
array [0..$effffff] of UCS4Char TUCS4CharArray;
```

File

System

Description

This is type System.TUCS4CharArray.

38.5.120 System.TVarArrayBoundArray Type

Pascal

```
TVarArrayBoundArray = array [0..0] of TVarArrayBound;
```

C++

```
array [0..0] of TVarArrayBound TVarArrayBoundArray;
```

File

System

Description

This is type System.TVarArrayBoundArray.

38.5.121 System.TVarArrayCoorArray Type

Pascal

```
TVarArrayCoorArray = array [0..0] of Integer;
```

C++

```
array [0..0] of Integer TVarArrayCoorArray;
```

File

System

Description

This is type System.TVarArrayCoorArray.

38.5.122 System.TVarOp Type

TVarOp defines a Variant (see page 3430) operation.

Pascal

```
TVarOp = Integer;
```

C++

```
int TVarOp;
```

File

System

Description

TVarOp is used to define an Variant (see page 3430) operation. There are three operation categories :

Type	Meaning
Two operand processing	A logical operation of left and right operands. Such as BinaryOp.
Two operand comparison	A comparison of left and right operands. Such as CompareOp.
One operand processing	Processing of a single operand. Such as UnaryOp.

TVarOp may hold one of the following values :

Value	Meaning
opAdd (see page 3917)	Addition.
opAnd (see page 3917)	Bitwise and.
opCmpEQ (see page 3918)	Compare for equality.
opCmpGE (see page 3918)	Compare left greater than or equal to right.
opCmpGT (see page 3919)	Compare left greater than right.
opCmpLE (see page 3919)	Compare left less than or equal to right.
opCmpLT (see page 3920)	Compare left less than right.

opCmpNE (see page 3920)	Compare for inequality.
opCompare (see page 3921)	Generalised comparison.
opDivide (see page 3921)	Division of left by right.
opIntDivide (see page 3922)	Integer division of left by right.
opModulus (see page 3922)	Remainder of left divide by right.
opMultiply (see page 3923)	Left multiplied by right.
opNegate (see page 3923)	Negation of a single operand value.
opNot (see page 3924)	Bitwise negation of a single operand.
opOr (see page 3924)	Bitwise or.
opShiftLeft (see page 3925)	Bitwise shift left.
opShiftRight (see page 3925)	Bitwise shift right.
opSubtract (see page 3926)	Right subtracted from left.
opXor (see page 3926)	Bitwise exclusive or.

See Also

BinaryOp

CompareOp (see page 5021)

UnaryOp

38.5.123 System::TWideStringDynArray Type

Pascal

```
TWideStringDynArray = DynamicArray<WideString>;
```

C++

```
typedef DynamicArray<WideString> TWideStringDynArray;
```

File

sysdyn.h

Description

This is type System::TWideStringDynArray.

38.5.124 System::TWordDynArray Type

Pascal

```
TWordDynArray = DynamicArray<Word>;
```

C++

```
typedef DynamicArray<Word> TWordDynArray;
```

File

sysdyn.h

Description

This is type System::TWordDynArray.

38.5.125 System.UCS4Char Type

UCS4Char is used for working with 4–byte Unicode characters.

Pascal

```
UCS4Char = type LongWord;
```

C++

```
type LongWord UCS4Char;
```

File

System

Description

UCS4Char represents a 4–byte Unicode character. UCS4String (see page 3871) is a dynamic array of UCS4Char.

See Also

UCS4String (see page 3871)

UCS4StringToWideString

38.5.126 System.UCS4String Type

UCS4String is used for working with 4–byte Unicode characters.

Pascal

```
UCS4String = array of UCS4Char;
```

C++

```
array of UCS4Char UCS4String;
```

File

System

Description

UCS4Char (see page 3871) represents a 4–byte Unicode character. UCS4String, which is a dynamic array of UCS4Char (see page 3871), represents a string encoded using UCS4 (4–byte Unicode).

See Also

UCS4Char (see page 3871)

UCS4StringToWideString

38.5.127 System.UInt16 Type

Pascal

```
UInt16 = Word;
```

C++

```
Word UInt16;
```

File

System

Description

This is type System.UInt16.

38.5.128 System.UInt32 Type

Pascal

```
UInt32 = Cardinal;
```

C++

```
unsigned UInt32;
```

File

System

Description

This is type System.UInt32.

38.5.129 System.UInt8 Type

Pascal

```
UInt8 = Byte;
```

C++

```
Byte UInt8;
```

File

System

Description

This is type System.UInt8.

38.5.130 System.UnitEntryTable Type

Pascal

```
UnitEntryTable = array [0..9999999] of PackageUnitEntry;
```

C++

```
array [0..9999999] of PackageUnitEntry UnitEntryTable;
```

File

System

Description

This is type System.UnitEntryTable.

38.5.131 System.UTF8String Type

UTF8String represents UTF-8 encoded Unicode strings.

Pascal

```
UTF8String = type AnsiString(65001);
```

C++

```
type AnsiString(65001) UTF8String;
```

File

System

Description

UTF8String represents a string encoded using UTF-8 (variable number of bytes Unicode). It is a System::AnsiStringBase (see page 3185) type with a UTF-8 code page.

In Delphi, UTF8String is a true compiler type. The compiler does conversions between System::UnicodeString (see page 3393) and UTF8String as necessary.

See Also

System::AnsiStringBase (see page 3185)

System::UnicodeString (see page 3393)

38.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AbstractErrorProc (see page 3875)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 3875)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 3877)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 3878)	Points to the assertion error-handler.
CmdLine (see page 3880)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 3880)	Specifies the flags for displaying the application's main window.
CPUCount (see page 3881)	This is variable System.CPUCount.
DebugHook (see page 3881)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 3881)	Specifies the Default 8087 control word.
DefaultSystemCodePage (see page 3883)	This is variable System.DefaultSystemCodePage.
DefaultTextLineBreakStyle (see page 3883)	Specifies the characters that are used by default to separate lines in text.
DefaultUnicodeCodePage (see page 3884)	This is variable System.DefaultUnicodeCodePage.
DispCallByIDProc (see page 3884)	This is variable System.DispCallByIDProc.
DLLProc (see page 3884)	Points to a procedure invoked by a DLL entry point.
envp (see page 3885)	This is variable System.envp.
ErrorAddr (see page 3885)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 3886)	Points to the RTL runtime error handler.
ErrOutput (see page 3888)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 3888)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionAcquired (see page 3889)	This is variable System.ExceptionAcquired.
ExceptionClass (see page 3889)	This is variable System.ExceptionClass.
ExceptObjProc (see page 3889)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 3890)	Points to the lowest-level exception handler.
ExitCode (see page 3891)	Contains the application's exit code.
ExitProc (see page 3891)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 3892)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 3892)	Indicates the access mode on typed and untyped files opened by the Reset (see page 3734) procedure.
HeapAllocFlags (see page 3893)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 3894)	This is variable System.HPrevInst.
InitProc (see page 3894)	InitProc is the last installed initialization procedure.
Input (see page 3895)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 3896)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 3896)	Indicates whether the module is a shared library.
IsMultiThread (see page 3897)	Indicates whether the application spawned additional threads using BeginThread (see page 3632) or TThread objects.
JITEnable (see page 3897)	Controls when the just-in-time debugger is called.
LibModuleList (see page 3898)	Lists module records for enumeration procedures.
MainInstance (see page 3898)	Indicates the instance handle for the main executable.
MainThreadID (see page 3899)	Indicates the instance handle for the current module's main thread of execution.
ModuleUnloadList (see page 3899)	This is variable System.ModuleUnloadList.
MonitorSupport (see page 3900)	This is variable System.MonitorSupport.
NeverSleepOnMMThreadContention (see page 3900)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 3901)	Controls whether the application displays an error message when a runtime error occurs.

Output (see page 3901)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 3902)	Points to RaiseException Windows function.
RaiseExceptObjProc (see page 3902)	This is variable System.RaiseExceptObjProc.
RandSeed (see page 3902)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 3903)	Report memory leaks on shutdown.
RTLUnwindProc (see page 3903)	This is variable System.RTLUnwindProc.
SafeCallErrorProc (see page 3904)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 3904)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 3905)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 3905)	This is variable System.Test8086.
Test8087 (see page 3905)	This is variable System.Test8087.
TestFDIV (see page 3906)	This is variable System.TestFDIV.
UTF8CompareLocale (see page 3906)	This is variable System.UTF8CompareLocale.
VarAddRefProc (see page 3906)	This is variable System.VarAddRefProc.
VarClearProc (see page 3907)	This is variable System.VarClearProc.
VarCopyProc (see page 3907)	This is variable System.VarCopyProc.
VarToLStrProc (see page 3907)	This is variable System.VarToLStrProc.
VarToWStrProc (see page 3908)	This is variable System.VarToWStrProc.

38.6.1 System.AbstractErrorProc Variable

AbstractErrorProc defines a callback routine for abstract method call errors.

Pascal

```
AbstractErrorProc: procedure;
```

C++

```
procedure AbstractErrorProc;
```

File

System

Description

An abstract method is a virtual or dynamic method that has no implementation in the class where it is declared. Its implementation is deferred to a descendant class. An EAbstractError exception will be thrown if a call is attempted to an abstact method.

You can define an AbstractErrorProc routine that will be called prior to the exception throw.

See Also

EAbstractError (see page 3982)

38.6.2 System.AllocMemCount Variable

Represents the total number of allocated memory blocks in a Delphi application.

Pascal

```
AllocMemCount: Integer deprecated;
```

C++

```
Integer deprecated AllocMemCount;
```

File

System

Description

Note: On Windows, AllocMemCount is the number of currently allocated blocks of memory that were requested by the user. AllocMemCount is incremented each time a block of memory is allocated and is decremented each time a block of memory is freed. Use AllocMemCount to find out how many unfreed blocks of memory remain.

Note: On Linux, the AllocMemCount variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemCount is a global variable declared in the System (see page 3165) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemCount. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemCount
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked libraries can safely access AllocMemCount directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and also uses libraries that are statically linked to the RTL, then the AllocMemCount variable should never be accessed directly, since the EXE and each library will have its own instance of it. Instead, use the GetAllocMemCount function, implemented in BorlandMM, which returns the instance of the AllocMemCount global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem (see page 359) unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemCount in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLs	It is not advisable to create applications that mix runtime packages with statically linked libraries. In this circumstance, use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemCount that references memory that is allocated and freed only by that particular module.

See Also

AllocMemSize (see page 3877)

AllocMem (see page 3622)

38.6.3 System.AllocMemSize Variable

Represents the total size of allocated memory blocks.

Pascal

```
AllocMemSize: Integer deprecated;
```

C++

```
Integer deprecated AllocMemSize;
```

File

System

Description

Note: On Windows, AllocMemSize is the total size, in bytes, of all currently allocated blocks of memory in use by an application. Use AllocMemSize to find out how many bytes of memory an application is currently using.

Note: On Linux, the AllocMemSize variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemSize is a global variable declared in the System (see page 3165) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemSize. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemSize
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked DLLs can safely access AllocMemSize directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and if that application is also using DLLs that are statically linked to the RTL then the AllocMemSize variable should never be accessed directly, since the EXE and each DLL will have its own instance of it. Instead use the GetAllocMemSize function, implemented in BorlandMM, which returns the instance of the AllocMemSize global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem (see page 359) unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemSize in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLS	It is not advisable to create applications that mix runtime packages with statically linked DLLs. In this circumstance use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemSize that references memory that is allocated and freed only by that particular module.

See Also

AllocMem (see page 3622)

AllocMemCount (see page 3875)

38.6.4 System.AssertErrorProc Variable

Points to the assertion error-handler.

Pascal

```
AssertErrorProc: TAssertErrorProc;
```

C++

```
TAssertErrorProc AssertErrorProc;
```

File

System

Description

AssertErrorProc points to a routine that produces runtime error 227 in Delphi code. The SysUtils (see page 3967) unit replaces it to cause an EAssertionFailed exception. If you do not use assertions, you can replace this with a method that that does something else, such as call OutputDebugString to add messages to the event log (see the example).

The value of AssertErrorProc must be a procedure with the following signature

procedure AssertErrorHandler(const Message, Filename: string; LineNumber: Integer; ErrorAddr (see page 3885): Pointer);

Delphi Examples:

```
{
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project. Build the executable in the IDE
by hitting "Run".  Then go to the debug directory for the project and
execute the command line "AssertErrorProc_proj /Debug".
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;
```

```
ModifyStorage(nil, 'Ooops');
```

See Also

EAssertionFailed

Assert (see page 3627)

ErrorProc (see page 3886)

38.6.5 System.CmdLine Variable

Points to the command-line arguments specified when the application is invoked.

Pascal

```
CmdLine: PChar platform;
```

C++

```
PChar platform CmdLine;
```

File

System

Description

Note: On Windows, the CmdLine variable points to a null-terminated string that contains the command-line arguments specified when the application was started. Use ParamStr (see page 3718) to access individual arguments. In a library (DLL), CmdLine is nil (Delphi) or NULL (C++).

Note: On Linux, CmdLine is always nil (Delphi) or NULL (C++).

See Also

ParamStr (see page 3718)

38.6.6 System.CmdShow Variable

Specifies the flags for displaying the application's main window.

Pascal

```
CmdShow: Integer platform;
```

C++

```
Integer platform CmdShow;
```

File

System

Description

Note: On Windows, CmdShow specifies the parameter that should be passed to the Windows API ShowWindow when displaying the main window of the application. In a library (DLL), CmdShow is always 0.

Note: On Linux, CmdShow is always 0. It exists only for compatibility with Windows code in cross-platform applications.

38.6.7 System.CPUCount Variable

Pascal

```
CPUCount: Integer;
```

C++

```
int CPUCount;
```

File

System

Description

This is variable System.CPUCount.

38.6.8 System.DebugHook Variable

DebugHook is set if an application is running under the IDE debugger.

Pascal

```
DebugHook: Byte platform = 0;
```

C++

```
Byte platform DebugHook = 0;
```

File

System

Description

DebugHook is 0 when an application operates outside the IDE debugger. DebugHook is 1 when an application operates inside the IDE debugger.

38.6.9 System.Default8087CW Variable

Specifies the Default 8087 control word.

Pascal

```
Default8087CW: Word = $1332;
```

C++

```
Word Default8087CW = 0x1332;
```

File

System

Description

Default8087CW is the value to which the Floating Point Unit (FPU) control register is set. The FPU control word includes bits that control the FPU's precision, rounding mode, and whether exceptions generate signals if they occur. See Intel's processor documentation for details.

The Set8087CW (see page 3744) procedure allows the programmer to have direct access to the Default8087CW. Setting the floating-point precision and rounding mechanism can be useful if you are reusing old code that is sensitive to the floating-point precision standard used. However, be aware that using this procedure to change the value of the 8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

C++ Examples:

```
/*
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
*/
Word Saved8087CW;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit3->Text = FloatToStr(StrToFloat(Edit1->Text) / StrToFloat(Edit2->Text));
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "FPU Exceptions")
    System::Set8087CW(Saved8087CW);
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "No FPU Exceptions")
    System::Set8087CW(0x133f); // Disable all fpu exceptions
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("No FPU Exceptions");
  RadioGroup1->Items->Add("FPU Exceptions");
  RadioGroup1->ItemIndex = 2;
  Saved8087CW = Default8087CW;  // Save this because Set8087CW changes it!
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  System::Set8087CW(Saved8087CW); // Default value (with exceptions) is 0x1372
}
```

Delphi Examples:

```
{
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
}
var
  Form1: TForm1;
  Saved8087CW: Word;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit3.Text := FloatToStr(StrToFloat(Edit1.Text) / StrToFloat(Edit2.Text));
end;
```

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'FPU Exceptions' then
    System.Set8087CW(Saved8087CW);
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'No FPU Exceptions' then
    System.Set8087CW($133f); { Disable all fpu exceptions }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('No FPU Exceptions');
  RadioGroup1.Items.Add('FPU Exceptions');
  RadioGroup1.ItemIndex := 2;
  Saved8087CW := Default8087CW;  // Save this because Set8087CW changes it!
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  System.Set8087CW(Saved8087CW); // Default value (with exceptions) is $1372
end;
```

See Also

Set8087CW (see page 3744)

38.6.10 System.DefaultSystemCodePage Variable

Pascal

```
DefaultSystemCodePage: Integer;
```

C++

```
int DefaultSystemCodePage;
```

File

System

Description

This is variable System.DefaultSystemCodePage.

38.6.11 System.DefaultTextLineBreakStyle Variable

Specifies the characters that are used by default to separate lines in text.

Pascal

```
DefaultTextLineBreakStyle: TTextLineBreakStyle = tlbsCRLF;
```

C++

```
TTextLineBreakStyle DefaultTextLineBreakStyle = tlbsCRLF;
```

File

System

Description

DefaultTextLineBreakStyle controls how the code editor represents the separation between lines of text in new files. For existing files, it maintains the line break style that the file currently uses.

See Also

SetLineBreakStyle (see page 3750)

38.6.12 System.DefaultUnicodeCodePage Variable

Pascal

```
DefaultUnicodeCodePage: Integer;
```

C++

```
int DefaultUnicodeCodePage;
```

File

System

Description

This is variable System.DefaultUnicodeCodePage.

38.6.13 System.DispCallByIDProc Variable

Pascal

```
DispCallByIDProc: Pointer;
```

C++

```
void * DispCallByIDProc;
```

File

System

Description

This is variable System.DispCallByIDProc.

38.6.14 System.DLLProc Variable

Points to a procedure invoked by a DLL entry point.

Pascal

```
DLLProc: Pointer;
```

C++

```
void * DLLProc;
```

File

System

Description

DLLProc is used to specify a procedure that is invoked every time a DLL's entry point is called. A procedure assigned to DLLProc must take one parameter of type Integer or int. For example,

- procedure LibraryProc(Reason: Integer); { Delphi }
- void LibraryProc(int Reason); // C++
- When the procedure is invoked, this single parameter contains a value between 0 and 3 as defined by the following group of constants (from the Windows unit):
- const
- DLL_PROCESS_DETACH = 0;
- DLL_PROCESS_ATTACH = 1;
- DLL_THREAD_ATTACH = 2;
- DLL_THREAD_DETACH = 3;
- For further details on the meaning of these constants, refer to the description of the DllEntryPoint function in the Win32 API online help.

Note: DLL_PROCESS_ATTACH is passed to the procedure only if the DLL's initialization code calls the procedure and specifies DLL_PROCESS_ATTACH as a parameter.

Note: When the Reason parameter is DLL_PROCESS_ATTACH, setting ExitCode (see page 3891) to a nonzero value causes the entry point to return false.

38.6.15 System.envp Variable

Pascal

```
envp: PPChar platform;
```

C++

```
PPChar platform envp;
```

File

System

Description

This is variable System.envp.

38.6.16 System.ErrorAddr Variable

Contains the address of a statement causing a runtime error.

Pascal

```
ErrorAddr: Pointer = nil;
```

C++

```
void * ErrorAddr = nil;
```

File

System

Description

If a program terminates normally or stops due to a call to Halt (see page 3693), ErrorAddr is nil (Delphi) or NULL (C++). If a program ends because of a runtime error, ErrorAddr contains the address of the statement in error.

See Also

ErrorProc (see page 3886)

ExceptProc (see page 3890)

AcquireExceptionObject (see page 3619)

ExceptAddr (see page 3663)

ExceptionErrorMessage

ExceptObject (see page 3664)

GetLastError (see page 3681)

38.6.17 System.ErrorProc Variable

Points to the RTL runtime error handler.

Pascal

```
ErrorProc: procedure (ErrorCode: Byte; ErrorAddr: Pointer);
```

C++

```
procedure (ErrorCode: Byte; ErrorAddr: Pointer) ErrorProc;
```

File

System

Description

In Delphi, ErrorProc is a procedure variable pointing to the runtime error handler. The standard ErrorProc procedure reports the runtime error and terminates the program.

The ErrorProc variable is retained primarily for compatibility with older code. Newer applications should not specify an alternate runtime error procedure using ErrorProc as SysUtils (see page 3967) uses this variable to convert runtime errors into exceptions.

If you use SysUtils (see page 3967) in your program, it will force ErrorProc to its own routine and convert the runtime error into an exception. When a runtime library error occurs, the ErrorProc is called. The ErrorCode parameter contains the runtime error number and the ErrorAddr (see page 3885) parameter contains the machine code address that caused the runtime error.

Delphi Examples:

```
{
This example demostrates the use of ErrorProc and how to catch runtime
errors.
}
```

```
procedure MyErrorProc(ErrorCode : Byte; ErrorPtr : Pointer);
var
  ErrorName : String;
begin
  { Get the "name" of the enum value using RTTI }
  ErrorName := GetEnumName(TypeInfo(TRuntimeError), ErrorCode);

  { Show an error box }
  MessageDlg(Format('Error with code %d (%s) risen at $%X.',
      [ErrorCode, ErrorName, Integer(ErrorPtr)]), mtError, [mbOK], 0);
end;

procedure TForm2.btIOErrorClick(Sender: TObject);
begin
  {
  Try to write something onto the console - will raise an
  exception.
  }
  WriteLn('This will generate an error because there is no' +
          ' console attached!');
end;

{$OVERFLOWCHECKS ON}
{$OPTIMIZATION OFF}
{$HINTS OFF}
procedure TForm2.btOverflowErrClick(Sender: TObject);
var
  b : Cardinal;
begin
  {
  Simulate an overflow: Note, enabled the overflow
  checking and disabled optimizations because Delphi's
  compiler will not compile this code otherwise.
  }
  b := $FFFFFFFF;
  b := b * b;
end;
{$HINTS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}

procedure TForm2.btRuntimeErrorClick(Sender: TObject);
begin
  { Simulate an access violation }
  System.Error(reAccessViolation);
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Register my System error proc }
  ErrorProc := MyErrorProc;
end;
```

See Also

ErrorAddr (see page 3885)

ExceptAddr (see page 3663)

ExceptionErrorMessage

ExceptObject (see page 3664)

GetLastError (see page 3681)

ExceptProc (see page 3890)

GetLastError (see page 3681)

AcquireExceptionObject (see page 3619)

38.6.18 System.ErrOutput Variable

ErrOutput specifies a write-only Text file associated with the process's error output file.

Pascal

```
ErrOutput: Text;
```

C++

```
Text ErrOutput;
```

File

System

Description

In Delphi, the ErrOutput variable is a write-only Text file associated with the process's error output file.

Note: On Windows, most processes don't have an error output file, and writing to ErrOutput raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have an error output file. There are various ways to view the error output. One simple method is to run the program from a shell window.

See Also

Input (see page 3895)

Output (see page 3901)

TextFile

38.6.19 System.ExceptClsProc Variable

ExceptClsProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptClsProc: Pointer;
```

C++

```
void * ExceptClsProc;
```

File

System

Description

By default, ExceptClsProc is defined internally to point to a function that handles exceptions on your behalf. It returns the exception class. If you override ExceptClsProc to point to your function, that function will be called when an exception occurs in a try-except block. It will be passed an updateable TExceptionRecord value that describes the exception.

Internally, the class returned is used to determine which exception handler to use before ExceptObjProc (see page 3889) is called to create the exception instance.

See Also

ExceptObjProc (see page 3889)

TExceptionRecord

38.6.20 System.ExceptionAcquired Variable

Pascal

```
ExceptionAcquired: Pointer;
```

C++

```
void * ExceptionAcquired;
```

File

System

Description

This is variable System.ExceptionAcquired.

38.6.21 System.ExceptionClass Variable

Pascal

```
ExceptionClass: TClass;
```

C++

```
TClass ExceptionClass;
```

File

System

Description

This is variable System.ExceptionClass.

38.6.22 System.ExceptObjProc Variable

ExceptObjProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptObjProc: Pointer;
```

C++

```
void * ExceptObjProc;
```

File

System

Description

ExceptClsProc (see page 3888) defines the function that returns a class that is used to determine which exception handler to use before an exception object is created.

You can override the default exception object creation by defining your own function that returns a TObject (see page 3364). You point ExceptObjProc to that function. It is passed the an updateable version of the TExceptionRecord.

See Also

ExceptClsProc (see page 3888)

TExceptionRecord

38.6.23 System.ExceptProc Variable

Points to the lowest-level exception handler.

Pascal

```
ExceptProc: Pointer;
```

C++

```
void * ExceptProc;
```

File

System

Description

In Delphi code, ExceptProc handles unhandled exceptions (exceptions that have not been caught in the except portion of a try..except statement.) If your application has a TApplication object, it automatically catches exceptions and generates an OnException event (to which you can respond using either the application's or a TApplicationEvents object's OnException event handler). Thus, because the TApplication object catches the exceptions, they never reach the ExceptProc handler. However, even if you have a TApplication object, it does not catch exceptions generated during initialization and finalization. These are handled by ExceptProc.

You can hook into ExceptProc to change how unhandled exceptions are reported. To do so, save the contexts of ExceptProc before changing it to the address of your own exception handler. The first statement of your new exception handler must then reinstall the saved value of ExceptProc. In this way, the ExceptProc procedure becomes part of a chain of exception procedures, each of which is executed in reverse order of installation.

The procedure assigned to ExceptProc must have a signature such as the following:

procedure ExceptHandler(ExceptObject (see page 3664): System::TObject (see page 3364); ExceptAddr (see page 3663): Pointer);

Note: :On Linux, ExceptProc may be called when most of the runtime has already shut down. This happens if an application terminates with an unhandled exception.

See Also

ErrorAddr (see page 3885)

ErrorProc (see page 3886)

OnException

TApplication

ExceptObject (see page 3664)

38.6.24 System.ExitCode Variable

Contains the application's exit code.

Pascal

```
ExitCode: Integer = 0;
```

C++

```
int ExitCode = 0;
```

File

System

Description

In Delphi code, the ExitCode variable contains the application's exit code. An exit procedure can learn the cause of termination by examining ExitCode. If the program terminates normally, ExitCode is zero. If the program terminates due to a call to Halt (see page 3693), ExitCode contains the value passed to Halt (see page 3693). If the program terminates due to a runtime error, ExitCode contains the error code.

Note: On Windows, setting the ExitCode variable to a value other than 0 in the initialization section of a unit will make loading of a DLL fail (DllMain will return false). You can also set ExitCode in code called from DllProc or DllProcEx to signal failure of loading.

Note: On Linux, although the ExitCode variable is accessible from shared object code, its value does not affect the loader of the shared object. Thus, using ExitCode in a shared object has no effect.

See Also

AddExitProc

AddTerminateProc

CallTerminateProcs

ExitProc (see page 3891)

38.6.25 System.ExitProc Variable

Points to a program's exit procedure (provided for backward compatibility).

Pascal

```
ExitProc: Pointer;
```

C++

```
void * ExitProc;
```

File

System

Description

In Delphi code, ExitProc points to an exit procedure in an application. The exit procedure always gets called as part of a program's termination. ExitProc is only used when generating executable files.

An exit procedure takes no parameters and must be compiled with a far procedure directive to force it to use the far call model.

Do not use ExitProc in a dynamically loaded package. This procedure is called when the process halts, not when a package or DLL is unloaded. If you set ExitProc from a package or DLL, it may interfere with the ExitProc of the host executable. Because ExitProc is not compatible with packages, it is recommended that you add code to the finalization section of a unit instead. Inside a DLL, you can use the DLLProc (see page 3884) variable instead.

See Also

AddExitProc

ExitCode (see page 3891)

AddTerminateProc

CallTerminateProcs

DLLProc (see page 3884)

38.6.26 System.ExitProcessProc Variable

Specifies the last procedure to execute before the application shuts down.

Pascal

```
ExitProcessProc: procedure;
```

C++

```
procedure ExitProcessProc;
```

File

System

Description

Assign a value to ExitProcessProc to specify a procedure that you want to execute when the application shuts down. The value of ExitProcessProc is a procedure with no parameters.

See Also

InitProc (see page 3894)

38.6.27 System.FileMode Variable

Indicates the access mode on typed and untyped files opened by the Reset (see page 3734) procedure.

Pascal

```
FileMode: Byte = 2;
```

C++

```
Byte FileMode = 2;
```

File

System

Description

In Delphi code, FileMode determines the access mode to use when typed and untyped files (not text files) are opened using the Reset (see page 3734) procedure.

The default FileMode is 2 (Read (see page 3727)/Write (see page 3789) access). Assigning another value to FileMode causes all subsequent Resets to use that mode. The SysUtils (see page 3967) unit declares a number of file open mode constants that can be assigned to the FileMode variable.

Note: Be sure to reset FileMode before calling Reset (see page 3734) with a read-only file. Even if you are only reading from the file, the default FileMode value (Read (see page 3727)/Write (see page 3789) access) will cause an attempt to open a read-only file to fail.

Warning: FileMode is declared as a variable, not as a threadvar. Therefore this variable is not thread-safe.

See Also

Reset (see page 3734)

Rewrite (see page 3736)

TFileStream

38.6.28 System.HeapAllocFlags Variable

Indicates how the memory manager obtains memory from the operating system.

Pascal

```
HeapAllocFlags: Word platform = 2;
```

C++

```
Word platform HeapAllocFlags = 2;
```

File

System

Description

On Windows, use HeapAllocFlags to customize the flags that the system memory manager uses when requesting blocks of memory from the operating system. By default, the value is GMEM_MOVEABLE.

HeapAllocFlags can be a combination of the following values:

GMEM_FIXED	Allocates fixed memory. The operating system can't move blocks, so there is no need to lock memory when obtaining a pointer. (can't be combined with GMEM_MOVEABLE).
GMEM_MOVEABLE	Allocates movable memory. In Win32, memory blocks are never moved in physical memory, but they can be moved within the default heap.

GMEM_ZEROINIT	Initializes memory contents to zero.
GMEM_MODIFY	Used to modify the attributes of an already-allocated block of memory.
GMEM_DDESHARE GMEM_SHARE	This flag is primarily for compatibility with 16-bit Window, but. can be used to enhance the performance of DDE operations. It should only be specified if the memory is to be used for DDE.
GPTR	Combines the GMEM_FIXED and GMEM_ZEROINIT flags.
GHND	Combines the GMEM_MOVEABLE and GMEM_ZEROINIT flags.

See Also

SetMemoryManager (see page 3751)

38.6.29 System.HPrevInst Variable

Pascal

```
HPrevInst: LongWord deprecated;
```

C++

```
LongWord deprecated HPrevInst;
```

File

System

Description

This is variable System.HPrevInst.

38.6.30 System.InitProc Variable

InitProc is the last installed initialization procedure.

Pascal

```
InitProc: Pointer;
```

C++

```
void * InitProc;
```

File

System

Description

Assign a value to InitProc to specify a procedure that you want to execute when the application starts up. The value of InitProc is a procedure with no parameters. For example:

In Delphi, the assignment can be made in the initialization section of a unit:

```
procedure MyInitProcedure;
...
initialization
  InitProc = @MyInitProcedure;
begin
```

```
end;
```

In C++, the assignment is most easily made in the project source file. This example is for Windows. On Linux applications, the code is similar.

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    // Note that call must precede call to Application->Initialize();
    System.InitProc = MyInitProcedure;
    Application->Initialize();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  catch (...)
  {
    try
    {
      throw Exception("");
    }
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
  }
  return 0;
}
```

Only one initialization procedure can be assigned to the InitProc variable. If your application defines more than one initialization procedure, only the last assignment to InitProc executes. To allow other initialization procedures to execute, you must "chain" the procedures together, calling the previously assigned value of InitProc from the new value.

See Also

TApplication

ExitProcessProc (see page 3892)

38.6.31 System.Input Variable

Specifies a read-only Text file associated with the processes standard Input file.

Pascal

```
Input: Text;
```

C++

```
Text Input;
```

File

System

Description

In Delphi, the Input variable is a read-only Text file associated with the process's standard Input file.

Note: On Windows, most processes don't have a standard Input file, and reading from Input raises an error. Delphi programs

have a standard Input file if they are linked as console applications.

Note: On Linux, all processes have a standard Input file. There are various ways to use the standard Input. One simple method is to run the program from a shell window.

See Also

Output (see page 3901)

TextFile

38.6.32 System.IsConsole Variable

Indicates whether the module was compiled as a console application.

Pascal

```
IsConsole: Boolean;
```

C++

```
Boolean IsConsole;
```

File

System

Description

The IsConsole variable is true if the module was linked as a console application. This applies both to an executable and a shared library.

Note: On Linux, IsConsole is always true.

See Also

IsLibrary (see page 3896)

IsMultiThread (see page 3897)

Input (see page 3895)

Output (see page 3901)

ErrOutput (see page 3888)

38.6.33 System.IsLibrary Variable

Indicates whether the module is a shared library.

Pascal

```
IsLibrary: Boolean;
```

C++

```
Boolean IsLibrary;
```

File

System

Description

The IsLibrary variable is true if the module is a dynamic link library (Windows) or shared object (Linux).

See Also

IsConsole (see page 3896)

IsMultiThread (see page 3897)

38.6.34 System.IsMultiThread Variable

Indicates whether the application spawned additional threads using BeginThread (see page 3632) or TThread objects.

Pascal

```
IsMultiThread: Boolean;
```

C++

```
Boolean IsMultiThread;
```

File

System

Description

IsMultiThread is set to true to indicate that the memory manager should support multiple threads. IsMultiThread is set to true by BeginThread (see page 3632) and class factories.

See Also

BeginThread (see page 3632)

IsConsole (see page 3896)

IsLibrary (see page 3896)

CoInitFlags

TComObjectFactory

38.6.35 System.JITEnable Variable

Controls when the just-in-time debugger is called.

Pascal

```
JITEnable: Byte platform = 0;
```

C++

```
Byte platform JITEnable = 0;
```

File

System

Description

Use JITEnable to indicate what types of exceptions trigger the just-in-time debugger. JITEnable only has an effect on applications that run under Windows.

When JITEnable is 0 (the default), all exceptions are handled using the application's exception handling mechanism. The just-in-time debugger is only called if the application provides no mechanism to catch exceptions. Note that any application that includes the Forms unit provides exception handling (via the Application object) for all but the initialization and finalization sections in Delphi units and does not trigger the just-in-time debugger.

When JITEnable is 1, any non-native exceptions (Exceptions raised by code that was compiled using other products) trigger the just-in-time debugger. Native exceptions are handled by the application's exception handling mechanism.

When JITEnable is greater than 1, all exceptions trigger the just-in-time debugger.

Note: When you install the IDE, it is registered as the just-in-time debugger. If you install other tools with debuggers, they may register themselves, overriding the registration of the IDE.

38.6.36 System.LibModuleList Variable

Lists module records for enumeration procedures.

Pascal

```
LibModuleList: PLibModule = nil;
```

C++

```
PLibModule LibModuleList = nil;
```

File

System

Description

LibModuleList implements a linked list of records for executables and packages in an application. LibModuleList is used by the EnumModules (see page 3658) and the EnumResourceModules (see page 3658) procedures.

See Also

EnumModules (see page 3658)

EnumResourceModules (see page 3658)

38.6.37 System.MainInstance Variable

Indicates the instance handle for the main executable.

Pascal

```
MainInstance: LongWord;
```

C++

```
LongWord MainInstance;
```

File

System

Description

Use MainInstance to obtain the instance handle for the main executable of an application. This is useful in applications that use runtime libraries or packages, when you need the handle for the executable rather than for the library.

Note:

See Also

HInstance

FindHInstance (see page 3673)

38.6.38 System.MainThreadID Variable

Indicates the instance handle for the current module's main thread of execution.

Pascal

```
MainThreadID: LongWord;
```

C++

```
LongWord MainThreadID;
```

File

System

Description

Use MainThreadID to obtain the thread ID for the main execution thread (the current thread when the module was initialized).

During debugging, this ID appears in the thread status box for the main thread.

Note: On Windows, the value of MainThreadID is compatible with the thread ID required by some Win32 API calls.

See Also

TThread

38.6.39 System.ModuleUnloadList Variable

Pascal

```
ModuleUnloadList: PModuleUnloadRec = nil;
```

C++

```
PModuleUnloadRec ModuleUnloadList = nil;
```

File

System

Description

This is variable System.ModuleUnloadList.

38.6.40 System.MonitorSupport Variable

Pascal

```
MonitorSupport: PMonitorSupport;
```

C++

```
PMonitorSupport MonitorSupport;
```

File

System

Description

This is variable System.MonitorSupport.

38.6.41 System.NeverSleepOnMMThreadContention Variable

Dictates memory manager behavior during a thread contention.

Pascal

```
NeverSleepOnMMThreadContention: Boolean;
```

C++

```
Boolean NeverSleepOnMMThreadContention;
```

File

System

Description

The Memory Manager is a shared resource, and when many threads in the application attempt to perform a memory operation at the same time, one or more threads may have to wait for a pending operation in another thread to complete before it can continue. This situation is called a thread contention. When a thread contention occurs inside the Memory Manager the default behavior is to first relinquish the remaining time in the thread's time slice. If the resource is still not available when the thread enters its next time slice, the Memory Manager calls the OS procedure Sleep before trying again.

This behavior works well on machines with single or dual core CPUs, and also when the ratio of the number of running threads to number of CPU cores is relatively high (greater than 2:1). In other situations, better performance can be obtained by entering a busy waiting loop until the resource becomes available. If NeverSleepOnMMThreadContention is **True**, the Memory Manager will use this alternate strategy when a thread contention occurs. The default value for NeverSleepOnMMThreadContention is **False**.

See Also

Memory Management

Configuring the Memory Manager

38.6.42 System.NoErrMsg Variable

Controls whether the application displays an error message when a runtime error occurs.

Pascal

```
NoErrMsg: Boolean platform = False;
```

C++

```
Boolean platform NoErrMsg = False;
```

File

System

Description

Note: On Windows, set NoErrMsg to control whether a message appears to indicate that a runtime error has occurred. When NoErrMsg is false (the default), runtime errors cause the application to display a message box indicating the type of error that occurred. When NoErrMsg is true, these messages are suppressed.

On Linux, NoErrMsg only no effect.

Note: The SysUtils (see page 3967) unit turns most runtime errors into exceptions. If your application includes SysUtils (see page 3967), the runtime error message box is unlikely to appear, even if NoErrMsg is false.

38.6.43 System.Output Variable

Specifies a write-only Text file associated with the process's standard output file.

Pascal

```
Output: Text;
```

C++

```
Text Output;
```

File

System

Description

In Delphi, the Output variable is a write-only Text file associated with the process's standard output file.

Note: On Windows, most processes don't have a standard output file, and writing to Output raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have a standard output file. There are various ways to view the standard error output. One simple method is to run the program from a shell window.

See Also

Input (see page 3895)

TextFile

38.6.44 System.RaiseExceptionProc Variable

Points to RaiseException Windows function.

Pascal

```
RaiseExceptionProc: Pointer;
```

C++

```
void * RaiseExceptionProc;
```

File

System

Description

RaiseExceptionProc points to the RaiseException Windows API function. This can be used to invoke the function from assembly language.

See Also

RaiseException Function (MSDN)

38.6.45 System.RaiseExceptObjProc Variable

Pascal

```
RaiseExceptObjProc: Pointer;
```

C++

```
void * RaiseExceptObjProc;
```

File

System

Description

This is variable System.RaiseExceptObjProc.

38.6.46 System.RandSeed Variable

RandSeed stores the built-in random number generator's seed.

Pascal

```
RandSeed: Longint = 0;
```

C++

```
Longint RandSeed = 0;
```

File

System

Description

By assigning a specific value to RandSeed, you can use the Random (see page 3725) function to repetitively generate a specific sequence of random numbers. This is useful for applications that deal with data statistics and simulations. For applications that deal with encryption see the note under the Random (see page 3725) function.

See Also

Random (see page 3725)

Randomize (see page 3726)

38.6.47 System.ReportMemoryLeaksOnShutdown Variable

Report memory leaks on shutdown.

Pascal

```
ReportMemoryLeaksOnShutdown: Boolean;
```

C++

```
Boolean ReportMemoryLeaksOnShutdown;
```

File

System

Description

Set ReportMemoryLeaksOnShutdown to report memory leaks on shutdown.

The memory manager can report memory that was allocated but not freed by the time the memory manager shuts down. Such memory blocks are called memory leaks and are often the result of programming errors. When this global variable is set to **True**, the Memory Manager will scan the memory pool when it shuts down and report all unregistered memory leaks in a message dialog. The default value for ReportMemoryLeaksOnShutdown is **False**.

Expected memory leaks can be registered so the Memory Manager will ignore them during its checks for memory leaks. Use the procedures RegisterExpectedMemoryLeak (see page 3731) and UnregisterExpectedMemoryLeak (see page 3777) to control the memory areas that the Memory Manager checks for leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak

UnregisterExpectedMemoryLeak

38.6.48 System.RTLUnwindProc Variable

Pascal

```
RTLUnwindProc: Pointer;
```

C++

```
void * RTLUnwindProc;
```

File

System

Description

This is variable System.RTLUnwindProc.

38.6.49 System.SafeCallErrorProc Variable

SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.

Pascal

```
SafeCallErrorProc: TSafeCallErrorProc;
```

C++

```
TSafeCallErrorProc SafeCallErrorProc;
```

File

System

Description

A function or procedure defined with the safecall directive is invoked using a safe calling convention. When such a routine incurs an error, the System::TObject::SafeCallException (see page 3391) is invoked. However, you can override this call by defining your own handling routine, and setting SafeCallErrorProc to point to it.

SafeCallErrorProc should point to a procedure that takes two parameters :

ErrorCode is an integer holding the error code.

ErrorAddr (see page 3885) is the code address at the time of error.

If SafeCallErrorProc returns normally, or is set to nil, then the ESafeCallException is raised.

See Also

ESafeCallException

System::TObject::SafeCallException (see page 3391)

38.6.50 System.SystemThreadEndProc Variable

Pascal

```
SystemThreadEndProc: TSystemThreadEndProc = nil;
```

C++

```
TSystemThreadEndProc SystemThreadEndProc = nil;
```

File

System

Description

This is variable System.SystemThreadEndProc.

38.6.51 System.SystemThreadFuncProc Variable

Pascal

```
SystemThreadFuncProc: TSystemThreadFuncProc = nil;
```

C++

```
TSystemThreadFuncProc SystemThreadFuncProc = nil;
```

File

System

Description

This is variable System.SystemThreadFuncProc.

38.6.52 System.Test8086 Variable

Pascal

```
Test8086: Byte;
```

C++

```
Byte Test8086;
```

File

System

Description

This is variable System.Test8086.

38.6.53 System.Test8087 Variable

Pascal

```
Test8087: Byte = 3;
```

C++

```
Byte Test8087 = 3;
```

File

System

Description

This is variable System.Test8087.

38.6.54 System.TestFDIV Variable

Pascal

```
TestFDIV: Shortint;
```

C++

```
Shortint TestFDIV;
```

File

System

Description

This is variable System.TestFDIV.

38.6.55 System.UTF8CompareLocale Variable

Pascal

```
UTF8CompareLocale: Cardinal;
```

C++

```
unsigned UTF8CompareLocale;
```

File

System

Description

This is variable System.UTF8CompareLocale.

38.6.56 System.VarAddRefProc Variable

Pascal

```
VarAddRefProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarAddRefProc = nil;
```

File

System

Description

This is variable System.VarAddRefProc.

38.6.57 System.VarClearProc Variable

Pascal

```
VarClearProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarClearProc = nil;
```

File

System

Description

This is variable System.VarClearProc.

38.6.58 System.VarCopyProc Variable

Pascal

```
VarCopyProc: procedure (var Dest: TVarData; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: TVarData; const Source: TVarData) VarCopyProc = nil;
```

File

System

Description

This is variable System.VarCopyProc.

38.6.59 System.VarToLStrProc Variable

Pascal

```
VarToLStrProc: procedure (var Dest: AnsiString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: AnsiString; const Source: TVarData) VarToLStrProc = nil;
```

File

System

Description

This is variable System.VarToLStrProc.

38.6.60 System.VarToWStrProc Variable

Pascal

```
VarToWStrProc: procedure (var Dest: WideString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: WideString; const Source: TVarData) VarToWStrProc = nil;
```

File

System

Description

This is variable System.VarToWStrProc.

38.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CompilerVersion (see page 3911)	Designates the version number of the Delphi compiler.
CPUi386 (see page 3911)	This is constant System.CPUi386.
CPUi486 (see page 3912)	This is constant System.CPUi486.
CPUPentium (see page 3912)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 3912)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 3913)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 3913)	This is constant System.E_UNEXPECTED.
fmClosed (see page 3913)	This is constant System.fmClosed.
fmInOut (see page 3914)	This is constant System.fmInOut.
fmInput (see page 3914)	This is constant System.fmInput.
fmOutput (see page 3914)	This is constant System.fmOutput.
hfFieldSize (see page 3915)	This is constant System.hfFieldSize.
hfMonitorOffset (see page 3915)	This is constant System.hfMonitorOffset.
INFINITE (see page 3915)	This is constant System.INFINITE.
MaxInt (see page 3916)	The maximum value of the Integer data type.
MaxLongint (see page 3916)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 3916)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 3917)	opAdd defines an addition operation.
opAnd (see page 3917)	opAnd defines a logical and operation.
opCmpEQ (see page 3918)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 3918)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 3919)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 3919)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 3920)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 3920)	opCmpNE defines an inequality comparison operator.
opCompare (see page 3921)	opCompare defines a comparison operation.
opDivide (see page 3921)	opDivide defines a divide operation.
opIntDivide (see page 3922)	opIntDivide defines an integer divide operation.
opModulus (see page 3922)	opModulus defines a modulus operation.
opMultiply (see page 3923)	opMultiply defines a multiply operation.
opNegate (see page 3923)	opNegate defines a negate operation.
opNot (see page 3924)	opNot defines a binary negation operation.
opOr (see page 3924)	opOr defines a logical or operation.
opShiftLeft (see page 3925)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 3925)	opShiftRight defines a bitwise shift right operation.
opSubtract (see page 3926)	opSubtract defines a subtraction operation.
opXor (see page 3926)	opXor defines a logical exclusive or operation.
RTLVersion (see page 3927)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 3927)	This is constant System.S_FALSE.
S_OK (see page 3927)	This is constant System.S_OK.
sLineBreak (see page 3928)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 3928)	This is constant System.tfCRLF.
varAny (see page 3928)	Represents a Variant (see page 3430) that can hold any value.
varArray (see page 3929)	Represents a Variant (see page 3430) that holds dynamic array data.

varBoolean (see page 3930)	Represents a Variant (see page 3430) that holds WordBool data.
varByRef (see page 3930)	Represents a Variant (see page 3430) that points to another variable.
varByte (see page 3931)	Represents a Variant (see page 3430) that holds byte data.
varCurrency (see page 3931)	Represents a Variant (see page 3430) that holds a monetary floating point value.
varDate (see page 3932)	Represents a Variant (see page 3430) that holds a TDateTime (see page 3335) value.
varDispatch (see page 3932)	Represents a Variant (see page 3430) that has been assigned an IDispatch (see page 3498) value.
varDouble (see page 3933)	Represents a Variant (see page 3430) that holds a floating point value.
varEmpty (see page 3933)	Represents a Variant (see page 3430) that has not yet been assigned a value.
varError (see page 3934)	Represents a Variant (see page 3430) that holds a 32 bit error code.
varInt64 (see page 3934)	Represents a Variant (see page 3430) that holds Int64 data.
varInteger (see page 3935)	Represents a Variant (see page 3430) that holds Integer data.
varLongWord (see page 3935)	Represents a Variant (see page 3430) that holds LongWord data.
varNull (see page 3936)	Represents a Variant (see page 3430) that holds a null value.
varOleStr (see page 3936)	Represents a Variant (see page 3430) that has been assigned a COM interface type string value.
varShortInt (see page 3937)	Represents a Variant (see page 3430) that holds ShortInt data.
varSingle (see page 3937)	Represents a Variant (see page 3430) that holds byte data.
varSmallint (see page 3938)	This is constant System.varSmallint.
varStrArg (see page 3938)	Represents a Variant (see page 3430) that has holds a COM compatible string value.
varString (see page 3938)	Represents a Variant (see page 3430) that holds string data.
varTypeMask (see page 3939)	Defines a mask for filtering out the meta data part of a Variant (see page 3430) data type.
varUInt64 (see page 3940)	This is constant System.varUInt64.
varUnknown (see page 3940)	Represents a Variant (see page 3430) that holds an unknown value.
varUString (see page 3940)	This is constant System.varUString.
varVariant (see page 3941)	Represents a Variant (see page 3430) data type.
varWord (see page 3941)	Represents a Variant (see page 3430) that holds Word data.
vmtAddRef (see page 3942)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 3942)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 3942)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 3943)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 3943)	This is constant System.vmtClassName.
vmtCreateObject (see page 3943)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 3944)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 3944)	This is constant System.vmtDestroy.
vmtDispatch (see page 3944)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 3945)	This is constant System.vmtDynamicTable.
vmtEquals (see page 3945)	This is constant System.vmtEquals.
vmtFieldTable (see page 3945)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 3946)	This is constant System.vmtFreeInstance.
vmtGetHashCode (see page 3946)	This is constant System.vmtGetHashCode.
vmtInitTable (see page 3946)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 3947)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 3947)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 3947)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 3948)	This is constant System.vmtNewInstance.
vmtParent (see page 3948)	This is constant System.vmtParent.
vmtQueryInterface (see page 3948)	This is constant System.vmtQueryInterface.
vmtRelease (see page 3949)	This is constant System.vmtRelease.
vmtSafeCallException (see page 3949)	This is constant System.vmtSafeCallException.
vmtSelfPtr (see page 3949)	This is constant System.vmtSelfPtr.
vmtToString (see page 3950)	This is constant System.vmtToString.

vmtTypeInfo (see page 3950)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 3950)	vtAnsiString defines an AnsiString TVarData (see page 3813) VType value.
vtBoolean (see page 3951)	vtBoolean defines a Boolean TVarData (see page 3813) VType value.
vtChar (see page 3951)	vtChar defines a Char TVarData (see page 3813) VType value.
vtClass (see page 3952)	vtClass defines a TClass (see page 3856) TVarData (see page 3813) VType value.
vtCurrency (see page 3952)	vtCurrency defines a Currency (see page 3242) TVarData (see page 3813) VType value.
vtExtended (see page 3953)	vtExtended defines an Extended TVarData (see page 3813) VType value.
vtInt64 (see page 3953)	vtInt64 defines an Int64 TVarData (see page 3813) VType value.
vtInteger (see page 3954)	vtInteger defines an Integer TVarData (see page 3813) VType value.
vtInterface (see page 3954)	vtInterface defines an Interface pointer TVarData (see page 3813) VType value.
vtObject (see page 3955)	vtObject defines a TObject (see page 3364) TVarData (see page 3813) VType value.
vtPChar (see page 3955)	vtPChar defines a PChar TVarData (see page 3813) VType value.
vtPointer (see page 3956)	vtPointer defines a Pointer TVarData (see page 3813) VType value.
vtPWideChar (see page 3956)	vtPWideChar defines a PWideChar TVarData (see page 3813) VType value.
vtString (see page 3957)	vtString defines a Variant (see page 3430) TVarData (see page 3813) VType value.
vtUnicodeString (see page 3957)	This is constant System.vtUnicodeString.
vtVariant (see page 3957)	This is constant System.vtVariant.
vtWideChar (see page 3958)	vtWideChar defines a WideChar TVarData (see page 3813) VType value.
vtWideString (see page 3958)	vtWideString defines a WideString (see page 3476) TVarData (see page 3813) VType value.

38.7.1 System.CompilerVersion Constant

Designates the version number of the Delphi compiler.

Pascal

```
CompilerVersion = 18.0;
```

C++

```
CompilerVersion = 18.0;
```

File

System

Description

Use CompilerVersion to test the version level of the Delphi compiler. This is useful if you want to conditionalize code that takes advantage of newer features in the Delphi language.

See Also

RTLVersion (see page 3927)

38.7.2 System.CPUi386 Constant

Pascal

```
CPUi386 = 2;
```

C++

```
CPUi386 = 2;
```

File

System

Description

This is constant System.CPUi386.

38.7.3 System.CPUi486 Constant

Pascal

```
CPUi486 = 3;
```

C++

```
CPUi486 = 3;
```

File

System

Description

This is constant System.CPUi486.

38.7.4 System.CPUPentium Constant

Pascal

```
CPUPentium = 4;
```

C++

```
CPUPentium = 4;
```

File

System

Description

This is constant System.CPUPentium.

38.7.5 System.E_NOINTERFACE Constant

Pascal

```
E_NOINTERFACE = HRESULT($80004002);
```

C++

```
E_NOINTERFACE = HRESULT($80004002);
```

File

System

Description

This is constant System.E_NOINTERFACE.

38.7.6 System.E_NOTIMPL Constant

Pascal

```
E_NOTIMPL = HRESULT($80004001);
```

C++

```
E_NOTIMPL = HRESULT($80004001);
```

File

System

Description

This is constant System.E_NOTIMPL.

38.7.7 System.E_UNEXPECTED Constant

Pascal

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

C++

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

File

System

Description

This is constant System.E_UNEXPECTED.

38.7.8 System.fmClosed Constant

Pascal

```
fmClosed = $D7B0;
```

C++

```
fmClosed = 0xD7B0;
```

File

System

Description

This is constant System.fmClosed.

38.7.9 System.fmInOut Constant

Pascal

```
fmInOut = $D7B3;
```

C++

```
fmInOut = 0xD7B3;
```

File

System

Description

This is constant System.fmInOut.

38.7.10 System.fmInput Constant

Pascal

```
fmInput = $D7B1;
```

C++

```
fmInput = 0xD7B1;
```

File

System

Description

This is constant System.fmInput.

38.7.11 System.fmOutput Constant

Pascal

```
fmOutput = $D7B2;
```

C++

```
fmOutput = 0xD7B2;
```

File

System

Description

This is constant System.fmOutput.

38.7.12 System.hfFieldSize Constant

Pascal

```
hfFieldSize = 4;
```

C++

```
hfFieldSize = 4;
```

File

System

Description

This is constant System.hfFieldSize.

38.7.13 System.hfMonitorOffset Constant

Pascal

```
hfMonitorOffset = 0;
```

C++

```
hfMonitorOffset = 0;
```

File

System

Description

This is constant System.hfMonitorOffset.

38.7.14 System.INFINITE Constant

Pascal

```
INFINITE = Cardinal($FFFFFFFF);
```

C++

```
INFINITE = Cardinal($FFFFFFFF);
```

File

System

Description

This is constant System.INFINITE.

38.7.15 System.MaxInt Constant

The maximum value of the Integer data type.

Pascal

```
MaxInt = 2147483647;
```

C++

```
MaxInt = 2147483647;
```

File

System

Description

MaxInt represents the highest value in the range of the integer data type (2147483647).

See Also

High (see page 3694)

Integer

38.7.16 System.MaxLongint Constant

The maximum value of the Longint data type.

Pascal

```
MaxLongint = 2147483647;
```

C++

```
MaxLongint = 2147483647;
```

File

System

Description

MaxLongint represents the highest value in the range of the Longint (Delphi) or int (C++) data type (2147483647).

See Also

High (see page 3694)

Longint

38.7.17 System.NumSmallBlockTypes Constant

NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.

Pascal

```
NumSmallBlockTypes = 55;
```

C++

```
NumSmallBlockTypes = 55;
```

File

System

Description

GetMemoryManagerState (see page 3690) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a TMemoryManagerState (see page 3806) record. Within that record is held a TSmallBlockTypeStates (see page 3865) field - an array of TSmallBlockTypeState (see page 3810) records. There are NumSmallBlockTypes (55) different small memory block sizes, and therefore 55 TSmallBlockTypeState (see page 3810) records in this array.

See Also

GetMemoryManagerState (see page 3690)

TMemoryManagerState (see page 3806)

TSmallBlockTypeState (see page 3810)

TSmallBlockTypeStates (see page 3865)

38.7.18 System.opAdd Constant

opAdd defines an addition operation.

Pascal

```
opAdd = 0;
```

C++

```
opAdd = 0;
```

File

System

Description

opAdd is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opAdd represents an addition of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.19 System.opAnd Constant

opAnd defines a logical and operation.

Pascal

```
opAnd = 8;
```

C++

```
opAnd = 8;
```

File

System

Description

opAnd is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opAnd represents a logical and of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.20 System.opCmpEQ Constant

opCmpEQ defines an equality comparison operator.

Pascal

```
opCmpEQ = 14;
```

C++

```
opCmpEQ = 14;
```

File

System

Description

opCmpEQ is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpEQ represents a comparison that two operands are equal in value.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.21 System.opCmpGE Constant

opCmpGE defines a greater than or equal comparison operator.

Pascal

```
opCmpGE = 19;
```

C++

```
opCmpGE = 19;
```

File

System

Description

opCmpGE is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGE represents the comparison : left operand greater than or equal to the right operand.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.22 System.opCmpGT Constant

opCmpGT defines a greater than comparison operator.

Pascal

```
opCmpGT = 18;
```

C++

```
opCmpGT = 18;
```

File

System

Description

opCmpGT is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGT represents the comparison : left operand greater than the right operand.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.23 System.opCmpLE Constant

opCmpLE defines a less than or equal comparison operator.

Pascal

```
opCmpLE = 17;
```

C++

```
opCmpLE = 17;
```

File

System

Description

opCmpLE is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpLE represents the comparison : left operand less than or equal to the right operand.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.24 System.opCmpLT Constant

opCmpLT defines a less than comparison operator.

Pascal

```
opCmpLT = 16;
```

C++

```
opCmpLT = 16;
```

File

System

Description

opCmpLT is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpLT represents the comparison : left operand less than the right operand.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.25 System.opCmpNE Constant

opCmpNE defines an inequality comparison operator.

Pascal

```
opCmpNE = 15;
```

C++

```
opCmpNE = 15;
```

File

System

Description

opCmpNE is one of the opCmp* comparison values that TVarOp (see page 3869) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpNE represents a comparison that two operands are not equal in value.

See Also

CompareOp (see page 5021)

CompareOp

TVarOp (see page 3869)

38.7.26 System.opCompare Constant

opCompare defines a comparison operation.

Pascal

```
opCompare = 11;
```

C++

```
opCompare = 11;
```

File

System

Description

opCompare defines a comparison operation.

See Also

TVarOp (see page 3869)

38.7.27 System.opDivide Constant

opDivide defines a divide operation.

Pascal

```
opDivide = 3;
```

C++

```
opDivide = 3;
```

File

System

Description

opDivide is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opDivide represents the division of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.28 System.opIntDivide Constant

opIntDivide defines an integer divide operation.

Pascal

```
opIntDivide = 4;
```

C++

```
opIntDivide = 4;
```

File

System

Description

opIntDivide is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opIntDivide represents the integer division of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.29 System.opModulus Constant

opModulus defines a modulus operation.

Pascal

```
opModulus = 5;
```

C++

```
opModulus = 5;
```

File

System

Description

opModulus is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opModulus represents the remainder of the division of the left operand by the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.30 System.opMultiply Constant

opMultiply defines a multiply operation.

Pascal

```
opMultiply = 2;
```

C++

```
opMultiply = 2;
```

File

System

Description

opMultiply is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opMultiply represents the multiplication of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.31 System.opNegate Constant

opNegate defines a negate operation.

Pascal

```
opNegate = 12;
```

C++

```
opNegate = 12;
```

File

System

Description

opNegate is one of the operation types that TVarOp (see page 3869) may be set to when performing unary operations such as UnaryOp.

opNegate represents the inversion of an operand value.

See Also

UnaryOp

TVarOp (see page 3869)

38.7.32 System.opNot Constant

opNot defines a binary negation operation.

Pascal

```
opNot = 13;
```

C++

```
opNot = 13;
```

File

System

Description

opNegate (see page 3923) represents the binary (bitwise) inversion of an operand value.

See Also

TVarOp (see page 3869)

38.7.33 System.opOr Constant

opOr defines a logical or operation.

Pascal

```
opOr = 9;
```

C++

```
opOr = 9;
```

File

System

Description

opOr is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opOr represents a logical or of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.34 System.opShiftLeft Constant

opShiftLeft defines a bitwise shift left operation.

Pascal

```
opShiftLeft = 6;
```

C++

```
opShiftLeft = 6;
```

File

System

Description

opShiftLeft is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opShiftLeft represents the left operand shifted left N times, where N is the numeric value of the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.35 System.opShiftRight Constant

opShiftRight defines a bitwise shift right operation.

Pascal

```
opShiftRight = 7;
```

C++

```
opShiftRight = 7;
```

File

System

Description

opShiftRight is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opShiftRight represents the left operand shifted right N times, where N is the numeric value of the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.36 System.opSubtract Constant

opSubtract defines a subtraction operation.

Pascal

```
opSubtract = 1;
```

C++

```
opSubtract = 1;
```

File

System

Description

opSubtract is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opSubtract represents the right operand subtracted from the left operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.37 System.opXor Constant

opXor defines a logical exclusive or operation.

Pascal

```
opXor = 10;
```

C++

```
opXor = 10;
```

File

System

Description

opXor is one of the operation types that TVarOp (see page 3869) may be set to when performing binary operations such as BinaryOp.

opXor represents a logical exclusive or of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 3869)

38.7.38 System.RTLVersion Constant

Designates the version number of the Delphi runtime library.

Pascal

```
RTLVersion = 20.00;
```

C++

```
RTLVersion = 20.00;
```

File

System

Description

Use RTLVersion to test the Delphi runtime library version level independently of the compiler version level.

See Also

CompilerVersion (see page 3911)

38.7.39 System.S_FALSE Constant

Pascal

```
S_FALSE = $00000001;
```

C++

```
S_FALSE = 0x00000001;
```

File

System

Description

This is constant System.S_FALSE.

38.7.40 System.S_OK Constant

Pascal

```
S_OK = 0;
```

C++

```
S_OK = 0;
```

File

System

Description

This is constant System.S_OK.

38.7.41 System.sLineBreak Constant

Represents the character(s) used to generate a line feed.

Pascal

```
sLineBreak = AnsiString(#13#10);
```

C++

```
sLineBreak = AnsiString(#13#10);
```

File

System

Description

Windows systems define sLineBreak as a combination of carriage return (#13) and line feed (#10).

Linux systems define sLineBreak as simply a line feed (#10).

It is preferable to use this constabnt rather than hard code #13#10 or #10 (as appropriate) to move to a new line in a text output.

38.7.42 System.tfCRLF Constant

Pascal

```
tfCRLF = $1;
```

C++

```
tfCRLF = 0x1;
```

File

System

Description

This is constant System.tfCRLF.

38.7.43 System.varAny Constant

Represents a Variant (see page 3430) that can hold any value.

Pascal

```
varAny = $0101;
```

C++

```
varAny = 0x0101;
```

File

System

Description

A variant with a varAny type is used as a placeholder for any variant value, such as when passed as a paremeter to a routine. A varAny variant cannot be directly processed - use ChangeAnyProc to convert it to a variant type that can be processed.

See Also

TVarType

ChangeAnyProc (see page 5111)

VarType (see page 5101)

38.7.44 System.varArray Constant

Represents a Variant (see page 3430) that holds dynamic array data.

Pascal

```
varArray = $2000;
```

C++

```
varArray = 0x2000;
```

File

System

Description

The VarType routine returns a TVarType value that describes the current data type used to hold the contents of a Variant (see page 3430) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 3933), varSmallInt.

The high order bits represent meta data types varArray and varByRef (see page 3930).

Logically And varArray with a Variant (see page 3430) to determine if a Variant (see page 3430) holds array data.

Use VarArrayCreate to generate an array Variant (see page 3430);

See Also

TVarType

VarArrayCreate (see page 5077)

VarType (see page 5101)

38.7.45 System.varBoolean Constant

Represents a Variant (see page 3430) that holds WordBool data.

Pascal

```
varBoolean = $000B;
```

C++

```
varBoolean = 0x000B;
```

File

System

Description

varBoolean represents a Variant (see page 3430) variable that currently holds a 16-bit boolean (type WordBool) value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.46 System.varByRef Constant

Represents a Variant (see page 3430) that points to another variable.

Pascal

```
varByRef = $4000;
```

C++

```
varByRef = 0x4000;
```

File

System

Description

The VarType routine returns a TVarType value that describes the current data type used to hold the contents of a Variant (see page 3430) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 3933), varSmallInt.

The high order bits represent meta data types varArray (see page 3929) and varByRef.

In essence, a varByRef variant is a pointer. The data type of the variable it points to is determined in the low order bits.

Logically And varByRef with a Variant (see page 3430) to determine if a Variant (see page 3430) is a pointer.

See Also

TVarType

VarType (see page 5101)

38.7.47 System.varByte Constant

Represents a Variant (see page 3430) that holds byte data.

Pascal

```
varByte = $0011;
```

C++

```
varByte = 0x0011;
```

File

System

Description

When a Variant (see page 3430) is assigned an integer value less than 256, it will be set to a varByte Variant (see page 3430) type. varByte represents a Variant (see page 3430) variable that currently holds a byte data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.48 System.varCurrency Constant

Represents a Variant (see page 3430) that holds a monetary floating point value.

Pascal

```
varCurrency = $0006;
```

C++

```
varCurrency = 0x0006;
```

File

System

Description

When a Variant (see page 3430) is assigned a floating point value with less than 5 decimal places, it is set to the varCurrency type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.49 System.varDate Constant

Represents a Variant (see page 3430) that holds a TDateTime (see page 3335) value.

Pascal

```
varDate = $0007;
```

C++

```
varDate = 0x0007;
```

File

System

Description

When a Variant (see page 3430) is assignned a TDateTime (see page 3335) value, it is set to the varDate type.

Use VarType to determine the variant type.

See Also

TDateTime (see page 3858)

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.50 System.varDispatch Constant

Represents a Variant (see page 3430) that has been assigned an IDispatch (see page 3498) value.

Pascal

```
varDispatch = $0009;
```

C++

```
varDispatch = 0x0009;
```

File

System

Description

When a Variant (see page 3430) is assigned an IDispatch (see page 3498) value, it is set to the varDispatch type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.51 System.varDouble Constant

Represents a Variant (see page 3430) that holds a floating point value.

Pascal

```
varDouble = $0005;
```

C++

```
varDouble = 0x0005;
```

File

System

Description

When a Variant (see page 3430) is assigned a floating point value with more than 4 decimal places, it is set to the varDouble type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.52 System.varEmpty Constant

Represents a Variant (see page 3430) that has not yet been assigned a value.

Pascal

```
varEmpty = $0000;
```

C++

```
varEmpty = 0x0000;
```

File

System

Description

When a Variant (see page 3430) is first created, it is set to the varEmpty type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarIsEmpty (see page 5091)

VarType (see page 5101)

38.7.53 System.varError Constant

Represents a Variant (see page 3430) that holds a 32 bit error code.

Pascal

```
varError = $000A;
```

C++

```
varError = 0x000A;
```

File

System

Description

When a Variant (see page 3430) is assigned a 32 bit error code, it is set to the varError type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarIsEmpty (see page 5091)

VarType (see page 5101)

38.7.54 System.varInt64 Constant

Represents a Variant (see page 3430) that holds Int64 data.

Pascal

```
varInt64 = $0014;
```

C++

```
varInt64 = 0x0014;
```

File

System

Description

When a Variant (see page 3430) is assigned an integer value greater in value than an LongWord type can hold, it will be set to a varInt64 Variant (see page 3430) type. varInt64 represents a Variant (see page 3430) variable that currently holds an Int64 data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.55 System.varInteger Constant

Represents a Variant (see page 3430) that holds Integer data.

Pascal

```
varInteger = $0003;
```

C++

```
varInteger = 0x0003;
```

File

System

Description

To define a Variant (see page 3430) to be a varInteger type, you should use VarAsType to create one. By default, a Variant (see page 3430) with an integer value greater than the Word type capacity, but less than the Int64 type capacity will normally be assigned a varLongWord (see page 3935) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.56 System.varLongWord Constant

Represents a Variant (see page 3430) that holds LongWord data.

Pascal

```
varLongWord = $0013;
```

C++

```
varLongWord = 0x0013;
```

File

System

Description

When a Variant (see page 3430) is assigned an integer value greater in value than a Word type can hold, but less than an Int64 type can hold, it will be set to a varLongWord Variant (see page 3430) type. varLongWord represents a Variant (see page 3430) variable that currently holds a LongWord data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.57 System.varNull Constant

Represents a Variant (see page 3430) that holds a null value.

Pascal

```
varNull = $0001;
```

C++

```
varNull = 0x0001;
```

File

System

Description

When a Variant (see page 3430) is assigned a null value, it is set to the varNull type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarIsNull (see page 5093)

VarType (see page 5101)

38.7.58 System.varOleStr Constant

Represents a Variant (see page 3430) that has been assigned a COM interface type string value.

Pascal

```
varOleStr = $0008;
```

C++

```
varOleStr = 0x0008;
```

File

System

Description

When a Variant (see page 3430) is assigned a string value via a COM interface, it is set to the varOleStr type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.59 System.varShortInt Constant

Represents a Variant (see page 3430) that holds ShortInt data.

Pascal

```
varShortInt = $0010;
```

C++

```
varShortInt = 0x0010;
```

File

System

Description

To define a Variant (see page 3430) to be a varShortInt type, you should use VarAsType to create one. By default, a Variant (see page 3430) with an integer value less than the Word type capacity will normally be assigned a varByte (see page 3931) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.60 System.varSingle Constant

Represents a Variant (see page 3430) that holds byte data.

Pascal

```
varSingle = $0004;
```

C++

```
varSingle = 0x0004;
```

File

System

Description

To define a Variant (see page 3430) to be a varSingle type, you should use VarAsType to create one. By default, a Variant (see page 3430) with a floating point value with less than 4 decimal places will be assigned the varCurrency (see page 3931) type. With greater than 4 decimal places, it will be assigned by default the varDouble (see page 3933) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.61 System.varSmallint Constant

Pascal

```
varSmallint = $0002;
```

C++

```
varSmallint = 0x0002;
```

File

System

Description

This is constant System.varSmallint.

38.7.62 System.varStrArg Constant

Represents a Variant (see page 3430) that has holds a COM compatible string value.

Pascal

```
varStrArg = $0048;
```

C++

```
varStrArg = 0x0048;
```

File

System

Description

In order to use a Variant (see page 3430) with a COM interface, you should create a varStrArg type using VarAsType.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.63 System.varString Constant

Represents a Variant (see page 3430) that holds string data.

Pascal

```
varString = $0100;
```

C++

```
varString = 0x0100;
```

File

System

Description

When a Variant (see page 3430) is assigned a string value, it will be set to a varString Variant (see page 3430) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.64 System.varTypeMask Constant

Defines a mask for filtering out the meta data part of a Variant (see page 3430) data type.

Pascal

```
varTypeMask = $0FFF;
```

C++

```
varTypeMask = 0x0FFF;
```

File

System

Description

The VarType routine returns a TVarType value that describes the current data type used to hold the contents of a Variant (see page 3430) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The high order bits give a meta data type (varArray (see page 3929) and varByRef (see page 3930)). The low order bits determine the basic data type, such as varDouble (see page 3933), varSmallInt. Use varTypeMask to mask out the meta data so as to be able to determine the basic data type of a Variant (see page 3430).

See Also

TVarType

varArray (see page 3929)

VarAsType (see page 5083)

varByRef (see page 3930)

VarType (see page 5101)

38.7.65 System.varUInt64 Constant

Pascal

```
varUInt64 = $0015;
```

C++

```
varUInt64 = 0x0015;
```

File

System

Description

This is constant System.varUInt64.

38.7.66 System.varUnknown Constant

Represents a Variant (see page 3430) that holds an unknown value.

Pascal

```
varUnknown = $000D;
```

C++

```
varUnknown = 0x000D;
```

File

System

Description

When a Variant (see page 3430) is assignned a value that is not recognised, it is set to the varUnknown type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.67 System.varUString Constant

Pascal

```
varUString = $0102;
```

C++

```
varUString = 0x0102;
```

File

System

Description

This is constant System.varUString.

38.7.68 System.varVariant Constant

Represents a Variant (see page 3430) data type.

Pascal

```
varVariant = $000C;
```

C++

```
varVariant = 0x000C;
```

File

System

Description

varVariant is principally used when using VarArrayCreate to create an array of Variants (see page 4997). It determines that such an arrray has unspecified Variant (see page 3430) types. Use such an array when element contents may differ.

See Also

TVarType

VarArrayCreate (see page 5077)

VarType (see page 5101)

38.7.69 System.varWord Constant

Represents a Variant (see page 3430) that holds Word data.

Pascal

```
varWord = $0012;
```

C++

```
varWord = 0x0012;
```

File

System

Description

When a Variant (see page 3430) is assigned an integer value greater in value than a Byte type can hold, but less than an LongWord type can hold, it will be set to a varWord Variant (see page 3430) type. varWord represents a Variant (see page 3430) variable that currently holds a Word data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 5083)

VarType (see page 5101)

38.7.70 System.vmtAddRef Constant

Pascal

```
vmtAddRef = 4 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtAddRef = 4 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtAddRef.

38.7.71 System.vmtAfterConstruction Constant

Pascal

```
vmtAfterConstruction = -28 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtAfterConstruction = -28 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtAfterConstruction.

38.7.72 System.vmtAutoTable Constant

Pascal

```
vmtAutoTable = -80;
```

C++

```
vmtAutoTable = -80;
```

File

System

Description

This is constant System.vmtAutoTable.

38.7.73 System.vmtBeforeDestruction Constant

Pascal

```
vmtBeforeDestruction = -24 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtBeforeDestruction = -24 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtBeforeDestruction.

38.7.74 System.vmtClassName Constant

Pascal

```
vmtClassName = -56;
```

C++

```
vmtClassName = -56;
```

File

System

Description

This is constant System.vmtClassName.

38.7.75 System.vmtCreateObject Constant

Pascal

```
vmtCreateObject = 12 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtCreateObject = 12 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtCreateObject.

38.7.76 System.vmtDefaultHandler Constant

Pascal

```
vmtDefaultHandler = -16 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDefaultHandler = -16 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDefaultHandler.

38.7.77 System.vmtDestroy Constant

Pascal

```
vmtDestroy = -4 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDestroy = -4 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDestroy.

38.7.78 System.vmtDispatch Constant

Pascal

```
vmtDispatch = -20 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDispatch = -20 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDispatch.

38.7.79 System.vmtDynamicTable Constant

Pascal

```
vmtDynamicTable = -60;
```

C++

```
vmtDynamicTable = -60;
```

File

System

Description

This is constant System.vmtDynamicTable.

38.7.80 System.vmtEquals Constant

Pascal

```
vmtEquals = -44 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtEquals = -44 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtEquals.

38.7.81 System.vmtFieldTable Constant

Pascal

```
vmtFieldTable = -68;
```

C++

```
vmtFieldTable = -68;
```

File

System

Description

This is constant System.vmtFieldTable.

38.7.82 System.vmtFreeInstance Constant

Pascal

```
vmtFreeInstance = -8 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtFreeInstance = -8 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtFreeInstance.

38.7.83 System.vmtGetHashCode Constant

Pascal

```
vmtGetHashCode = -40 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtGetHashCode = -40 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtGetHashCode.

38.7.84 System.vmtInitTable Constant

Pascal

```
vmtInitTable = -76;
```

C++

```
vmtInitTable = -76;
```

File

System

Description

This is constant System.vmtInitTable.

38.7.85 System.vmtInstanceSize Constant

Pascal

```
vmtInstanceSize = -52;
```

C++

```
vmtInstanceSize = -52;
```

File

System

Description

This is constant System.vmtInstanceSize.

38.7.86 System.vmtIntfTable Constant

Pascal

```
vmtIntfTable = -84;
```

C++

```
vmtIntfTable = -84;
```

File

System

Description

This is constant System.vmtIntfTable.

38.7.87 System.vmtMethodTable Constant

Pascal

```
vmtMethodTable = -64;
```

C++

```
vmtMethodTable = -64;
```

File

System

Description

This is constant System.vmtMethodTable.

38.7.88 System.vmtNewInstance Constant

Pascal

```
vmtNewInstance = -12 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtNewInstance = -12 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtNewInstance.

38.7.89 System.vmtParent Constant

Pascal

```
vmtParent = -48;
```

C++

```
vmtParent = -48;
```

File

System

Description

This is constant System.vmtParent.

38.7.90 System.vmtQueryInterface Constant

Pascal

```
vmtQueryInterface = 0 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtQueryInterface = 0 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtQueryInterface.

38.7.91 System.vmtRelease Constant

Pascal

```
vmtRelease = 8 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtRelease = 8 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtRelease.

38.7.92 System.vmtSafeCallException Constant

Pascal

```
vmtSafeCallException = -32 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtSafeCallException = -32 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtSafeCallException.

38.7.93 System.vmtSelfPtr Constant

Pascal

```
vmtSelfPtr = -88;
```

C++

```
vmtSelfPtr = -88;
```

File

System

Description

This is constant System.vmtSelfPtr.

38.7.94 System.vmtToString Constant

Pascal

```
vmtToString = -36 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtToString = -36 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtToString.

38.7.95 System.vmtTypeInfo Constant

Pascal

```
vmtTypeInfo = -72;
```

C++

```
vmtTypeInfo = -72;
```

File

System

Description

This is constant System.vmtTypeInfo.

38.7.96 System.vtAnsiString Constant

vtAnsiString defines an AnsiString TVarData (see page 3813) VType value.

Pascal

```
vtAnsiString = 11;
```

C++

```
vtAnsiString = 11;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.97 System.vtBoolean Constant

vtBoolean defines a Boolean TVarData (see page 3813) VType value.

Pascal

```
vtBoolean = 1;
```

C++

```
vtBoolean = 1;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.98 System.vtChar Constant

vtChar defines a Char TVarData (see page 3813) VType value.

Pascal

```
vtChar = 2;
```

C++

```
vtChar = 2;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.99 System.vtClass Constant

vtClass defines a TClass (see page 3856) TVarData (see page 3813) VType value.

Pascal

```
vtClass = 8;
```

C++

```
vtClass = 8;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.100 System.vtCurrency Constant

vtCurrency defines a Currency (see page 3242) TVarData (see page 3813) VType value.

Pascal

```
vtCurrency = 12;
```

C++

```
vtCurrency = 12;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.101 System.vtExtended Constant

vtExtended defines an Extended TVarData (see page 3813) VType value.

Pascal

```
vtExtended = 3;
```

C++

```
vtExtended = 3;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.102 System.vtInt64 Constant

vtInt64 defines an Int64 TVarData (see page 3813) VType value.

Pascal

```
vtInt64 = 16;
```

C++

```
vtInt64 = 16;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.103 System.vtInteger Constant

vtInteger defines an Integer TVarData (see page 3813) VType value.

Pascal

```
vtInteger = 0;
```

C++

```
vtInteger = 0;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.104 System.vtInterface Constant

vtInterface defines an Interface pointer TVarData (see page 3813) VType value.

Pascal

```
vtInterface = 14;
```

C++

```
vtInterface = 14;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.105 System.vtObject Constant

vtObject defines a TObject (see page 3364) TVarData (see page 3813) VType value.

Pascal

```
vtObject = 7;
```

C++

```
vtObject = 7;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.106 System.vtPChar Constant

vtPChar defines a PChar TVarData (see page 3813) VType value.

Pascal

```
vtPChar = 6;
```

C++

```
vtPChar = 6;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.107 System.vtPointer Constant

vtPointer defines a Pointer TVarData (see page 3813) VType value.

Pascal

```
vtPointer = 5;
```

C++

```
vtPointer = 5;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.108 System.vtPWideChar Constant

vtPWideChar defines a PWideChar TVarData (see page 3813) VType value.

Pascal

```
vtPWideChar = 10;
```

C++

```
vtPWideChar = 10;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.109 System.vtString Constant

vtString defines a Variant (see page 3430) TVarData (see page 3813) VType value.

Pascal

```
vtString = 4;
```

C++

```
vtString = 4;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.110 System.vtUnicodeString Constant

Pascal

```
vtUnicodeString = 17;
```

C++

```
vtUnicodeString = 17;
```

File

System

Description

This is constant System.vtUnicodeString.

38.7.111 System.vtVariant Constant

Pascal

```
vtVariant = 13;
```

C++

```
vtVariant = 13;
```

File

System

Description

This is constant System.vtVariant.

38.7.112 System.vtWideChar Constant

vtWideChar defines a WideChar TVarData (see page 3813) VType value.

Pascal

```
vtWideChar = 9;
```

C++

```
vtWideChar = 9;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.7.113 System.vtWideString Constant

vtWideString defines a WideString (see page 3476) TVarData (see page 3813) VType value.

Pascal

```
vtWideString = 15;
```

C++

```
vtWideString = 15;
```

File

System

Description

The TVarData (see page 3813) type represents the internal structure of a Variant (see page 3430) variable (on Windows, this is identical to the Variant (see page 3430) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3430) it holds.

See Also

TVarData (see page 3813)

TVarRec (see page 3819)

38.8 Macros

The following table lists macros in this documentation.

Macros

Name	Description
USE (see page 3960)	This is macro System::USE.
USEADDITIONALFILES (see page 3960)	This is macro System::USEADDITIONALFILES.
USEASM (see page 3961)	This is macro System::USEASM.
USEDATAMODULE (see page 3961)	This is macro System::USEDATAMODULE.
USEDATAMODULENS (see page 3961)	This is macro System::USEDATAMODULENS.
USEDEF (see page 3962)	This is macro System::USEDEF.
USEFILE (see page 3962)	This is macro System::USEFILE.
USEFORM (see page 3963)	This is macro System::USEFORM.
USEFORMNS (see page 3963)	This is macro System::USEFORMNS.
USEFORMRES (see page 3963)	This is macro System::USEFORMRES.
USEIDL (see page 3964)	This is macro System::USEIDL.
USELIB (see page 3964)	This is macro System::USELIB.
USEOBJ (see page 3964)	This is macro System::USEOBJ.
USEPACKAGE (see page 3965)	This is macro System::USEPACKAGE.
USERC (see page 3965)	This is macro System::USERC.
USERES (see page 3965)	This is macro System::USERES.
USETLB (see page 3966)	This is macro System::USETLB.
USEUNIT (see page 3966)	This is macro System::USEUNIT.

38.8.1 System::USE Macro

Pascal

USE

C++

```
#define USE(FileName, ContainerId) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USE.

38.8.2 System::USEADDITIONALFILES Macro

Pascal

USEADDITIONALFILES

C++

```
#define USEADDITIONALFILES(Exts) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEADDITIONALFILES.

38.8.3 System::USEASM Macro

Pascal

USEASM

C++

```
#define USEASM(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEASM.

38.8.4 System::USEDATAMODULE Macro

Pascal

USEDATAMODULE

C++

```
#define USEDATAMODULE(FileName, DataModuleName) \
    class DELPHICLASS T##DataModuleName;       \
    extern PACKAGE T##DataModuleName *DataModuleName;
```

File

sysclass.h

Description

This is macro System::USEDATAMODULE.

38.8.5 System::USEDATAMODULENS Macro

Pascal

USEDATAMODULENS

C++

```
#define USEDATAMODULENS(FileName, UnitName, DataModuleName) \
    namespace UnitName {          \
    class DELPHICLASS T##DataModuleName;       \
    extern PACKAGE T##DataModuleName *DataModuleName;              \
    };                \
    using namespace UnitName
```

File

sysclass.h

Description

This is macro System::USEDATAMODULENS.

38.8.6 System::USEDEF Macro

Pascal

```
USEDEF
```

C++

```
#define USEDEF(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEDEF.

38.8.7 System::USEFILE Macro

Pascal

```
USEFILE
```

C++

```
#define USEFILE(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEFILE.

38.8.8 System::USEFORM Macro

Pascal

```
USEFORM
```

C++

```
#define USEFORM(FileName, FormName) \
    class DELPHICLASS T##FormName;       \
    extern PACKAGE T##FormName *FormName;
```

File

sysclass.h

Description

This is macro System::USEFORM.

38.8.9 System::USEFORMNS Macro

Pascal

```
USEFORMNS
```

C++

```
#define USEFORMNS(FileName, UnitName, FormName) \
    namespace UnitName {         \
    class DELPHICLASS T##FormName;       \
    extern PACKAGE T##FormName *FormName;          \
    };               \
    using namespace UnitName
```

File

sysclass.h

Description

This is macro System::USEFORMNS.

38.8.10 System::USEFORMRES Macro

Pascal

```
USEFORMRES
```

C++

```
#define USEFORMRES(FileName, FormName, AncestorName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEFORMRES.

38.8.11 System::USEIDL Macro

Pascal

```
USEIDL
```

C++

```
#define USEIDL(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEIDL.

38.8.12 System::USELIB Macro

Pascal

```
USELIB
```

C++

```
#define USELIB(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USELIB.

38.8.13 System::USEOBJ Macro

Pascal

```
USEOBJ
```

C++

```
#define USEOBJ(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEOBJ.

38.8.14 System::USEPACKAGE Macro

Pascal

```
USEPACKAGE
```

C++

```
#define USEPACKAGE(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEPACKAGE.

38.8.15 System::USERC Macro

Pascal

```
USERC
```

C++

```
#define USERC(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USERC.

38.8.16 System::USERES Macro

Pascal

```
USERES
```

C++

```
#define USERES(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USERES.

38.8.17 System::USETLB Macro

Pascal

```
USETLB
```

C++

```
#define USETLB(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USETLB.

38.8.18 System::USEUNIT Macro

Pascal

```
USEUNIT
```

C++

```
#define USEUNIT(ModName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEUNIT.

39 SysUtils Namespace

This is namespace SysUtils.

Topics

Name	Description
Classes (see page 3981)	The following table lists classes in this documentation.
Interfaces (see page 4101)	The following table lists interfaces in this documentation.
Functions (see page 4103)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4361)	The following table lists structs, records, enums in this documentation.
Types (see page 4377)	The following table lists types in this documentation.
Variables (see page 4387)	The following table lists variables in this documentation.
Constants (see page 4408)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAbort (see page 3982)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 3982)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 3983)	EAccessViolation is the exception class for invalid memory access errors.
	EArgumentException (see page 3984)	This is class SysUtils.EArgumentException.
	EArgumentOutOfRangeException (see page 3984)	This is class SysUtils.EArgumentOutOfRangeException.
	EAssertionFailed (see page 3984)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 3985)	EControlC is the exception class for user interrupts.
	EConvertError (see page 3985)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 3987)	EDivByZero is the exception class for integer divide-by-zero errors.
	EEncodingError (see page 3988)	This is class SysUtils.EEncodingError.
	EExternal (see page 3988)	EExternal is the exception class that captures external error information.
	EExternalException (see page 3991)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 3992)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 3995)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 3999)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 3999)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 4000)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 4000)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 4001)	This is class SysUtils.EInvalidContainer.

	EInvalidInsert (see page 4001)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 4002)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 4002)	EInvalidPointer is the exception class for invalid pointer operations.
	EMathError (see page 4003)	EMathError is the base exception class for floating-point math errors.
	EMonitor (see page 4004)	This is class SysUtils.EMonitor.
	EMonitorLockException (see page 4004)	This is class SysUtils.EMonitorLockException.
	ENoConstructException (see page 4004)	This is class SysUtils.ENoConstructException.
	ENoMonitorSupportException (see page 4005)	This is class SysUtils.ENoMonitorSupportException.
	EOSError (see page 4005)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 4008)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 4008)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 4009)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 4009)	EPrivilege is the exception class for processor privilege violations.
	EProgrammerNotFound (see page 4010)	This is class SysUtils.EProgrammerNotFound.
	EPropReadOnly (see page 4010)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 4011)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 4011)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 4012)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 4013)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 4013)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.
	EVariantError (see page 4014)	EVariantError is the exception class for errors involving Variant (see page 3430) data types.
	EWin32Error (see page 4015)	EWin32Error is the exception class for Windows errors.
	Exception (see page 4015)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 4032)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TBigEndianUnicodeEncoding (see page 4033)	This is class SysUtils.TBigEndianUnicodeEncoding.
	TEncoding (see page 4035)	TEncoding is a class that defines a specific encoding for a character or string.
	TLanguages (see page 4051)	TLanguages lists all available Windows locales.
	TMBCSEncoding (see page 4056)	This is class SysUtils.TMBCSEncoding.
	TMultiReadExclusiveWriteSynchronizer (see page 4061)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 4066)	This is class SysUtils.TSimpleRWSync.
	TStringBuilder (see page 4070)	An editable array of characters whose contents has all the functionality of a string object.
	TThreadLocalCounter (see page 4087)	This is class SysUtils.TThreadLocalCounter.
	TUnicodeEncoding (see page 4088)	This is class SysUtils.TUnicodeEncoding.
	TUTF7Encoding (see page 4093)	This is class SysUtils.TUTF7Encoding.
	TUTF8Encoding (see page 4098)	This is class SysUtils.TUTF8Encoding.

Constants

Name	Description
DateDelta (see page 4409)	Specifies the difference between TDateTime (see page 3335) and TTimeStamp (see page 4375) starting values.
DefaultFalseBoolStr (see page 4410)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 4410)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 4411)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 4411)	Specifies any file type.
faArchive (see page 4412)	Specifies only Windows archived file types.
faDirectory (see page 4412)	Specifies only directory 'file' types.
faHidden (see page 4413)	Specifies only hidden file types.
faNormal (see page 4414)	This is constant SysUtils.faNormal.

faReadOnly (see page 4414)	Specifies only read only file types.
faSymLink (see page 4415)	Specifies only symbolic link file types.
faSysFile (see page 4415)	Specifies only system file types.
faTemporary (see page 4416)	This is constant SysUtils.faTemporary.
faVolumeID (see page 4416)	Specifies Volume ID file types.
fmOpenRead (see page 4417)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 4417)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 4418)	Specifies a file should be opened for writing only.
fmShareCompat (see page 4419)	This is constant SysUtils.fmShareCompat.
fmShareDenyNone (see page 4419)	This is constant SysUtils.fmShareDenyNone.
fmShareDenyRead (see page 4419)	This is constant SysUtils.fmShareDenyRead.
fmShareDenyWrite (see page 4420)	This is constant SysUtils.fmShareDenyWrite.
fmShareExclusive (see page 4420)	This is constant SysUtils.fmShareExclusive.
HoursPerDay (see page 4420)	Specifies the number of hours in a day.
MaxCurrency (see page 4421)	Defines the largest value supported for Currency (see page 3242) data types.
MaxDateTime (see page 4421)	Specifies the highest supported TDateTime (see page 3335) value.
MaxEraCount (see page 4422)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 4422)	Defines the largest negative value supported for Currency (see page 3242) data types.
MinDateTime (see page 4423)	Specifies the earliest supported TDateTime (see page 3335) value.
MinsPerDay (see page 4423)	Specifies the number of minutes in a day.
MinsPerHour (see page 4424)	Specifies the number of minutes in an hour.
MonthDays (see page 4424)	An array specifying how many days are in a month.
MSecsPerDay (see page 4425)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 4425)	Specifies the number of milliseconds in a second.
PathDelim (see page 4426)	Specifies a path delimiter character.
PathSep (see page 4426)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 4427)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 4427)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 4428)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 4428)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 4428)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 4429)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 4429)	Indicates that a package was compiled with {$DESIGNONLY ON}.
pfExeModule (see page 4430)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 4430)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 4431)	Indicates that a module is a library.
pfModuleTypeMask (see page 4431)	Indicates that a package is a module.
pfNeverBuild (see page 4432)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 4432)	Indicates that a module is a package.
pfProducerMask (see page 4433)	Indicates the producer of a package.
pfProducerUndefined (see page 4433)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 4434)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 4434)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 4435)	Specifies the number of seconds in a day.
SecsPerHour (see page 4435)	This is constant SysUtils.SecsPerHour.
SecsPerMin (see page 4436)	Specifies the number of seconds in a minute.
SwitchChars (see page 4436)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 4437)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 4437)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 4438)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 4438)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 4439)	This is constant SysUtils.ufWeakPackageUnit.
ufWeakUnit (see page 4439)	Indicates that a unit is a weak unit.

UnixDateDelta (see page 4439)	Specifies the difference between TDateTime (see page 3335) and Lunix date/time origins.

Functions

	Name	Description
	Abort (see page 4111)	Raises a silent exception.
	AddExitProc (see page 4112)	Adds a procedure to the exit list.
	AddTerminateProc (see page 4112)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 4113)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 4113)	Compares file names based on the current locale.
	AnsiCompareStr (see page 4114)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 4115)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 4115)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 4116)	Converts a quoted string into an unquoted string.
	AnsiExtractQuotedStr (see page 4116)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 4116)	Returns a pointer to the last character in a string.
	AnsiLastChar (see page 4116)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 4117)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 4117)	Converts a file name to lower case.
	AnsiPos (see page 4118)	Locates the position of a substring.
	AnsiQuotedStr (see page 4118)	Returns the quoted version of a string.
	AnsiSameStr (see page 4119)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 4119)	Compares strings based on the current locale without case sensitivity.
	AnsiStrAlloc (see page 4120)	This is function SysUtils.AnsiStrAlloc.
	AnsiStrComp (see page 4120)	Compares null-terminated character strings.
	AnsiStrComp (see page 4120)	Compares null-terminated character strings.
	AnsiStrIComp (see page 4121)	Compares null terminated character strings case insensitively.
	AnsiStrIComp (see page 4121)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 4121)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLastChar (see page 4121)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 4122)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLComp (see page 4122)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 4122)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLIComp (see page 4122)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 4123)	Converts all characters in a null-terminated string to lower case.
	AnsiStrLower (see page 4123)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 4124)	Returns a pointer to the first occurrence of a substring.
	AnsiStrPos (see page 4124)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 4124)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrRScan (see page 4124)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 4125)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrScan (see page 4125)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 4125)	Converts all characters in a null-terminated string to upper case.
	AnsiStrUpper (see page 4125)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 4126)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 4126)	Converts a file name to upper case.
	AppendStr (see page 4127)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 4127)	Assigns a new dynamically allocated string to a specified pointer.
	Beep (see page 4127)	Emits a message beep.
	BoolToStr (see page 4130)	Converts a Boolean value to a string.
	ByteLength (see page 4132)	This is function SysUtils.ByteLength.
	BytesOf (see page 4132)	This is function SysUtils.BytesOf.
	BytesOf (see page 4133)	This is function SysUtils.BytesOf.

	BytesOf (see page 4133)	This is function SysUtils.BytesOf.
	BytesOf (see page 4133)	This is function SysUtils.BytesOf.
	ByteToCharIndex (see page 4134)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharIndex (see page 4134)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 4134)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteToCharLen (see page 4134)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 4135)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	ByteType (see page 4135)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 4136)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 4136)	Changes the extension of a file name.
	ChangeFilePath (see page 4137)	Changes the path of a file name.
	CharInSet (see page 4138)	This is function SysUtils.CharInSet.
	CharInSet (see page 4138)	This is function SysUtils.CharInSet.
	CharLength (see page 4139)	Returns number of bytes used by a character.
	CharLength (see page 4139)	Returns number of bytes used by a character.
	CharToByteIndex (see page 4139)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteIndex (see page 4139)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 4140)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToByteLen (see page 4140)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToElementIndex (see page 4140)	This is function SysUtils.CharToElementIndex.
	CharToElementIndex (see page 4140)	This is function SysUtils.CharToElementIndex.
	CharToElementLen (see page 4141)	This is function SysUtils.CharToElementLen.
	CharToElementLen (see page 4141)	This is function SysUtils.CharToElementLen.
	CheckWin32Version (see page 4141)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3165).
	CompareMem (see page 4142)	Performs a binary comparison of two memory images.
	CompareStr (see page 4142)	Compares two strings case sensitively.
	CompareStr (see page 4142)	Compares two strings case sensitively.
	CompareText (see page 4143)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 4143)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 4147)	Creates a new directory.
	CreateGUID (see page 4148)	Creates a globally unique identifier.
	CurrentYear (see page 4150)	Returns the current year.
	CurrToStr (see page 4150)	Formats a Currency (see page 3242) value as a string.
	CurrToStr (see page 4150)	Formats a Currency (see page 3242) value as a string.
	CurrToStrF (see page 4151)	Converts a Currency (see page 3242) value to a string, using a specified format.
	CurrToStrF (see page 4151)	Converts a Currency (see page 3242) value to a string, using a specified format.
	Date (see page 4151)	Returns the current date.
	DateTimeToFileDate (see page 4152)	Converts a TDateTime (see page 3335) value to an OS timestamp value.
	DateTimeToStr (see page 4153)	Converts a TDateTime (see page 3335) value to a string.
	DateTimeToStr (see page 4153)	Converts a TDateTime (see page 3335) value to a string.
	DateTimeToString (see page 4153)	Converts a TDateTime (see page 3335) value to a string using a specified Format (see page 4232).
	DateTimeToString (see page 4153)	Converts a TDateTime (see page 3335) value to a string using a specified Format (see page 4232).
	DateTimeToSystemTime (see page 4155)	Converts a TDateTime (see page 3335) value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 4156)	Converts a TDateTime (see page 3335) value into the corresponding TTimeStamp (see page 4375) value.
	DateToStr (see page 4156)	Converts a TDateTime (see page 3335) value to a string.
	DateToStr (see page 4156)	Converts a TDateTime (see page 3335) value to a string.
	DayOfWeek (see page 4158)	Returns the day of the week for a specified date.

	DecodeDate (see page 4159)	Returns Year, Month, and Day values for a TDateTime (see page 3335) value.
	DecodeDateFully (see page 4160)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3335) value.
	DecodeTime (see page 4161)	Breaks a TDateTime (see page 3335) value into hours, minutes, seconds, and milliseconds.
	DelegatesEqual (see page 4162)	This is function SysUtils.DelegatesEqual.
	DeleteFile (see page 4162)	Deletes a file from disk.
	DirectoryExists (see page 4164)	Determines whether a specified directory exists.
	DiskFree (see page 4165)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 4166)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 4167)	Disposes of a string pointer that was allocated using NewStr (see page 4271).
	ElementToCharIndex (see page 4168)	This is function SysUtils.ElementToCharIndex.
	ElementToCharIndex (see page 4168)	This is function SysUtils.ElementToCharIndex.
	ElementToCharLen (see page 4168)	This is function SysUtils.ElementToCharLen.
	ElementToCharLen (see page 4169)	This is function SysUtils.ElementToCharLen.
	EncodeDate (see page 4169)	Returns a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.
	EncodeTime (see page 4170)	Returns a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 4171)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 4172)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 4173)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 4173)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 4174)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 4175)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 4176)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 4176)	Returns the drive portion of a file name.
	ExtractFileExt (see page 4177)	Returns the extension portions of a file name.
	ExtractFileName (see page 4177)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 4179)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 4181)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 4182)	Converts a file name to the short 8.3 form.
	FileAge (see page 4183)	Returns the OS timestamp of a file.
	FileAge (see page 4183)	Returns the OS timestamp of a file.
	FileClose (see page 4185)	Closes a specified file.
	FileCreate (see page 4188)	Creates a new file.
	FileCreate (see page 4188)	Creates a new file.
	FileDateToDateTime (see page 4191)	Converts an OS timestamp value to TDateTime (see page 3335) value.
	FileExists (see page 4193)	Tests if a specified file exists.
	FileGetAttr (see page 4197)	Returns the file attributes of FileName.
	FileGetDate (see page 4199)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 4200)	Reports whether a file is read-only.
	FileOpen (see page 4200)	Opens a specified file using a specified access mode.
	FileRead (see page 4204)	Reads a specified number of bytes from a file.
	FileSearch (see page 4206)	Searches a specified directory path for a file.
	FileSeek (see page 4207)	Repositions read/write point.
	FileSeek (see page 4207)	Repositions read/write point.
	FileSetAttr (see page 4209)	Sets the file attributes of a specified file.
	FileSetDate (see page 4212)	Sets the OS time stamp for a specified file.
	FileSetDate (see page 4212)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 4212)	Sets the file permissions to read-only.
	FileWrite (see page 4213)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 4216)	Finalizes the specified package (.bpl file).
	FindClose (see page 4216)	Releases memory allocated by FindFirst (see page 4218).
	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.

	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 4218)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 4221)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4218).
	FloatToCurr (see page 4223)	Converts a floating-point value to a Currency (see page 3242) value.
	FloatToDateTime (see page 4223)	Converts a floating-point value to a TDateTime (see page 3335) value.
	FloatToDecimal (see page 4224)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 4224)	Converts a floating point value to a string.
	FloatToStr (see page 4224)	Converts a floating point value to a string.
	FloatToStrF (see page 4226)	Converts a floating point value to a string, using a specified Format (see page 4232), Precision, and Digits.
	FloatToStrF (see page 4226)	Converts a floating point value to a string, using a specified Format (see page 4232), Precision, and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 4229)	Returns formatted output using a resourced format string.
	FmtStr (see page 4229)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 4229)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 4230)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 4232)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 4232)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 4238)	Formats a Currency (see page 3242) object.
	FormatCurr (see page 4238)	Formats a Currency (see page 3242) object.
	FormatDateTime (see page 4239)	Formats a TDateTime (see page 3335) value.
	FormatDateTime (see page 4239)	Formats a TDateTime (see page 3335) value.
	FormatFloat (see page 4241)	Formats a floating point value.
	FormatFloat (see page 4241)	Formats a floating point value.
	FreeAndNil (see page 4245)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 4246)	Get Delphi Access Licence.
	GetCurrentDir (see page 4246)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 4248)	Retrieves an environment variable value.
	GetFileVersion (see page 4249)	Returns file version.

	GetFormatSettings (see page 4250)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 4250)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 4250)	Populates a TFormatSettings (see page 4367) data structure.
	GetLocaleStr (see page 4251)	Retrieves a localisation value.
	GetModuleName (see page 4251)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 4252)	Returns the description resource from a package.
	GetPackageInfo (see page 4253)	Enumerates all the units and required packages for a package.
	GetTime (see page 4253)	Returns the current time.
	GUIDToString (see page 4254)	Converts a class identifier GUID to a string.
	HashName (see page 4256)	This is function SysUtils.HashName.
	IncAMonth (see page 4256)	Increments date data by one month.
	IncludeTrailingBackslash (see page 4257)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 4257)	Ensures path name ends with delimiter.
	IncMonth (see page 4258)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 4259)	Validates and initializes a given package.
	InitializePackage (see page 4259)	Validates and initializes a given package.
	IntToHex (see page 4259)	Returns the hex representation of an integer.
	IntToHex (see page 4259)	Returns the hex representation of an integer.
	IntToStr (see page 4260)	Converts an integer to a string.
	IntToStr (see page 4260)	Converts an integer to a string.
	IsAssembly (see page 4261)	Determines whether a dll is an Assembly.
	IsDelimiter (see page 4262)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 4262)	Indicates whether two TGUID (see page 3800) values are the same.
	IsLeadChar (see page 4264)	This is function SysUtils.IsLeadChar.
	IsLeadChar (see page 4264)	This is function SysUtils.IsLeadChar.
	IsLeapYear (see page 4264)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 4266)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 4266)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 4267)	Lists the locales for which support is available.
	LastDelimiter (see page 4267)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 4268)	Loads a specified package.
	LoadPackage (see page 4268)	Loads a specified package.
	LoadStr (see page 4268)	Loads a string resource from the application's executable file.
	LowerCase (see page 4269)	Converts an ASCII string to lowercase.
	LowerCase (see page 4269)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 4270)	Converts a specified number of milliseconds into a TTimeStamp (see page 4375) value.
	NewStr (see page 4271)	Allocates a string on the heap.
	NextCharIndex (see page 4271)	Returns the byte index of the next character.
	NextCharIndex (see page 4271)	Returns the byte index of the next character.
	Now (see page 4271)	Returns the current date and time.
	OutOfMemoryError (see page 4272)	Raises an EOutOfMemory (see page 4008) exception.
	PlatformBytesOf (see page 4273)	This is function SysUtils.PlatformBytesOf.
	PlatformStringOf (see page 4273)	This is function SysUtils.PlatformStringOf.
	QuotedStr (see page 4273)	Returns the quoted version of a string.
	RaiseLastOSError (see page 4274)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 4274)	Raises an exception for the last occurring OS or system library error.
	RaiseLastWin32Error (see page 4275)	Raises an exception for the last occurring Win32 error.
	RCS (see page 4276)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 4276)	Deletes an existing empty directory.
	RenameFile (see page 4277)	Changes a file name.
	ReplaceDate (see page 4279)	Replaces the date portion of a TDateTime (see page 3335) value with a specified date.

	ReplaceTime (see page 4280)	Replaces the time portion of a TDateTime (see page 3335) value with a specified time.
	RPR (see page 4281)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 4281)	Safe and simple library loading.
	SameFileName (see page 4282)	Compares file names based on the current locale.
	SameStr (see page 4282)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 4282)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 4283)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 4283)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 4283)	Sets the current directory.
	ShowException (see page 4284)	Displays an exception message with its physical address
	Sleep (see page 4284)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 4285)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 4285)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4285).
	StrBufSize (see page 4285)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4285).
	StrByteType (see page 4286)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrByteType (see page 4286)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 4286)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCat (see page 4286)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 4288)	Returns the length, in bytes, of a character.
	StrCharLength (see page 4288)	Returns the length, in bytes, of a character.
	StrComp (see page 4288)	Compares two strings with case sensitivity.
	StrComp (see page 4288)	Compares two strings with case sensitivity.
	StrCopy (see page 4289)	Copies a null-terminated string.
	StrCopy (see page 4289)	Copies a null-terminated string.
	StrDispose (see page 4291)	Disposes of a string.
	StrDispose (see page 4291)	Disposes of a string.
	StrECopy (see page 4292)	Copies null-terminated string.
	StrECopy (see page 4292)	Copies null-terminated string.
	StrEnd (see page 4293)	Returns a pointer to the end of a null terminated string.
	StrEnd (see page 4293)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrIComp (see page 4295)	Compares two strings without case sensitivity.
	StrIComp (see page 4295)	Compares two strings without case sensitivity.
	StringOf (see page 4296)	This is function SysUtils.StringOf.
	StringReplace (see page 4297)	Returns a string with occurrences of one substring replaced by another substring.
	StringToGUID (see page 4297)	Converts a string to a GUID.
	StrLCat (see page 4299)	Appends up to a specified maximum number of characters to string.
	StrLCat (see page 4299)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 4300)	Compares up to a specified maximum number of characters in two strings.
	StrLComp (see page 4300)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 4302)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLCopy (see page 4302)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 4303)	Returns number of characters in a string excluding the null terminator.
	StrLen (see page 4303)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.

	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 4305)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLIComp (see page 4305)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 4307)	Converts a string to lowercase.
	StrLower (see page 4307)	Converts a string to lowercase.
	StrMove (see page 4308)	Copies specified number of characters to string.
	StrMove (see page 4308)	Copies specified number of characters to string.
	StrNew (see page 4310)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNew (see page 4310)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 4311)	Returns pointer to following character.
	StrNextChar (see page 4311)	Returns pointer to following character.
	StrPas (see page 4311)	Converts null-terminated string to an AnsiString (long string).
	StrPas (see page 4311)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 4313)	Copies an AnsiString (long string) to a null-terminated string.
	StrPCopy (see page 4313)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 4314)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPLCopy (see page 4314)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 4314)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrPos (see page 4314)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 4315)	Returns a pointer to the last occurrence of a specified character in a string.
	StrRScan (see page 4315)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 4317)	Returns a pointer to first occurrence of a specified character in a string.
	StrScan (see page 4317)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 4318)	Converts a string to a Boolean value.
	StrToBoolDef (see page 4318)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 4319)	Converts a string to a Currency (see page 3242) value.
	StrToCurr (see page 4319)	Converts a string to a Currency (see page 3242) value.
	StrToCurrDef (see page 4320)	Converts a string to a Currency (see page 3242) value, with error default.
	StrToCurrDef (see page 4320)	Converts a string to a Currency (see page 3242) value, with error default.
	StrToDate (see page 4320)	Converts a string to a TDateTime (see page 3335) value.
	StrToDate (see page 4320)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateDef (see page 4322)	Converts a string to a TDateTime (see page 3335) value, with error default.
	StrToDateDef (see page 4322)	Converts a string to a TDateTime (see page 3335) value, with error default.
	StrToDateTime (see page 4322)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateTime (see page 4322)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateTimeDef (see page 4323)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToDateTimeDef (see page 4323)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToFloat (see page 4324)	Converts a given string to a floating-point value.
	StrToFloat (see page 4324)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 4327)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 4327)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 4328)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 4329)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64Def (see page 4329)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 4330)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 4332)	Converts a string to a TDateTime (see page 3335) value.
	StrToTime (see page 4332)	Converts a string to a TDateTime (see page 3335) value.
	StrToTimeDef (see page 4333)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToTimeDef (see page 4333)	Converts a string to a TDateTime (see page 3335) value with error default.

	StrUpper (see page 4334)	Returns a string in upper case.
	StrUpper (see page 4334)	Returns a string in upper case.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	SysErrorMessage (see page 4336)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 4336)	Converts a system time value into a TDateTime (see page 3335) value.
	TextPos (see page 4337)	This is function SysUtils.TextPos.
	TextPos (see page 4337)	This is function SysUtils.TextPos.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	Time (see page 4338)	Returns the current time.
	TimeStampToDateTime (see page 4339)	Converts a TTimeStamp (see page 4375) value into the corresponding TDateTime (see page 3335) value.
	TimeStampToMSecs (see page 4340)	Converts a TTimeStamp (see page 4375) value into an absolute number of milliseconds.
	TimeToStr (see page 4340)	Returns a string that represents a TDateTime (see page 3335) value.
	TimeToStr (see page 4340)	Returns a string that represents a TDateTime (see page 3335) value.
	Trim (see page 4342)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 4343)	Trims leading spaces and control characters from a string.
	TrimRight (see page 4343)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 4343)	Creates a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 4344)	Creates a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 4345)	Converts a floating-point value to a Currency (see page 3242) value.
	TryFloatToDateTime (see page 4345)	Converts a floating-point value to a TDateTime (see page 3335) value.
	TryStrToBool (see page 4346)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 4347)	Converts a string to a Currency (see page 3242) value, with a Boolean success code.
	TryStrToCurr (see page 4347)	Converts a string to a Currency (see page 3242) value, with a Boolean success code.
	TryStrToDate (see page 4347)	Converts a string to a TDateTime (see page 3335) value, with a Boolean success code.
	TryStrToDate (see page 4347)	Converts a string to a TDateTime (see page 3335) value, with a Boolean success code.
	TryStrToDateTime (see page 4348)	Converts a string to a TDateTime (see page 3335) value with a Boolean success code.
	TryStrToDateTime (see page 4348)	Converts a string to a TDateTime (see page 3335) value with a Boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToInt (see page 4349)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 4350)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 4351)	Converts a string to a TDateTime (see page 3335) value with an error default.
	TryStrToTime (see page 4351)	Converts a string to a TDateTime (see page 3335) value with an error default.
	UIntToStr (see page 4351)	This is function SysUtils.UIntToStr.
	UIntToStr (see page 4352)	This is function SysUtils.UIntToStr.

	UnloadPackage (see page 4352)	Unloads a package.
	UpperCase (see page 4352)	Returns a copy of a string in uppercase.
	UpperCase (see page 4352)	Returns a copy of a string in uppercase.
	WideBytesOf (see page 4354)	This is function SysUtils.WideBytesOf.
	WideCompareStr (see page 4354)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 4355)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 4355)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 4355)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 4356)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 4356)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 4356)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 4356)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 4357)	Returns Unicode string converted to lower case.
	WideSameStr (see page 4357)	Compares Unicode strings based on the current locale with case sensitivity.
	WideSameText (see page 4358)	Compares Unicode strings based on the current locale without case sensitivity.
	WideStrAlloc (see page 4358)	This is function SysUtils.WideStrAlloc.
	WideStringOf (see page 4359)	This is function SysUtils.WideStringOf.
	WideUpperCase (see page 4359)	Returns Unicode string converted to upper case.
	Win32Check (see page 4359)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 4360)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 4360)	Splits a string into multiple lines as its length approaches a specified size.

Interfaces

	Name	Description
	IReadWriteSync (see page 4101)	This is class SysUtils.IReadWriteSync.

Structs, Records, Enums

	Name	Description
	Int64Rec (see page 4361)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 4362)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TCharSearch (see page 4362)	This is record SysUtils.TCharSearch.
	TExceptionRecord (see page 4363)	TExceptionRecord holds information about an exception in an application.
	TFilenameCaseMatch (see page 4364)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 4364)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatRec (see page 4366)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFloatValue (see page 4366)	This is record SysUtils.TFloatValue.
	TFormatSettings (see page 4367)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 4368)	TLangRec declares a record used to describe a Windows locale.
	TLocaleOptions (see page 4369)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 4369)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 4370)	This is record SysUtils.TNameType.
	TSearchRec (see page 4371)	TSearchRec defines file information searched for by FindFirst (see page 4218) or FindNext (see page 4221).
	TSysLocale (see page 4374)	TSysLocale represents locale information.
	TThreadInfo (see page 4375)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 4375)	TTimeStamp represents time and date values.
	WordRec (see page 4376)	WordRec declares a utility record to store high and low order bytes of a variable.

Types

Name	Description
ExceptClass (see page 4377)	ExceptClass declares an Exception (see page 4015) class.
PByteArray (see page 4378)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 4378)	PDayTable is a pointer to a TDayTable (see page 4381) type.
PExceptionRecord (see page 4379)	PExceptionRecord is a pointer to a TExceptionRecord (see page 4363) type.
PThreadInfo (see page 4379)	This is type SysUtils.PThreadInfo.
PWordArray (see page 4379)	PWordArray is a pointer to a TWordArray (see page 4386) type.
TByteArray (see page 4380)	TByteArray declares an array of Bytes.
TBytes (see page 4380)	TBytes declares an array of Bytes.
TCharArray (see page 4381)	This is type SysUtils.TCharArray.
TDayTable (see page 4381)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 4381)	TFileName is a string that represents a file name.
TIntegerSet (see page 4382)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 4382)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 4061).
TPackageInfoProc (see page 4383)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4253).
TProc (see page 4383)	This is type SysUtils.TProc.
TProcedure (see page 4384)	TProcedure declares a procedure.
TReplaceFlags (see page 4384)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 4384)	TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.
TTerminateProc (see page 4385)	TValidatePackageProc (see page 4385) is a call-back function invoked by LoadPackage (see page 4268).
TValidatePackageProc (see page 4385)	This is type SysUtils.TValidatePackageProc.
TWordArray (see page 4386)	TWordArray declares an array of Words.

Variables

Name	Description
CurrencyDecimals (see page 4388)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 4388)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 4389)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 4389)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 4390)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyAnsiStr (see page 4391)	This is variable SysUtils.EmptyAnsiStr.
EmptyStr (see page 4391)	An empty string constant.
EmptyWideStr (see page 4391)	An empty string version of the WideString (see page 3476) constant.
EraNames (see page 4392)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 4392)	Controls the DateTimeToStr (see page 4153) function (for East Asian locales).
FalseBoolStrs (see page 4392)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 4393)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 4393)	HexDisplayPrefix contains the prefix to display on hexadecimal values - $ for Pascal (Delphi) syntax, 0x for C++ syntax.
LeadBytes (see page 4394)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 4394)	The character used to separate items in a list.
LongDateFormat (see page 4394)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 4395)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a long day name to be substituted.

LongMonthNames (see page 4395)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a long month name to be substituted.
LongTimeFormat (see page 4396)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 4396)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullAnsiStr (see page 4397)	This is variable SysUtils.NullAnsiStr.
NullStr (see page 4397)	A pointer to an empty string.
NullWideStr (see page 4398)	A pointer to an empty wide string.
ShortDateFormat (see page 4398)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 4399)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a short day name to be substituted.
ShortMonthNames (see page 4399)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a short month name to be substituted.
ShortTimeFormat (see page 4400)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 4400)	SysLocale holds locale information.
ThousandSeparator (see page 4402)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.
TimeAMString (see page 4402)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.
TimePMString (see page 4402)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.
TimeSeparator (see page 4403)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 4403)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 4404)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 4404)	Specifies which the build level of the Windows Operating System (see page 3165) operating.
Win32CSDVersion (see page 4405)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 4405)	Specifies which major Windows Operating System (see page 3165) version is operating.
Win32MinorVersion (see page 4406)	Specifies which minor Windows Operating System (see page 3165) version is operating.
Win32Platform (see page 4406)	Specifies the Windows Operating System (see page 3165) platform identifier.

39.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAbort (see page 3982)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 3982)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 3983)	EAccessViolation is the exception class for invalid memory access errors.
	EArgumentException (see page 3984)	This is class SysUtils.EArgumentException.
	EArgumentOutOfRangeException (see page 3984)	This is class SysUtils.EArgumentOutOfRangeException.
	EAssertionFailed (see page 3984)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 3985)	EControlC is the exception class for user interrupts.
	EConvertError (see page 3985)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 3987)	EDivByZero is the exception class for integer divide-by-zero errors.
	EEncodingError (see page 3988)	This is class SysUtils.EEncodingError.
	EExternal (see page 3988)	EExternal is the exception class that captures external error information.
	EExternalException (see page 3991)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 3992)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 3995)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 3999)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 3999)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 4000)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 4000)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 4001)	This is class SysUtils.EInvalidContainer.
	EInvalidInsert (see page 4001)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 4002)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 4002)	EInvalidPointer is the exception class for invalid pointer operations.
	EMathError (see page 4003)	EMathError is the base exception class for floating-point math errors.
	EMonitor (see page 4004)	This is class SysUtils.EMonitor.
	EMonitorLockException (see page 4004)	This is class SysUtils.EMonitorLockException.
	ENoConstructException (see page 4004)	This is class SysUtils.ENoConstructException.
	ENoMonitorSupportException (see page 4005)	This is class SysUtils.ENoMonitorSupportException.
	EOSError (see page 4005)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 4008)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 4008)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 4009)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 4009)	EPrivilege is the exception class for processor privilege violations.
	EProgrammerNotFound (see page 4010)	This is class SysUtils.EProgrammerNotFound.
	EPropReadOnly (see page 4010)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 4011)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 4011)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 4012)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 4013)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 4013)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.

	EVariantError (see page 4014)	EVariantError is the exception class for errors involving Variant (see page 3430) data types.
	EWin32Error (see page 4015)	EWin32Error is the exception class for Windows errors.
	Exception (see page 4015)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 4032)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TBigEndianUnicodeEncoding (see page 4033)	This is class SysUtils.TBigEndianUnicodeEncoding.
	TEncoding (see page 4035)	TEncoding is a class that defines a specific encoding for a character or string.
	TLanguages (see page 4051)	TLanguages lists all available Windows locales.
	TMBCSEncoding (see page 4056)	This is class SysUtils.TMBCSEncoding.
	TMultiReadExclusiveWriteSynchronizer (see page 4061)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 4066)	This is class SysUtils.TSimpleRWSync.
	TStringBuilder (see page 4070)	An editable array of characters whose contents has all the functionality of a string object.
	TThreadLocalCounter (see page 4087)	This is class SysUtils.TThreadLocalCounter.
	TUnicodeEncoding (see page 4088)	This is class SysUtils.TUnicodeEncoding.
	TUTF7Encoding (see page 4093)	This is class SysUtils.TUTF7Encoding.
	TUTF8Encoding (see page 4098)	This is class SysUtils.TUTF8Encoding.

39.1.1 EAbort Class

EAbort is the exception class for errors that should not display an error message dialog box.

Class Hierarchy

Pascal

```
EAbort = class(Exception);
```

C++

```
class EAbort : public Exception;
```

File

SysUtils

Description

Use EAbort to raise an exception without displaying an error message in a dialog box. If applications do not trap such "silent" exceptions, the EAbort exception is passed to the HandleException method of the Application object.

The Abort (see page 4111) procedure provides a simple, standard way to raise EAbort.

See Also

Abort (see page 4111)

TApplication

Application

39.1.2 EAbstractError Class

EAbstractError is the exception class for attempts to call an abstract method.

Class Hierarchy

Pascal

```
EAbstractError = class(Exception);
```

C++

```
class EAbstractError : public Exception;
```

File

SysUtils

Description

EAbstractError is raised when an application tries to call a Delphi abstract method. It is also raised at design time when a component with Delphi abstract method is placed on a form.

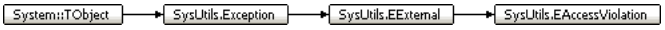
Delphi abstract methods are declared with the abstract directive and must be overridden in descendant classes before an instance object can be created.

C++ code can catch this error, but never throws it.

39.1.3 EAccessViolation Class

EAccessViolation is the exception class for invalid memory access errors.

Class Hierarchy



Pascal

```
EAccessViolation = class(EExternal);
```

C++

```
class EAccessViolation : public EExternal;
```

File

SysUtils

Description

EAccessViolation is raised when an application

Dereferences a nil (Delphi) or NULL (C++) pointer.

Writes to memory reserved for executable code.

Attempts to access a memory address for which there is no virtual memory allocated to the application.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Applications should not raise EAccessViolation directly, but should instead rely on the runtime to raise this exception.

See Also

EExternal (see page 3988)

EOutOfMemory (see page 4008)

EOutOfResources

39.1.4 EArgumentException Class

Class Hierarchy

Pascal

```
EArgumentException = class(Exception);
```

C++

```
class EArgumentException : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EArgumentException.

39.1.5 EArgumentOutOfRangeException Class

Class Hierarchy

Pascal

```
EArgumentOutOfRangeException = class(EArgumentException);
```

C++

```
class EArgumentOutOfRangeException : public EArgumentException;
```

File

SysUtils

Description

This is class SysUtils.EArgumentOutOfRangeException.

39.1.6 EAssertionFailed Class

EAssertionFailed is the exception class for Delphi assertion failure.

Class Hierarchy

Pascal

```
EAssertionFailed = class(Exception);
```

C++

```
class EAssertionFailed : public Exception;
```

File

SysUtils

Description

EAssertionFailed is raised when an assertion fails in Delphi code.

Note: To raise this exception, use the Assert (see page 3627) procedure. The exception is raised only in code compiled with the $ASSERTIONS ON directive.

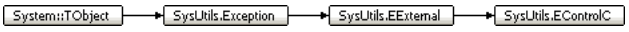
See Also

Assert

39.1.7 EControlC Class

EControlC is the exception class for user interrupts.

Class Hierarchy



Pascal

```
EControlC = class(EExternal);
```

C++

```
class EControlC : public EExternal;
```

File

SysUtils

Description

EControlC is raised when the application is interrupted by the user.

Note: On Windows, a user interrupt can only occur in console applications, when the user presses Ctrl+C.

Note: On Linux, EControlC is raised when the program receives a TERM (15) signal. The TERM signal can be generated by the user typing the interrupt character (normally Ctrl+C) or by another program using the signal system call.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

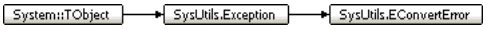
See Also

EExternal (see page 3988)

39.1.8 EConvertError Class

EConvertError is the exception class for string and object conversion errors.

Class Hierarchy



Pascal

```
EConvertError = class(Exception);
```

C++

```
class EConvertError : public Exception;
```

File

SysUtils

Description

EConvertError is raised when

An application makes an unsuccessful attempt to convert an integer, float, date, or time to a string, or to convert a string to one of these other types.

An application passes an invalid argument to a formatting routine.

An application attempts to assign one type of component derived from TPersistent to another component derived from TPersistent when such an assignment is not possible. For example, EConvertError is raised by the attempted assignment of a TButton control to a TEdit control.

In the examples below, an EConvertError exception is raised on the attempt to convert a String to a TDateTime (see page 3335) and the date in the String is invalid. The type of exception and the error message are displayed.

//Delphi example

```
const
  LF = #10;
var
  TempDate: TDateTime;
begin
try
    TempDate := StrToDateTime('99/99/1998');
except
on E: EConvertError do
      ShowMessage(E.ClassName + LF + E.Message);
```

end;

//C++ example

```
  TDateTime TempDate;
try
  {
    TempDate = StrToDateTime("99/99/1998");
  }
catch (EConvertError &E)
  {
    ShowMessage(AnsiString(E.ClassName()) + "\n" + AnsiString(E.Message));
  }
```

See Also

DateTimeToStr (see page 4153)

DateTimeToString (see page 4153)

DateToStr (see page 4156)

EncodeDate (see page 4169)

EncodeTime (see page 4170)

FmtLoadStr (see page 4229)

FmtStr (see page 4229)

Format (see page 4232)

FormatBuf (see page 4237)

FormatCurr (see page 4238)

FormatDateTime (see page 4239)

FormatFloat (see page 4241)

IntToStr (see page 4260)

StrFmt (see page 4294)

StrLFmt (see page 4304)

StrToCurr (see page 4319)

StrToDate (see page 4320)

StrToDateTime (see page 4322)

StrToFloat (see page 4324)

StrToInt (see page 4328)

StrToTime (see page 4332)

TimeToStr (see page 4340)

TPersistent

39.1.9 EDivByZero Class

EDivByZero is the exception class for integer divide-by-zero errors.

Class Hierarchy

Pascal

```
EDivByZero = class(EIntError);
```

C++

```
class EDivByZero : public EIntError;
```

File

SysUtils

Description

EDivByZero is raised when an application tries to divide an integer by zero.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Floating-point divide-by-zero errors raise the EZeroDivide (see page 4032) exception.

See Also

EExternal (see page 3988)

EIntOverflow (see page 4000)

ERangeError (see page 4011)

EZeroDivide (see page 4032)

39.1.10 EEncodingError Class

Class Hierarchy

Pascal

```
EEncodingError = class(Exception);
```

C++

```
class EEncodingError : public Exception;
```

File

SysUtils

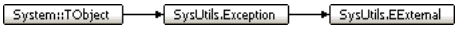
Description

This is class SysUtils.EEncodingError.

39.1.11 EExternal Class

EExternal is the exception class that captures external error information.

Class Hierarchy



Pascal

```
EExternal = class(Exception);
```

C++

```
class EExternal : public Exception;
```

File

SysUtils

Description

EExternal is the ancestor of all exception classes that represent external run time errors.

Note: Under Windows, EExternal classes represent Win32 exception codes. The ExceptionRecord (see page 3991) member points to a Win32 exception record:

//Delphi syntax

```
type
PExceptionRecord = ^TExceptionRecord;
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
  ExceptionAddress: Pointer;
  NumberParameters: Cardinal;
  ExceptionInformation: array[0..14] of Cardinal;
end;
```

ExceptionRecord (see page 3991): PExceptionRecord (see page 4379);

//C++ syntax

```
struct TExceptionRecord;
typedef TExceptionRecord *PExceptionRecord;
struct TExceptionRecord
{
unsigned ExceptionCode;
unsigned ExceptionFlags;
  TExceptionRecord *ExceptionRecord;
void *ExceptionAddress;
unsigned NumberParameters;
unsigned ExceptionInformation[15];
} ;
```

Note: Under Linux, EExternal classes represent various runtime errors. EExternal has three fields with exception information:

ExceptionAddress, which contains the address of the instruction that caused the error;

AccessAddress, the address that was being accessed when the error occurred;

SignalNumber, for exceptions raised by Linux signals.

See Also

EAccessViolation (see page 3983)

EIntError (see page 3999)

EMathError (see page 4003)

EPrivilege (see page 4009)

EStackOverflow (see page 4013)

39.1.11.1 EExternal Members

The following tables list the members exposed by EExternal.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	ExceptionRecord (see page 3991)	This is ExceptionRecord, a member of class EExternal.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.

	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.11.2 EExternal Fields

The fields of the EExternal class are listed here.

Public Fields

	Name	Description
	ExceptionRecord (see page 3991)	This is ExceptionRecord, a member of class EExternal.

39.1.11.2.1 EExternal.ExceptionRecord Field

Pascal

```
ExceptionRecord: PExceptionRecord platform;
```

C++

```
PExceptionRecord platform ExceptionRecord;
```

Description

This is ExceptionRecord, a member of class EExternal.

39.1.12 EExternalException Class

EExternalException is the exception class for invalid exception codes.

Class Hierarchy

Pascal

```
EExternalException = class(EExternal);
```

C++

```
class EExternalException : public EExternal;
```

File

SysUtils

Description

EExternalException is raised when the run time library detects an unrecognized exception code. The run time library recognizes the following exception codes:

STATUS_INTEGER_DIVIDE_BY_ZERO

STATUS_ARRAY_BOUNDS_EXCEEDED

STATUS_FLOAT_OVERFLOW

STATUS_FLOAT_INEXACT_RESULT

STATUS_FLOAT_INVALID_OPERATION

STATUS_FLOAT_STACK_CHECK

STATUS_FLOAT_DIVIDE_BY_ZERO

STATUS_INTEGER_OVERFLOW

STATUS_FLOAT_UNDERFLOW

STATUS_FLOAT_DENORMAL_OPERAND

STATUS_ACCESS_VIOLATION

STATUS_PRIVILEGED_INSTRUCTION

STATUS_CONTROL_C_EXIT

STATUS_STACK_OVERFLOW

Any exception code that is not on the list above raises an EExternalException.

Note: As a descendant of EExternal (see page 3988), EExternalException saves the exception record in its ExceptionRecord (see page 3991) data structure. An exception record is only generated under Windows.

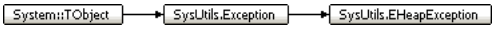
See Also

EExternal (see page 3988)

39.1.13 EHeapException Class

EHeapException is the exception class for errors related to heap-allocated memory.

Class Hierarchy



Pascal

```
EHeapException = class(Exception);
```

C++

```
class EHeapException : public Exception;
```

File

SysUtils

Description

EHeapException's descendants—EOutOfMemory (see page 4008) and EInvalidPointer (see page 4002)—are used to handle failed allocations of dynamic memory and invalid pointer operations.

Note: Memory for these exceptions is pre-allocated whenever an application starts and remains allocated as long as the application is running. Never raise EHeapException or its descendants directly.

See Also

EOutOfMemory (see page 4008)

EInvalidPointer (see page 4002)

39.1.13.1 EHeapException Members

The following tables list the members exposed by EHeapException.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3995)	This is RaisingException, a member of class EHeapException.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3994)	Deallocates the memory occupied by an EHeapException (see page 3992) instance.

	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.13.2 EHeapException Methods

The methods of the EHeapException class are listed here.

Protected Methods

	Name	Description
	RaisingException (see page 3995)	This is RaisingException, a member of class EHeapException.

Public Methods

	Name	Description
	FreeInstance (see page 3994)	Deallocates the memory occupied by an EHeapException (see page 3992) instance.

39.1.13.2.1 EHeapException.FreeInstance Method

Deallocates the memory occupied by an EHeapException (see page 3992) instance.

Pascal

```
procedure FreeInstance; override;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

FreeInstance is called internally when an application shuts down. Do not call this method directly or attempt to create or destroy instances of EHeapException (see page 3992).

39.1.13.2.2 EHeapException.RaisingException Method

Pascal

```
procedure RaisingException(P: PExceptionRecord); override;
```

C++

```
virtual __fastcall RaisingException(PExceptionRecord P);
```

Description

This is RaisingException, a member of class EHeapException.

39.1.14 EInOutError Class

EInOutError is the exception class for file input/output errors.

Class Hierarchy

Pascal

```
EInOutError = class(Exception);
```

C++

```
class EInOutError : public Exception;
```

File

SysUtils

Description

EInOutError is raised when an file input/output error occurs, provided I/O checking is enabled.

Note: In Delphi code, use the $I+ directive to enable I/O checking. If an I/O error occurs while this directive is disabled, the application must call IOResult (see page 3700) to clear the error.

Note: In C++ programs, I/O checking is a project option.

The error code is available in the ErrorCode (see page 3998) class member. Error (see page 3662) codes are come in ranges: 0-99 (native OS errors)

Error (see page 3662) codes in the range 0-99 represent OS error conditions, which are different for Windows and Linux. Refer to the OS documentation for complete error summaries. The SysErrorMessage (see page 4336) function returns descriptive text for OS errors.

Here are some common OS I/O errors, arranged by rough equivalents in Linux and Windows:

LinuxWindows

Linux		Windows	
Error (see page 3662) Code	Description	Error (see page 3662) Code	Description
2	No such file or directory	2	File not found
3	Path not found		
5	I/O Error (see page 3662)		
13	Permission denied	5	Access denied
20	Not a directory		
21	Is a directory		
32	Sharing violation		
Error (see page 3662) codes in the range 100 to 149 represent error conditions raised by CLX. Here are the CLX I/O error codes:			

Error (see page 3662) code	Meaning
100	End of File
101	Disk Full
102	File variable not assigned
103	File not open
104	File not open for input
105	File not open for output
106	Error (see page 3662) in formatted input
107	File already open

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
```

```
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

IOResult (see page 3700)

EStreamError

SysErrorMessage (see page 4336)

39.1.14.1 EInOutError Members

The following tables list the members exposed by EInOutError.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	ErrorCode (see page 3998)	This is ErrorCode, a member of class EInOutError.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.

	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.14.2 EInOutError Fields

The fields of the EInOutError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 3998)	This is ErrorCode, a member of class EInOutError.

39.1.14.2.1 EInOutError.ErrorCode Field

Pascal

```
ErrorCode: Integer;
```

C++

```
int ErrorCode;
```

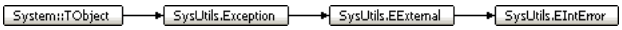
Description

This is ErrorCode, a member of class EInOutError.

39.1.15 EIntError Class

EIntError is the base class for integer math error exceptions.

Class Hierarchy



Pascal

```
EIntError = class(EExternal);
```

C++

```
class EIntError : public EExternal;
```

File

SysUtils

Description

EIntError is a generic integer math exception. Although it is never raised in the runtime library, it is the base from which other integer math exceptions descend. These are the integer math exceptions:

Exception (see page 4015)	Meaning
EDivByZero (see page 3987)	An attempt was made to divide by zero
ERangeError (see page 4011)	Number or expression out of range
EIntOverflow (see page 4000)	Integer operation overflowed

Run-time exception information is saved in fields provided by EExternal (see page 3988).

See Also

EDivByZero (see page 3987)

EExternal (see page 3988)

ERangeError (see page 4011)

EIntOverflow (see page 4000)

39.1.16 EIntfCastError Class

EIntfCastError is the exception class for interface casting errors.

Class Hierarchy

Pascal

```
EIntfCastError = class(Exception);
```

C++

```
class EIntfCastError : public Exception;
```

File

SysUtils

Description

EIntfCastError is raised when an illegal cast is attempted on an interface.

See Also

EInvalidCast (see page 4000)

39.1.17 EIntOverflow Class

EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.

Class Hierarchy

Pascal

```
EIntOverflow = class(EIntError);
```

C++

```
class EIntOverflow : public EIntError;
```

File

SysUtils

Description

EIntOverflow is a occurs when data is lost because an integer result is too large to retain.

Note: In Delphi Code, EIntOverflow is raised only if overflow checking is turned on. To turn on overflow checking, include the $Q+ directive in project source code, or select Project|Options, choose the Compiler tab, and check the Overflow-checking option in the dialog box.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Floating-point overflows raise the EOverflow (see page 4008) exception.

See Also

EExternal (see page 3988)

ERangeError (see page 4011)

EDivByZero (see page 3987)

EOverflow (see page 4008)

39.1.18 EInvalidCast Class

EInvalidCast is an exception class for typecasting errors.

Class Hierarchy

Pascal

```
EInvalidCast = class(Exception);
```

C++

```
class EInvalidCast : public Exception;
```

File

SysUtils

Description

EInvalidCast is raised when an illegal typecast is attempted.

See Also

EIntfCastError (see page 3999)

Bad

39.1.19 EInvalidContainer Class

Class Hierarchy

Pascal

```
EInvalidContainer = class(Exception);
```

C++

```
class EInvalidContainer : public Exception;
```

File

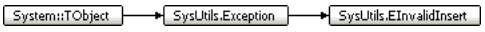
SysUtils

Description

This is class SysUtils.EInvalidContainer.

39.1.20 EInvalidInsert Class

Class Hierarchy



Pascal

```
EInvalidInsert = class(Exception);
```

C++

```
class EInvalidInsert : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EInvalidInsert.

39.1.21 EInvalidOp Class

EInvalidOp is the exception class for undefined floating-point operations.

Class Hierarchy

Pascal

```
EInvalidOp = class(EMathError);
```

C++

```
class EInvalidOp : public EMathError;
```

File

SysUtils

Description

EInvalidOp is raised when the CPU encounters an undefined instruction, invalid operation, or floating-point stack overflow.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Do not confuse EInvalidOp with EInvalidOperation, the exception class for invalid component operations.

See Also

EExternal (see page 3988)

EInvalidOperation

EInvalidArgument

EOverflow (see page 4008)

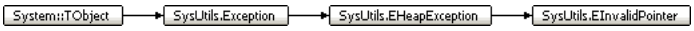
EUnderflow (see page 4013)

EZeroDivide (see page 4032)

39.1.22 EInvalidPointer Class

EInvalidPointer is the exception class for invalid pointer operations.

Class Hierarchy



Pascal

```
EInvalidPointer = class(EHeapException);
```

C++

```
class EInvalidPointer : public EHeapException;
```

File

SysUtils

Description

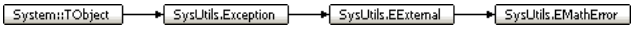
EInvalidPointer is raised when an application attempts an invalid pointer operation. For example, it can occur if an application tries to dispose of the same pointer twice, or refers to a pointer which has already been disposed of.

Note: Never raise an EInvalidPointer exception directly. EInvalidPointer is raised internally by the memory manager.

39.1.23 EMathError Class

EMathError is the base exception class for floating-point math errors.

Class Hierarchy



Pascal

```
EMathError = class(EExternal);
```

C++

```
class EMathError : public EExternal;
```

File

SysUtils

Description

EMathError is the base class for floating-point math exception classes. EMathError itself is never raised.

The following exceptions descend from EMathError:

Exception (see page 4015)	Meaning
EInvalidArgument	Parameter out of range
EInvalidOp (see page 4002)	Processor encountered an undefined instruction
EOverflow (see page 4008)	Floating-point operation produced result too large to store
EUnderflow (see page 4013)	Floating-point operation produced result with no precision
EZeroDivide (see page 4032)	Attempt to divide by zero

Run-time exception information is saved in fields provided by EExternal (see page 3988).

See Also

EExternal (see page 3988)

EInvalidArgument

EInvalidOp (see page 4002)

EOverflow (see page 4008)

EUnderflow (see page 4013)

EZeroDivide (see page 4032)

39.1.24 EMonitor Class

Class Hierarchy

Pascal

```
EMonitor = class(Exception);
```

C++

```
class EMonitor : public Exception;
```

File

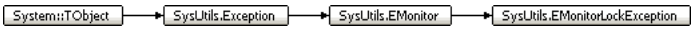
SysUtils

Description

This is class SysUtils.EMonitor.

39.1.25 EMonitorLockException Class

Class Hierarchy



Pascal

```
EMonitorLockException = class(EMonitor);
```

C++

```
class EMonitorLockException : public EMonitor;
```

File

SysUtils

Description

This is class SysUtils.EMonitorLockException.

39.1.26 ENoConstructException Class

Class Hierarchy

Pascal

```
ENoConstructException = class(Exception);
```

C++

```
class ENoConstructException : public Exception;
```

File

SysUtils

Description

This is class SysUtils.ENoConstructException.

39.1.27 ENoMonitorSupportException Class

Class Hierarchy

Pascal

```
ENoMonitorSupportException = class(EMonitor);
```

C++

```
class ENoMonitorSupportException : public EMonitor;
```

File

SysUtils

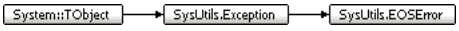
Description

This is class SysUtils.ENoMonitorSupportException.

39.1.28 EOSError Class

EOSError is the exception class for operating system errors.

Class Hierarchy



Pascal

```
EOSError = class(Exception);
```

C++

```
class EOSError : public Exception;
```

File

SysUtils

Description

EOSError is raised by the RaiseLastOSError (see page 4274) procedure if the operating system returns an error code. The exception dialog contains the error number and the message associated with the error.

Note: EOSError replaces EWin32Error (see page 4015), which is now deprecated.

See Also

RaiseLastOSError (see page 4274)

39.1.28.1 EOSError Members

The following tables list the members exposed by EOSError.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	ErrorCode (see page 4007)	This is ErrorCode, a member of class EOSError.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.

	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.28.2 EOSError Fields

The fields of the EOSError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 4007)	This is ErrorCode, a member of class EOSError.

39.1.28.2.1 EOSError.ErrorCode Field

Pascal

```
ErrorCode: DWORD;
```

C++

```
DWORD ErrorCode;
```

Description

This is ErrorCode, a member of class EOSError.

39.1.29 EOutOfMemory Class

EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.

Class Hierarchy

Pascal

```
EOutOfMemory = class(EHeapException);
```

C++

```
class EOutOfMemory : public EHeapException;
```

File

SysUtils

Description

EOutOfMemory occurs when an application attempts to allocate dynamic memory, but there is not enough free memory in the system to meet the request.

Memory for the EOutOfMemory exception is pre-allocated whenever an application starts and remains allocated as long as the application is running.

Note: Never raise EOutOfMemory directly. Instead, call the global OutOfMemoryError (see page 4272) procedure.

See Also

EOutOfResources

OutOfMemoryError (see page 4272)

EHeapException (see page 3992)

39.1.30 EOverflow Class

EOverflow is the exception class for floating-point register overflows.

Class Hierarchy

Pascal

```
EOverflow = class(EMathError);
```

C++

```
class EOverflow : public EMathError;
```

File

SysUtils

Description

EOverflow is raised when a calculated result is too large to fit in the floating-point register allocated for it and data is therefore

lost.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Integer register overflows raise the EIntOverflow (see page 4000) exception.

See Also

EExternal (see page 3988)

EInvalidArgument

EInvalidOp (see page 4002)

EUnderflow (see page 4013)

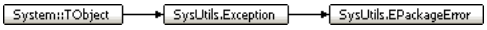
EZeroDivide (see page 4032)

EIntOverflow (see page 4000)

39.1.31 EPackageError Class

EPackageError is the exception class for package-related errors.

Class Hierarchy



Pascal

```
EPackageError = class(Exception);
```

C++

```
class EPackageError : public Exception;
```

File

SysUtils

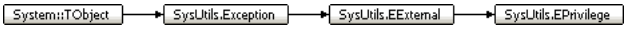
Description

EPackageError is raised at design time only, when an error occurs loading or utilizing a package.

39.1.32 EPrivilege Class

EPrivilege is the exception class for processor privilege violations.

Class Hierarchy



Pascal

```
EPrivilege = class(EExternal);
```

C++

```
class EPrivilege : public EExternal;
```

File

SysUtils

Description

EPrivilege is raised when an application tries to execute a processor instruction that is invalid for the current processor privilege level.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

See Also

EExternal (see page 3988)

39.1.33 EProgrammerNotFound Class

Class Hierarchy

Pascal

```
EProgrammerNotFound = class(Exception);
```

C++

```
class EProgrammerNotFound : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EProgrammerNotFound.

39.1.34 EPropReadOnly Class

EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.

Class Hierarchy

Pascal

```
EPropReadOnly = class(Exception);
```

C++

```
class EPropReadOnly : public Exception;
```

File

SysUtils

Description

EPropReadOnly is raised when an application tries to write to a read-only property using OLE automation.

39.1.35 EPropWriteOnly Class

EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.

Class Hierarchy

Pascal

```
EPropWriteOnly = class(Exception);
```

C++

```
class EPropWriteOnly : public Exception;
```

File

SysUtils

Description

EPropWriteOnly is raised when an application tries to read a write-only property using OLE automation.

39.1.36 ERangeError Class

ERangeError is the exception for Delphi range errors.

Class Hierarchy

Pascal

```
ERangeError = class(EIntError);
```

C++

```
class ERangeError : public EIntError;
```

File

SysUtils

Description

ERangeError occurs in Delphi code when range checking is enabled and an ordinal values goes outside its declared range.

Note: Range checking is not a C++ feature. However, ERangeError can occur when C++ code calls Delphi code.

Note: By default, the Delphi compiler disables range checking. The compiler can always detect obvious range errors, which prevents the following code from compiling.

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
begin;
  SmallValue := 4;  //Won't compile: constant expression out of bounds
```

```
  SevenBits := 256; //Won't compile: constant expression out of bounds
```

end;

Note: However, the compiler can't anticipate out-of-range values provided at run time:

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
  ThirtyTwoBits: Integer
begin;
  ThirtyTwoBits := 4;
  SmallValue := ThirtyTwoBits; //Assigns 4 to SmallValue (SmallValue is stored as a Byte)
  ThirtyTwoBits := 256;
  SevenBits := ThirtyTwoBits;  //Assigns 0 to ThirtyTwoBits (high-order bits discarded)
```

end;

Note: If range checking is enabled, both out-of-bounds assignments throw ERangeError exceptions. To enable range checking, use the Project option in the IDE or the $R+ directive.

Note: Range checking is considered a debugging feature. It produces bigger and slower executables.

Note: Do not confuse ERangeError with ERangeException, which relates to errors using graduated controls in VisualCLX.

See Also

EExternal (see page 3988)

EIntOverflow (see page 4000)

EDivByZero (see page 3987)

ERangeException

39.1.37 ESafecallException Class

ESafecallException is the exception for problems with the safecall calling convention.

Class Hierarchy

Pascal

```
ESafecallException = class(Exception);
```

C++

```
class ESafecallException : public Exception;
```

File

SysUtils

Description

ESafecallException is raised when the safecall error handler has not been set up and a safecall routine returns a non-0 HResult, or if the safecall error handler does not raise an exception.

If this exception occurs, the Comobj unit is probably missing from the application's uses list (Delphi) or not included in the project source file (C++). You may want to consider removing the safecall calling convention from the routine that gave rise to the exception.

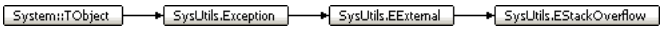
See Also

TComponent

39.1.38 EStackOverflow Class

EStackOverflow is the exception class for stack overflows.

Class Hierarchy



Pascal

```
EStackOverflow = class(EExternal);
```

C++

```
class EStackOverflow : public EExternal;
```

File

SysUtils

Description

EStackOverflow is raised when the system is unable to allocate necessary stack space. This can happen because of extremely large local variables, deeply recursive routines, or invalid machine-language code.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

See Also

EExternal (see page 3988)

39.1.39 EUnderflow Class

EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.

Class Hierarchy

Pascal

```
EUnderflow = class(EMathError);
```

C++

```
class EUnderflow : public EMathError;
```

File

SysUtils

Description

When a floating-point math operation produces a value that is too small to be represented in a floating-point variable, the result ordinarily becomes zero. However, an application can change the numeric coprocessor's control word to unmask underflow hardware exceptions. In this case, EUnderflow is raised when the processor reports an underflow hardware exception. The control word is reset after the exception.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

See Also

EExternal (see page 3988)

EInvalidArgument

EInvalidOp (see page 4002)

EOverflow (see page 4008)

EZeroDivide (see page 4032)

39.1.40 EVariantError Class

EVariantError is the exception class for errors involving Variant (see page 3430) data types.

Class Hierarchy

Pascal

```
EVariantError = class(Exception);
```

C++

```
class EVariantError : public Exception;
```

File

SysUtils

Description

EVariantError is raised when a variant operation fails. EVariantError has a number of descendant classes which provide more specific information about the error.

See Also

EVariantArrayCreateError (see page 5002)

EVariantArrayLockedError (see page 5003)

EVariantBadIndexError (see page 5004)

EVariantBadVarTypeError (see page 5004)

EVariantDispatchError (see page 5005)

EVariantInvalidArgError (see page 5005)

EVariantInvalidOpError (see page 5007)

EVariantNotImplError (see page 5008)

EVariantOutOfMemoryError (see page 5008)

EVariantOverflowError (see page 5009)

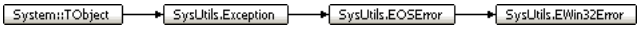
EVariantTypeCastError (see page 5010)

EVariantUnexpectedError (see page 5011)

39.1.41 EWin32Error Class

EWin32Error is the exception class for Windows errors.

Class Hierarchy



Pascal

```
EWin32Error = class(EOSError);
```

C++

```
class EWin32Error : public EOSError;
```

File

SysUtils

Description

EWin32Error is now deprecated in favor of EOSError (see page 4005).

See Also

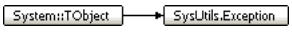
RaiseLastWin32Error (see page 4275)

EOSError (see page 4005)

39.1.42 Exception Class

Exception is the base class for all runtime exceptions.

Class Hierarchy



Pascal

```
Exception = class(TObject);
```

C++

```
class Exception : public TObject;
```

File

SysUtils

Description

Exception encapsulates the fundamental properties and methods for all exceptions. Use Exception as the base class for creating custom exceptions.

All methods introduced for the Exception object are constructors that provide alternative ways to create exception messages. Some constructors also create help context IDs. Typically, an application calls these constructors dynamically when an exception occurs.

Exception messages can be hard-coded strings, formatted strings, or strings (including formatted strings) loaded from an application resource.

Exceptions are raised when a runtime error occurs in an application, such as attempting to divide by zero. When an exception is raised, typically an exception instance displays a dialog box describing the error condition. If an application does not handle the exception condition, then the default exception handler is called. This handler also displays a dialog box with an OK button that usually permits an application to continue processing when the user clicks OK.

Note: Standard practice is to derive all exception classes from Exception, and specify a name beginning with "E", followed by an abbreviated, descriptive name.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

39.1.42.1 Exception Members

The following tables list the members exposed by Exception.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.

	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.

	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.42.2 Exception.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const Msg: string);
```

C++

```
__fastcall Exception(const UnicodeString Msg);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
```

```
  the contents of the string grid are written to the specified
  file. Additional information is also written to the file so
  that it can be read easily with the FileRead function.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    BackupName: string;
    FileHandle: Integer;
    StringLen: Integer;
    X, Y, I: Integer;
    colCountLength, rowCountLength: Integer;
    Buffer: PChar;
    cellString: string;
  begin
    if SaveDialog1.Execute then
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
```

```
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  In addition to displaying the exception message, which
  happens by default, the following code shuts down the
  application when an exception is not caught and handled.
  AppException should be declared a method of TForm1.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnException := AppException;
  end;

  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    Application.ShowException(E);
    Application.Terminate;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    raise EPasswordInvalid.Create('Incorrect password entered');
  end;
```

See Also

CreateFmt (see page 4020)

CreateRes (see page 4022)

CreateResFmt (see page 4023)

Message (see page 4031)

39.1.42.3 Exception.CreateFmt Constructor

Instantiates an instance of an exception with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const);
```

C++

```
__fastcall Exception(const UnicodeString Msg, const array of const Args);
```

Description

Call CreateFmt to construct an exception with a formatted message string. Msg is a string constant containing format specifiers to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt calls the Format (see page 4232) function to transform Msg with the values in Arg.

See Also

CreateResFmt (see page 4023)

CreateResFmtHelp (see page 4024)

Format (see page 4232)

Message (see page 4031)

39.1.42.4 Exception.CreateFmtHelp Constructor

Instantiates an instance of an exception with a formatted message string and a help-context ID.

Pascal

```
constructor CreateFmtHelp(const Msg: string; const Args: array of const; AHelpContext:
Integer);
```

C++

```
__fastcall Exception(const UnicodeString Msg, const array of const Args, int AHelpContext);
```

Description

Call CreateFmtHelp to construct an exception with a formatted message string and a help context ID. Msg is a string constant containing format specifiers to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt (see page 4020) calls the Format (see page 4232) function to transform Msg with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

CreateHelp (see page 4021)

CreateResFmtHelp (see page 4024)

Format (see page 4232)

HelpContext (see page 4030)

Message (see page 4031)

39.1.42.5 Exception.CreateHelp Constructor

Creates an instance of an exception with a simple message string and a help- context ID.

Pascal

```
constructor CreateHelp(const Msg: string; AHelpContext: Integer);
```

C++

```
__fastcall Exception(const UnicodeString Msg, int AHelpContext);
```

Description

Call CreateHelp to construct an exception with a simple message string and a help-context ID. Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 4030)

Message (see page 4031)

CreateFmtHelp (see page 4021)

CreateResFmtHelp (see page 4024)

Format (see page 4232)

39.1.42.6 CreateRes Constructor

This is the overview for the CreateRes constructor overload.

Overload List

	Name	Description
	Exception.CreateRes (Integer) (see page 4022)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.
	Exception.CreateRes (PResStringRec) (see page 4022)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.

39.1.42.6.1 Exception.CreateRes Constructor (Integer)

Creates an instance of an exception with a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateRes(Ident: Integer); overload;
constructor CreateRes(ResStringRec: PResStringRec); overload;
```

C++

```
__fastcall Exception(int Ident);
__fastcall Exception(PResStringRec ResStringRec);
```

Description

Call CreateRes to construct an exception with a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateRes creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
...
Exception.Create(@sMyNewErrorMessage);
```

See Also

Message (see page 4031)

CreateResFmt (see page 4023)

CreateResFmtHelp (see page 4024)

39.1.42.7 CreateResFmt Constructor

This is the overview for the CreateResFmt constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmt (Integer, array of const) (see page 4023)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.
	Exception.CreateResFmt (PResStringRec, array of const) (see page 4023)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

39.1.42.7.1 Exception.CreateResFmt Constructor (Integer, array of const)

Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

Pascal

```
constructor CreateResFmt(Ident: Integer; const Args: array of const); overload;
constructor CreateResFmt(ResStringRec: PResStringRec; const Args: array of const); overload;
```

C++

```
__fastcall Exception(int Ident, const array of const Args);
__fastcall Exception(PResStringRec ResStringRec, const array of const Args);
```

Description

Call CreateResFmt to construct an exception with a message string loaded from an application's resources and then formatted with additional information. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1]);
```

Args is an array of constants containing values to:

1Format (see page 4232) according to format specifiers embedded in the string, and

2Insert (see page 3698) into the message string.

CreateResFmt calls the Format (see page 4232) function to transform the message string with the values in Arg.

See Also

Message (see page 4031)

CreateRes (see page 4022)

CreateResFmtHelp (see page 4024)

Format (see page 4232)

39.1.42.8 CreateResFmtHelp Constructor

This is the overview for the CreateResFmtHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmtHelp (Integer, array of const, Integer) (see page 4024)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources
	Exception.CreateResFmtHelp (PResStringRec, array of const, Integer) (see page 4024)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

39.1.42.8.1 Exception.CreateResFmtHelp Constructor (Integer, array of const, Integer)

Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

Pascal

```
constructor CreateResFmtHelp(ResStringRec: PResStringRec; const Args: array of const;
AHelpContext: Integer); overload;
constructor CreateResFmtHelp(Ident: Integer; const Args: array of const; AHelpContext:
Integer); overload;
```

C++

```
__fastcall Exception(PResStringRec ResStringRec, const array of const Args, int AHelpContext);
__fastcall Exception(int Ident, const array of const Args, int AHelpContext);
```

Description

Call CreateResFmtHelp to construct an exception with a:

1Message (see page 4031) string loaded from an application's resources and formatted with additional information.

2Help-context ID.

Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file. Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt (see page 4023) creates an empty message string.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1], IDH_MYNEWERROR);
```

Args is an array of constants containing values to:

1Format (see page 4232) according to format specifiers embedded in the string, and

2Insert (see page 3698) into the message string.

CreateResFmtHelp calls the Format (see page 4232) function to transform the message string with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 4030)

Message (see page 4031)

CreateHelp (see page 4021)

CreateRes (see page 4022)

CreateResHelp (see page 4025)

Format (see page 4232)

39.1.42.9 CreateResHelp Constructor

This is the overview for the CreateResHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResHelp (Integer, Integer) (see page 4025)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.
	Exception.CreateResHelp (PResStringRec, Integer) (see page 4025)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

39.1.42.9.1 Exception.CreateResHelp Constructor (Integer, Integer)

Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateResHelp(Ident: Integer; AHelpContext: Integer); overload;
constructor CreateResHelp(ResStringRec: PResStringRec; AHelpContext: Integer); overload;
```

C++

```
__fastcall Exception(int Ident, int AHelpContext);
__fastcall Exception(PResStringRec ResStringRec, int AHelpContext);
```

Description

Call CreateResHelp to construct an exception with a help-context ID and a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResHelp creates an empty message string for the exception.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

ResStringRec is a pointer to a resource string. Its syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.Create(@sMyNewErrorMessage, IDH_MYNEWERROR);
```

See Also

HelpContext (see page 4030)

Message (see page 4031)

CreateHelp (see page 4021)

CreateRes (see page 4022)

CreateResFmtHelp (see page 4024)

Format (see page 4232)

39.1.42.10 Exception.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~Exception();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

39.1.42.11 Exception Fields

The fields of the Exception class are listed here.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

39.1.42.11.1 Exception.CleanUpStackInfoProc Field

Pascal

```
class var CleanUpStackInfoProc: procedure (Info: Pointer);
```

C++

```
procedure (Info: Pointer) CleanUpStackInfoProc;
```

Description

This is CleanUpStackInfoProc, a member of class Exception.

39.1.42.11.2 Exception.GetExceptionStackInfoProc Field

Pascal

```
class var GetExceptionStackInfoProc: function (P: PExceptionRecord): Pointer;
```

C++

```
function (P: PExceptionRecord): Pointer GetExceptionStackInfoProc;
```

Description

This is GetExceptionStackInfoProc, a member of class Exception.

39.1.42.11.3 Exception.GetStackInfoStringProc Field

Pascal

```
class var GetStackInfoStringProc: function (Info: Pointer): string;
```

C++

```
function (Info: Pointer): string GetStackInfoStringProc;
```

Description

This is GetStackInfoStringProc, a member of class Exception.

39.1.42.12 Exception Methods

The methods of the Exception class are listed here.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Methods

	Name	Description
V	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 4029)	Returns a string containing the class name.

39.1.42.12.1 Exception.GetBaseException Method

Pascal

```
function GetBaseException: Exception; virtual;
```

C++

```
virtual __fastcall Exception GetBaseException();
```

Description

This is GetBaseException, a member of class Exception.

39.1.42.12.2 Exception.GetStackTrace Method

Pascal

```
function GetStackTrace: string;
```

C++

```
__fastcall UnicodeString GetStackTrace();
```

Description

This is GetStackTrace, a member of class Exception.

39.1.42.12.3 Exception.RaiseOuterException Method

Pascal

```
class procedure RaiseOuterException(E: Exception); static;
```

C++

```
static __fastcall RaiseOuterException(Exception E);
```

Description

This is RaiseOuterException, a member of class Exception.

39.1.42.12.4 Exception.RaisingException Method

Pascal

```
procedure RaisingException(P: PExceptionRecord); virtual;
```

C++

```
virtual __fastcall RaisingException(PExceptionRecord P);
```

Description

This is RaisingException, a member of class Exception.

39.1.42.12.5 Exception.SetInnerException Method

Pascal

```
procedure SetInnerException;
```

C++

```
__fastcall SetInnerException();
```

Description

This is SetInnerException, a member of class Exception.

39.1.42.12.6 Exception.SetStackInfo Method

Pascal

```
procedure SetStackInfo(AStackInfo: Pointer);
```

C++

```
__fastcall SetStackInfo(void * AStackInfo);
```

Description

This is SetStackInfo, a member of class Exception.

39.1.42.12.7 Exception.ThrowOuterException Method

Pascal

```
class procedure ThrowOuterException(E: Exception); static;
```

C++

```
static __fastcall ThrowOuterException(Exception E);
```

Description

This is ThrowOuterException, a member of class Exception.

39.1.42.12.8 Exception.ToString Method

Returns a string containing the class name.

Pascal

```
function ToString: string; override;
```

C++

```
virtual __fastcall UnicodeString ToString();
```

Description

By default, the System::TObject::ToString returns a string containing the class name of the instance that is being called.

Example

Calling System::TObject::ToString on a TButton instance returns a string containing "TButton".

Note: System::TObject::ToString is intended to be overridden in user derived classes, in order to provide consumer objects with a string representation.

See Also

System::TObject::ClassType (see page 3374)

System::TObject::ClassName (see page 3369)

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

System::TObject::FieldAddress (see page 3382)

System::TObject::MethodAddress (see page 3390)

System::TObject::InstanceSize (see page 3389)

System::TObject::MethodName (see page 3390)

System::TObject::InheritsFrom (see page 3388)

39.1.42.13 Exception Properties

The properties of the Exception class are listed here.

Public Properties

	Name	Description
R	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

39.1.42.13.1 Exception.BaseException Property

Pascal

```
property BaseException: Exception;
```

C++

```
__property Exception BaseException;
```

Description

This is BaseException, a member of class Exception.

39.1.42.13.2 Exception.HelpContext Property

Indicates the context ID number for context-sensitive online help associated with the exception.

Pascal

```
property HelpContext: Integer;
```

C++

```
__property int HelpContext;
```

Description

HelpContext provides storage for the help context ID number associated with the exception object. The context ID number determines the contents of the help screen that appears when a user presses F1 in response to an exception error message.

Note: To initialize HelpContext when the Exception (see page 4015) object is instantiated, use the appropriate constructor.

Note: In Delphi code, instantiate the Exception (see page 4015) object with CreateFmtHelp (see page 4021), CreateHelp (see page 4021), CreateResFmtHelp (see page 4024), or CreateResHelp (see page 4025).

Note: In C++ code, use a form of the constructor that takes a help context parameter.

See Also

Create (see page 4018)

CreateFmtHelp (see page 4021)

CreateHelp (see page 4021)

CreateResFmtHelp (see page 4024)

CreateResHelp (see page 4025)

39.1.42.13.3 Exception.InnerException Property

Pascal

```
property InnerException: Exception;
```

C++

```
__property Exception InnerException;
```

Description

This is InnerException, a member of class Exception.

39.1.42.13.4 Exception.Message Property

Message contains the text string to display in the exception dialog box when the exception is raised.

Pascal

```
property Message: string;
```

C++

```
__property UnicodeString Message;
```

Description

Message stores the error-message string to display when the exception is raised. All Exception (see page 4015) constructors expect a string parameter to store in Message. Message text can be hard-coded as a parameter to an Exception (see page 4015) constructor, created as dynamic parameter, or loaded from a resource file as a static or dynamically formatted parameter.

See Also

Create (see page 4018)

CreateFmt (see page 4020)

CreateFmtHelp (see page 4021)

CreateHelp (see page 4021)

CreateRes (see page 4022)

CreateResFmt (see page 4023)

CreateResFmtHelp (see page 4024)

CreateResHelp (see page 4025)

39.1.42.13.5 Exception.StackInfo Property

Pascal

```
property StackInfo: Pointer;
```

C++

```
__property void * StackInfo;
```

Description

This is StackInfo, a member of class Exception.

39.1.42.13.6 Exception.StackTrace Property

Pascal

```
property StackTrace: string;
```

C++

```
__property UnicodeString StackTrace;
```

Description

This is StackTrace, a member of class Exception.

39.1.43 EZeroDivide Class

EZeroDivide is the exception class for floating-point divide-by-zero errors.

Class Hierarchy

Pascal

```
EZeroDivide = class(EMathError);
```

C++

```
class EZeroDivide : public EMathError;
```

File

SysUtils

Description

EZeroDivide exception is raised when an application attempts to divide a floating-point value by zero.

Run-time exception information is saved in fields provided by EExternal (see page 3988).

Note: Integer divide-by-zero errors raise the EDivByZero (see page 3987) exception.

See Also

EExternal (see page 3988)

EInvalidArgument

EInvalidOp (see page 4002)

EOverflow (see page 4008)

EUnderflow (see page 4013)

EDivByZero (see page 3987)

39.1.44 TBigEndianUnicodeEncoding Class

Class Hierarchy

Pascal

```
TBigEndianUnicodeEncoding = class(TUnicodeEncoding);
```

C++

```
class TBigEndianUnicodeEncoding : public TUnicodeEncoding;
```

File

SysUtils

Description

This is class SysUtils.TBigEndianUnicodeEncoding.

39.1.44.1 TBigEndianUnicodeEncoding Members

The following tables list the members exposed by TBigEndianUnicodeEncoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
V	Create (see page 4090)	This is Create, a member of class TUnicodeEncoding.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
A	GetMaxByteCount (see page 4046)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxByteCount (see page 4092)	Calculates the maximum bytes generated to encode the specified number of characters.
A	GetMaxCharCount (see page 4046)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetMaxCharCount (see page 4092)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4035)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4090)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4034)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4091)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 4035)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.44.2 TBigEndianUnicodeEncoding Methods

The methods of the TBigEndianUnicodeEncoding class are listed here.

Public Methods

	Name	Description
V	GetPreamble (see page 4035)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
V	GetBytes (see page 4034)	Encodes a set of characters into a sequence of bytes.
V	GetChars (see page 4035)	Decodes a set of bytes into a set of characters.

39.1.44.2.1 TBigEndianUnicodeEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

39.1.44.2.2 TBigEndianUnicodeEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

39.1.44.2.3 TBigEndianUnicodeEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

39.1.45 TEncoding Class

TEncoding is a class that defines a specific encoding for a character or string.

Class Hierarchy

Pascal

```
TEncoding = class;
```

C++

```
class TEncoding;
```

File

SysUtils

Description

Use the TEncoding class to define a specific encoding desired for a given string. For instance, the TStringStream class now includes an overloaded constructor that enables the developer to define encoding for strings that will be added to an instance of TStringStream.

To simplify cross platform developement, there are similarities between this interface and System.Text.Encoding. For that reason, links are provided to MSDN counterparts in Encoding (MSDN) class.

C++ Examples:

```
/*
This example reads any file and converts the encoding to UTF8.
This example uses the TBytesStream to read the files into a TBytes'
and also uses the auto_ptr construction to clean up created class
instances.

*/
#include <tchar.h>
#include <memory>       //for STL auto_ptr class
//---------------------------------------------------------------------------

#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
  // Sample to convert a file of any encoding to UTF8.
  TEncoding *LEncoding = NULL;
  std::auto_ptr<TFileStream> LFileStream(new TFileStream("..\\Sample.txt", fmOpenRead));

  // Read file into buffer
  TBytes myBytes;
  std::auto_ptr<TBytesStream> myBytesStream(new TBytesStream(myBytes));
  myBytesStream->CopyFrom(LFileStream.get(), LFileStream->Size);

  // Identify encoding and convert buffer to UTF8
  int LOffset = TEncoding::GetBufferEncoding(myBytesStream->Bytes, LEncoding);
  if (LOffset == 0)
    return 1; // Unknown encoding, don't convert.
  myBytes = TEncoding::Convert(LEncoding, TEncoding::UTF8,
                               myBytesStream->Bytes,
                               LOffset, myBytesStream->Size-LOffset);
  // Create output file
  std::auto_ptr<TFileStream> DestFileStream(new TFileStream("..\\SampleUTF8.txt", fmCreate));

  // Write UTF8 byte order mark and buffer to output file
  TBytes LByteOrderMark;
  LByteOrderMark = TEncoding::UTF8->GetPreamble();

  // Grab preamble and write to destination
  DestFileStream->Write(&LByteOrderMark[0], LByteOrderMark.Length);
```

```
  // Write converted buffer
  DestFileStream->Write(&myBytes[0], myBytes.Length);
  return 0;
}
```

Delphi Examples:

```
{
This example determines the encoding of a source file and then
write the file in any encoding chosen.  Use this project to
create files of any encoding.  This example requires two text edits
and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  LFileStream: TFileStream;
  EncodingArray: array[0..5] of TEncoding;
begin
  LEncoding:= nil;
  EncodingArray[0]:= TEncoding.UTF8;
  EncodingArray[1]:= TEncoding.UTF7;
  EncodingArray[2]:= TEncoding.Unicode;
  EncodingArray[3]:= TEncoding.Default;
  EncodingArray[4]:= TEncoding.BigEndianUnicode;
  EncodingArray[5]:= TEncoding.ASCII;
  DestEncoding := EncodingArray[ComboBox1.ItemIndex];
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
//    LFileStream.Read(LBuffer[0], Length(LBuffer));
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));

    // Identify encoding and convert buffer to UTF8
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
//    for I := 0 to Length(LBuffer) - 1 do
//      Memo1.Lines.Add((LBuffer[I]));
    LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
      LOffset, Length(LBuffer) - LOffset);
  finally
    LFileStream.Free;
  end;

  LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
  try
    // Write an encoding byte order mark and buffer to output file
    LByteOrderMark := DestEncoding.GetPreamble;
    LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
    LFileStream.Write(LBuffer[0], Length(LBuffer));
  finally
    LFileStream.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  StringList: TStrings;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('TEncoding.UTF8');
      Add('TEncoding.UTF7');
```

```
        Add('TEncoding.Unicode');
        Add('TEncoding.Default');
        Add('TEncoding.BigEndianUnicode');
        Add('TEncoding.ASCII');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

Convert (see page 4041)

FreeEncodings (see page 4041)

GetBufferEncoding (see page 4042)

GetByteCount (see page 4042)

GetBytes (see page 4043)

GetCharCount (see page 4044)

GetChars (see page 4045)

GetEncoding (see page 4046)

GetMaxByteCount (see page 4046)

GetMaxCharCount (see page 4046)

GetPreamble (see page 4047)

GetString (see page 4047)

IsStandardEncoding (see page 4048)

ASCII (see page 4048)

BigEndianUnicode (see page 4049)

Default (see page 4049)

IsSingleByte (see page 4049)

Unicode (see page 4050)

UTF7 (see page 4050)

UTF8 (see page 4050)

Encoding (MSDN)

Using TEncoding for Unicode

39.1.45.1 TEncoding Members

The following tables list the members exposed by TEncoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
A	GetMaxByteCount (see page 4046)	Calculates the maximum bytes generated to encode the specified number of characters.
A	GetMaxCharCount (see page 4046)	Calculates the maximum number of characters generated to decode a specified number of bytes.
A	GetPreamble (see page 4047)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
A	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
A	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
A	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
A	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.45.2 TEncoding Fields

The fields of the TEncoding class are listed here.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

39.1.45.2.1 TEncoding.FIsSingleByte Field

Pascal

```
FIsSingleByte: Boolean;
```

C++

```
Boolean FIsSingleByte;
```

Description

This is FIsSingleByte, a member of class TEncoding.

39.1.45.2.2 TEncoding.FMaxCharSize Field

Pascal

```
FMaxCharSize: Integer;
```

C++

```
int FMaxCharSize;
```

Description

This is FMaxCharSize, a member of class TEncoding.

39.1.45.3 TEncoding Methods

The methods of the TEncoding class are listed here.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
	GetMaxByteCount (see page 4046)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4046)	Calculates the maximum number of characters generated to decode a specified number of bytes.
	GetPreamble (see page 4047)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.45.3.1 Convert Method

This is the overview for the Convert method overload.

Overload List

	Name	Description
	TEncoding.Convert (TEncoding, TEncoding, TBytes) (see page 4041)	Convert an array of bytes from one encoding to another.
	TEncoding.Convert (TEncoding, TEncoding, TBytes, Integer, Integer) (see page 4041)	Convert an array of bytes from one encoding to another.

39.1.45.3.1.1 TEncoding.Convert Method (TEncoding, TEncoding, TBytes)

Convert an array of bytes from one encoding to another.

Pascal

```
class function Convert(Source: TEncoding; Destination: TEncoding; Bytes: TBytes): TBytes;
overload;
class function Convert(Source: TEncoding; Destination: TEncoding; Bytes: TBytes; StartIndex:
Integer; Count: Integer): TBytes; overload;
```

C++

```
__fastcall TBytes Convert(TEncoding Source, TEncoding Destination, TBytes Bytes);
__fastcall TBytes Convert(TEncoding Source, TEncoding Destination, TBytes Bytes, int
StartIndex, int Count);
```

Description

TEncoding.Convert converts an array of bytes from one encoding to another.

Source refers to the encoding type of the encoding of the original bytes.

Destination refers to the encoding type that the bytes are to be converted to.

StartIndex specifies the index of the array to begin converting.

Count specifies the number of bytes to convert.

The **Return Value** is the converted array of bytes.

See Also

Encoding.Convert (MSDN)

39.1.45.3.2 TEncoding.FreeEncodings Method

Frees the internal standard encoding instances.

Pascal

```
class procedure FreeEncodings;
```

C++

```
__fastcall FreeEncodings();
```

Description

TEncoding.FreeEncodings clears and frees the internal standard encodings returned by the properties ASCII (see page 4048), Default (see page 4049), UTF8 (see page 4050), Unicode (see page 4050) and BigEndianUnicode (see page 4049). This method is for internal use only, do not call directly.

See Also

ASCII (see page 4048)

Default (see page 4049)

UTF8 (see page 4050)

Unicode (see page 4050)

BigEndianUnicode (see page 4049)

39.1.45.3.3 TEncoding.GetBufferEncoding Method

Determines the encoding for a given buffer.

Pascal

```
class function GetBufferEncoding(const Buffer: TBytes; var AEncoding: TEncoding): Integer;
static;
```

C++

```
static __fastcall int GetBufferEncoding(const TBytes Buffer, TEncoding AEncoding);
```

Description

TEncoding.GetBufferEncoding examines the Buffer parameter and determines its encoding by looking for known byte order marks (BOM). If the buffer doesn't contain a byte order mark for one of the standard encodings, the Default (see page 4049) standard encoding is returned.

Buffer refers to a TBytes (see page 4380) instance that will be examined.

AEncoding refers to an instance of TEncoding (see page 4035) that will be filled out with the encoding type of the buffer.

The **Return Value** of GetBufferEncoding is the length of the preamble in the Buffer of bytes.

39.1.45.3.4 GetByteCount Method

This is the overview for the GetByteCount method overload.

Overload List

	Name	Description
	TEncoding.GetByteCount (PChar, Integer) (see page 4042)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (string) (see page 4042)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (string, Integer, Integer) (see page 4042)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (TCharArray) (see page 4042)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (TCharArray, Integer, Integer) (see page 4042)	Returns the number of bytes generated by encoding Chars.

39.1.45.3.4.1 TEncoding.GetByteCount Method (PChar, Integer)

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; virtual; abstract; overload;
function GetByteCount(const Chars: TCharArray): Integer; overload;
function GetByteCount(const Chars: TCharArray; CharIndex: Integer; CharCount: Integer):
Integer; overload;
function GetByteCount(const S: string): Integer; overload;
function GetByteCount(const S: string; CharIndex: Integer; CharCount: Integer): Integer;
overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount) = 0;
__fastcall int GetByteCount(const TCharArray Chars);
__fastcall int GetByteCount(const TCharArray Chars, int CharIndex, int CharCount);
__fastcall int GetByteCount(const UnicodeString S);
__fastcall int GetByteCount(const UnicodeString S, int CharIndex, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString (see page 3393) from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

39.1.45.3.5 GetBytes Method

This is the overview for the GetBytes method overload.

Overload List

	Name	Description
	TEncoding.GetBytes (PChar, Integer, PByte, Integer) (see page 4043)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (string) (see page 4043)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (string, Integer, Integer, TBytes, Integer) (see page 4043)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (TCharArray) (see page 4043)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (TCharArray, Integer, Integer, TBytes, Integer) (see page 4043)	Encodes a set of characters into a sequence of bytes.

39.1.45.3.5.1 TEncoding.GetBytes Method (PChar, Integer, PByte, Integer)

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; virtual; abstract; overload;
function GetBytes(const Chars: TCharArray): TBytes; overload;
function GetBytes(const Chars: TCharArray; CharIndex: Integer; CharCount: Integer; var Bytes:
TBytes; ByteIndex: Integer): Integer; overload;
function GetBytes(const S: string): TBytes; overload;
function GetBytes(const S: string; CharIndex: Integer; CharCount: Integer; var Bytes: TBytes;
ByteIndex: Integer): Integer; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount)
= 0;
__fastcall TBytes GetBytes(const TCharArray Chars);
__fastcall int GetBytes(const TCharArray Chars, int CharIndex, int CharCount, TBytes Bytes,
int ByteIndex);
__fastcall TBytes GetBytes(const UnicodeString S);
__fastcall int GetBytes(const UnicodeString S, int CharIndex, int CharCount, TBytes Bytes, int
ByteIndex);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString (see page 3393) to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes (see page 4380) sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

39.1.45.3.6 GetCharCount Method

This is the overview for the GetCharCount method overload.

Overload List

	Name	Description
	TEncoding.GetCharCount (PByte, Integer) (see page 4044)	Calcuates a number of characters generated by decoding a sequence of bytes.
	TEncoding.GetCharCount (TBytes) (see page 4044)	Calcuates a number of characters generated by decoding a sequence of bytes.
	TEncoding.GetCharCount (TBytes, Integer, Integer) (see page 4044)	Calcuates a number of characters generated by decoding a sequence of bytes.

39.1.45.3.6.1 TEncoding.GetCharCount Method (PByte, Integer)

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; virtual; abstract; overload;
function GetCharCount(const Bytes: TBytes): Integer; overload;
function GetCharCount(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): Integer;
overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount) = 0;
__fastcall int GetCharCount(const TBytes Bytes);
__fastcall int GetCharCount(const TBytes Bytes, int ByteIndex, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

39.1.45.3.7 GetChars Method

This is the overview for the GetChars method overload.

Overload List

	Name	Description
	TEncoding.GetChars (PByte, Integer, PChar, Integer) (see page 4045)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes) (see page 4045)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes, Integer, Integer) (see page 4045)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes, Integer, Integer, TCharArray, Integer) (see page 4045)	Decodes a set of bytes into a set of characters.

39.1.45.3.7.1 TEncoding.GetChars Method (PByte, Integer, PChar, Integer)

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; virtual; abstract; overload;
function GetChars(const Bytes: TBytes): TCharArray; overload;
function GetChars(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): TCharArray;
overload;
function GetChars(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer; var Chars:
TCharArray; CharIndex: Integer): Integer; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount)
= 0;
__fastcall TCharArray GetChars(const TBytes Bytes);
__fastcall TCharArray GetChars(const TBytes Bytes, int ByteIndex, int ByteCount);
__fastcall int GetChars(const TBytes Bytes, int ByteIndex, int ByteCount, TCharArray Chars,
int CharIndex);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

39.1.45.3.8 TEncoding.GetEncoding Method

Returns the encoding used for the code page supplied.

Pascal

```
class function GetEncoding(CodePage: Integer): TEncoding; static;
```

C++

```
static __fastcall TEncoding GetEncoding(int CodePage);
```

Description

TEncoding.GetEncoding returns the preferred encoding for CodePage.

CodePage must be a valid codepage identifier or one of the CP_XXX constants defined by the Windows API.

See Also

Encoding.GetEncoding (MSDN)

CodePage Identifiers (MSDN)

39.1.45.3.9 TEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount) = 0;
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

39.1.45.3.10 TEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount) = 0;
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

39.1.45.3.11 TEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; virtual; abstract;
```

C++

```
virtual __fastcall TBytes GetPreamble() = 0;
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes (see page 4380) that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

39.1.45.3.12 GetString Method

This is the overview for the GetString method overload.

Overload List

	Name	Description
	TEncoding.GetString (TBytes) (see page 4047)	Decodes the byte array into a string.
	TEncoding.GetString (TBytes, Integer, Integer) (see page 4047)	Decodes the byte array into a string.

39.1.45.3.12.1 TEncoding.GetString Method (TBytes)

Decodes the byte array into a string.

Pascal

```
function GetString(const Bytes: TBytes): string; overload;
function GetString(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): string;
overload;
```

C++

```
__fastcall UnicodeString GetString(const TBytes Bytes);
__fastcall UnicodeString GetString(const TBytes Bytes, int ByteIndex, int ByteCount);
```

Description

TEncoding.GetString decodes the Bytes byte array into a string.

ByteIndex specifies the first byte to decode.

ByteCount specifies the number of bytes to decode.

See Also

Encoding.GetString (MSDN)

39.1.45.3.13 TEncoding.IsStandardEncoding Method

Reports if this instance is a standard encoding instance.

Pascal

```
class function IsStandardEncoding(AEncoding: TEncoding): Boolean; static;
```

C++

```
static __fastcall Boolean IsStandardEncoding(TEncoding AEncoding);
```

Description

TEncoding.IsStandardEncoding compares AEncoding to the instances of TEncoding (see page 4035) provided in the TEncoding (see page 4035) class properties: Default (see page 4049), ASCII (see page 4048), BigEndianUnicode (see page 4049), Unicode (see page 4050), UTF7 (see page 4050) and UTF8 (see page 4050). The Return Value specifies if that search was successful.

See Also

ASCII (see page 4048)

BigEndianUnicode (see page 4049)

Default (see page 4049)

Unicode (see page 4050)

UTF7 (see page 4050)

UTF8 (see page 4050)

39.1.45.4 TEncoding Properties

The properties of the TEncoding class are listed here.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

39.1.45.4.1 TEncoding.ASCII Property

Returns an encoding for the ASCII character set.

Pascal

```
class property ASCII: TEncoding;
```

C++

```
__property TEncoding ASCII;
```

Description

TEncoding.ASCII returns an encoding for the ASCII character set.

The **Return Value** is an instance of type TMBCSEncoding (see page 4056).

See Also

Encoding.ASCII (MSDN)

39.1.45.4.2 TEncoding.BigEndianUnicode Property

Returns an encoding for the UTF-16 format with big-endian byte order.

Pascal

```
class property BigEndianUnicode: TEncoding;
```

C++

```
__property TEncoding BigEndianUnicode;
```

Description

TEncoding.BigEndianUnicode returns an encoding for the UTF-16 format with big-endian byte order.

The **Return Value** is an instance of type TBigEndianUnicodeEncoding (see page 4033). This encoding has a byte order mark of hexadecimal FEFF.

See Also

Encoding.BigEndianUnicode (MSDN)

39.1.45.4.3 TEncoding.Default Property

Returns an encoding for the operating system's default ANSI code page.

Pascal

```
class property Default: TEncoding;
```

C++

```
__property TEncoding Default;
```

Description

TEncoding.Default returns an encoding for the operating system's default ANSI code page.

The **Return Value** is an instance of type TMBCSEncoding (see page 4056).

See Also

Encoding.Default (MSDN)

39.1.45.4.4 TEncoding.IsSingleByte Property

This boolean specifies if the current encoding is of single-byte format.

Pascal

```
property IsSingleByte: Boolean;
```

C++

```
__property Boolean IsSingleByte;
```

Description

TEncoding.IsSingleByte specifies if the current encoding is of single-byte format.

See Also

Encoding.IsSingleByte (MSDN)

39.1.45.4.5 TEncoding.Unicode Property

Returns an encoding for the UTF-16 format with little-endian byte order.

Pascal

```
class property Unicode: TEncoding;
```

C++

```
__property TEncoding Unicode;
```

Description

TEncoding.Unicode returns an encoding for the UTF-16 format with little-endian byte order.

The **Return Value** is an instance of type TUnicodeEncoding (see page 4088). This encoding has a byte order mark of hexadecimal FFFE.

See Also

Encoding.Unicode (MSDN)

39.1.45.4.6 TEncoding.UTF7 Property

Returns an encoding for the UTF-7 format.

Pascal

```
class property UTF7: TEncoding;
```

C++

```
__property TEncoding UTF7;
```

Description

TEncoding.UTF7 returns an encoding for the UTF-7 format..

The **Return Value** is an instance of type TUTF7Encoding (see page 4093).

See Also

Encoding.UTF7 (MSDN)

39.1.45.4.7 TEncoding.UTF8 Property

Returns an encoding for the UTF-8 format.

Pascal

```
class property UTF8: TEncoding;
```

C++

```
__property TEncoding UTF8;
```

Description

TEncoding.UTF8 returns an encoding for the UTF-8 format.

The **Return Value** is an instance of type TUTF8Encoding (see page 4098). This encoding has a byte order mark of hexadecimal EFBBBF.

See Also

Encoding.UTF8 (MSDN)

39.1.46 TLanguages Class

TLanguages lists all available Windows locales.

Class Hierarchy

Pascal

```
TLanguages = class;
```

C++

```
class TLanguages;
```

File

SysUtils

Description

The Languages (see page 4267) function returns a TLanguages instance that lists the Windows locales supported by the current system. For each locale, TLanguages indicates the locale ID (see page 4054) (LCID), standard name, and recommended file-name extension. This information is obtained from the operating system; all TLanguages properties are read-only.

See Also

Languages (see page 4267)

39.1.46.1 TLanguages Members

The following tables list the members exposed by TLanguages.

Public Methods

	Name	Description
	Create (see page 4052)	This is Create, a member of class TLanguages.
	IndexOf (see page 4052)	Returns the index of the first locale in the array with the specified LCID.

Public Properties

	Name	Description
	Count (see page 4053)	Contains the number of locales listed in the TLanguages (see page 4051) object.
	Ext (see page 4053)	Contains the standard file-name extension at the specified position in the array.
	ID (see page 4054)	Contains a locale ID as a string.
	LocaleID (see page 4054)	Contains a locale ID (see page 4054) as an integer.
	Name (see page 4054)	Contains the locale name at the specified position in the array.
	NameFromLCID (see page 4055)	Contains the locale name specified by a string representation of a locale ID (see page 4054).
	NameFromLocaleID (see page 4055)	Contains the locale name specified by an integer locale ID (see page 4054).

39.1.46.2 TLanguages.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TLanguages();
```

Description

This is Create, a member of class TLanguages.

39.1.46.3 TLanguages Methods

The methods of the TLanguages class are listed here.

Public Methods

	Name	Description
	IndexOf (see page 4052)	Returns the index of the first locale in the array with the specified LCID.

39.1.46.3.1 TLanguages.IndexOf Method

Returns the index of the first locale in the array with the specified LCID.

Pascal

```
function IndexOf(ID: LCID): Integer;
```

C++

```
__fastcall int IndexOf(LCID ID);
```

Description

Call IndexOf to get the position in the array of a specified locale ID (see page 4054). For example, if U.S. English is the first locale listed in TLanguages (see page 4051), then IndexOf($0409) returns 0.

If the specified LCID is not found, IndexOf returns -1.

See Also

Ext (see page 4053)

ID (see page 4054)

LocaleID (see page 4054)

Name (see page 4054)

39.1.46.4 TLanguages Properties

The properties of the TLanguages class are listed here.

Public Properties

	Name	Description
	Count (see page 4053)	Contains the number of locales listed in the TLanguages (see page 4051) object.
	Ext (see page 4053)	Contains the standard file-name extension at the specified position in the array.
	ID (see page 4054)	Contains a locale ID as a string.
	LocaleID (see page 4054)	Contains a locale ID (see page 4054) as an integer.
	Name (see page 4054)	Contains the locale name at the specified position in the array.
	NameFromLCID (see page 4055)	Contains the locale name specified by a string representation of a locale ID (see page 4054).
	NameFromLocaleID (see page 4055)	Contains the locale name specified by an integer locale ID (see page 4054).

39.1.46.4.1 TLanguages.Count Property

Contains the number of locales listed in the TLanguages (see page 4051) object.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to determine the number of locales supported by the current system.

See Also

ID (see page 4054)

LocaleID (see page 4054)

IndexOf (see page 4052)

39.1.46.4.2 TLanguages.Ext Property

Contains the standard file-name extension at the specified position in the array.

Pascal

```
property Ext [Index: Integer]: string;
```

C++

```
__property UnicodeString Ext[int Index];
```

Description

Read (see page 3727) Ext to get the standard three-letter file-name extension for a locale. For example, Ext[IndexOf (see page 4052)($0409)] returns the string ENU.

Ext is a zero-based array.

See Also

IndexOf (see page 4052)

39.1.46.4.3 TLanguages.ID Property

Contains a locale ID as a string.

Pascal

```
property ID [Index: Integer]: string;
```

C++

```
__property UnicodeString ID[int Index];
```

Description

Read (see page 3727) ID to get the locale ID at a specified position in the array as a string. (To get the same ID as an integer, use the LocaleID (see page 4054) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 4051), then ID[0] returns the string $00000409. You can iterate through the ID array (from 0 to Count (see page 4053) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 4053)

LocaleID (see page 4054)

39.1.46.4.4 TLanguages.LocaleID Property

Contains a locale ID (see page 4054) as an integer.

Pascal

```
property LocaleID [Index: Integer]: LCID;
```

C++

```
__property LCID LocaleID[int Index];
```

Description

Read (see page 3727) LocaleID to get the locale ID (see page 4054) at a specified position in the array as an integer. (To get the same ID (see page 4054) as a string, use the ID (see page 4054) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 4051), then LocaleID[0] returns the integer $409. You can iterate through the LocaleID array (from 0 to Count (see page 4053) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 4053)

ID (see page 4054)

39.1.46.4.5 TLanguages.Name Property

Contains the locale name at the specified position in the array.

Pascal

```
property Name [Index: Integer]: string;
```

C++

```
__property UnicodeString Name[int Index];
```

Description

Read (see page 3727) Name to get the standard Windows name for a locale. For example, Name[IndexOf (see page 4052)($0409)] returns the string English (United States).

Name is a zero-based array.

See Also

IndexOf (see page 4052)

NameFromLCID (see page 4055)

NameFromLocaleID (see page 4055)

39.1.46.4.6 TLanguages.NameFromLCID Property

Contains the locale name specified by a string representation of a locale ID (see page 4054).

Pascal

```
property NameFromLCID [const ID: string]: string;
```

C++

```
__property UnicodeString NameFromLCID[UnicodeString const ID];
```

Description

Read (see page 3727) NameFromLCID to get the standard Windows name for a locale using a locale ID (see page 4054) represented by a string.

See Also

Name (see page 4054)

NameFromLocaleID (see page 4055)

39.1.46.4.7 TLanguages.NameFromLocaleID Property

Contains the locale name specified by an integer locale ID (see page 4054).

Pascal

```
property NameFromLocaleID [ID: LCID]: string;
```

C++

```
__property UnicodeString NameFromLocaleID[LCID ID];
```

Description

Read (see page 3727) NameFromLocaleID to get the standard Windows name for a locale using an integer locale ID (see page 4054). For example, NameFromLCID (see page 4055)[$0409] returns the string English (United States).

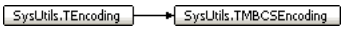
See Also

NameFromLCID (see page 4055)

Name (see page 4054)

39.1.47 TMBCSEncoding Class

Class Hierarchy



Pascal

```
TMBCSEncoding = class(TEncoding);
```

C++

```
class TMBCSEncoding : public TEncoding;
```

File

SysUtils

Description

This is class SysUtils.TMBCSEncoding.

39.1.47.1 TMBCSEncoding Members

The following tables list the members exposed by TMBCSEncoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
V	Create (see page 4057)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4060)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4060)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4061)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4058)	Returns the number of bytes generated by encoding Chars.
	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
	GetBytes (see page 4059)	Encodes a set of characters into a sequence of bytes.
	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
	GetCharCount (see page 4059)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
	GetChars (see page 4060)	Decodes a set of bytes into a set of characters.
	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.47.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TMBCSEncoding.Create () (see page 4057)	This is Create, a member of class TMBCSEncoding.
	TMBCSEncoding.Create (Integer) (see page 4057)	This is Create, a member of class TMBCSEncoding.
	TMBCSEncoding.Create (Integer, Integer, Integer) (see page 4058)	This is Create, a member of class TMBCSEncoding.

39.1.47.2.1 TMBCSEncoding.Create Constructor ()

Pascal

```
constructor Create; virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding();
```

Description

This is Create, a member of class TMBCSEncoding.

39.1.47.2.2 TMBCSEncoding.Create Constructor (Integer)

Pascal

```
constructor Create(CodePage: Integer); virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding(int CodePage);
```

Description

This is Create, a member of class TMBCSEncoding.

39.1.47.2.3 TMBCSEncoding.Create Constructor (Integer, Integer, Integer)

Pascal

```
constructor Create(CodePage: Integer; MBToWCharFlags: Integer; WCharToMBFlags: Integer);
virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding(int CodePage, int MBToWCharFlags, int WCharToMBFlags);
```

Description

This is Create, a member of class TMBCSEncoding.

39.1.47.3 TMBCSEncoding Methods

The methods of the TMBCSEncoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 4060)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4060)	Calculates the maximum number of characters generated to decode a specified number of bytes.
	GetPreamble (see page 4061)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4058)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 4059)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 4059)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 4060)	Decodes a set of bytes into a set of characters.

39.1.47.3.1 TMBCSEncoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

39.1.47.3.2 TMBCSEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

39.1.47.3.3 TMBCSEncoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

39.1.47.3.4 TMBCSEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

39.1.47.3.5 TMBCSEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

39.1.47.3.6 TMBCSEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

39.1.47.3.7 TMBCSEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

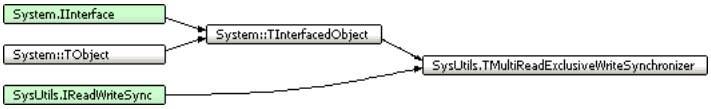
See Also

Encoding.GetPreamble (MSDN)

39.1.48 TMultiReadExclusiveWriteSynchronizer Class

TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.

Class Hierarchy



Pascal

```
TMultiReadExclusiveWriteSynchronizer = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TMultiReadExclusiveWriteSynchronizer : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

Use TMultiReadExclusiveWriteSynchronizer to guard access to memory in a multi-threaded application. Unlike a critical section, which blocks all other threads from reading or writing its associated memory, TMultiReadExclusiveWriteSynchronizer allows multiple threads to read from the protected memory simultaneously, while ensuring that any thread writing to the memory has exclusive access.

In applications where threads read from an object or variable frequently and only write to it occasionally, using the multi-read exclusive-write synchronizer instead of a critical section can result in considerable performance improvement.

All access to the protected memory must be bracketed by calls to the BeginRead (see page 4064) and EndRead (see page 4065) or BeginWrite (see page 4064) and EndWrite (see page 4065) methods. Any thread that reads from or writes to this memory without using these calls can introduce thread conflicts.

See Also

TCriticalSection (see page 3068)

TThread

Using the Multi-read Exclusive-write Synchronizer

39.1.48.1 TMultiReadExclusiveWriteSynchronizer Members

The following tables list the members exposed by TMultiReadExclusiveWriteSynchronizer.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	BeginRead (see page 4064)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 4064)	Allows a thread to write to the protected memory when it is safe to do so.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4063)	Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 4061).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 4063)	Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 4061).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	EndRead (see page 4065)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 4065)	Allows other threads to access the associated memory.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
R	RevisionLevel (see page 4066)	Changes revision level of protected memory.

39.1.48.2 TMultiReadExclusiveWriteSynchronizer.Create Constructor

Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 4061).

Pascal

```
constructor Create;
```

C++

```
__fastcall TMultiReadExclusiveWriteSynchronizer();
```

Description

Call Create to instantiate TMultiReadExclusiveWriteSynchronizer (see page 4061). The synchronizer must be created in the same thread as the memory it protects, so that it is accessible to all threads that need to access that memory.

39.1.48.3 TMultiReadExclusiveWriteSynchronizer.Destroy Destructor

Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 4061).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultiReadExclusiveWriteSynchronizer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMultiReadExclusiveWriteSynchronizer (see page 4061) reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

39.1.48.4 TMultiReadExclusiveWriteSynchronizer Methods

The methods of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4064)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 4064)	Allows a thread to write to the protected memory when it is safe to do so.
	EndRead (see page 4065)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 4065)	Allows other threads to access the associated memory.

39.1.48.4.1 TMultiReadExclusiveWriteSynchronizer.BeginRead Method

Allows a thread to read from the protected memory when it is safe to do so.

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

Call BeginRead before reading from the memory protected by the multi-read exclusive-write synchronizer. BeginRead does not return until there are no other threads writing to the memory. After calling BeginRead, other threads can be granted read access to the protected memory by the BeginRead method. However, any thread asking for write permission by calling BeginWrite (see page 4064) must wait until all reading threads have called EndRead (see page 4065).

See Also

EndRead (see page 4065)

BeginWrite (see page 4064)

Using the Multi-read Exclusive-write Synchronizer

39.1.48.4.2 TMultiReadExclusiveWriteSynchronizer.BeginWrite Method

Allows a thread to write to the protected memory when it is safe to do so.

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

BeginWrite obtains an exclusive write lock on the memory protected by the synchronizer object. Call BeginWrite before writing to this memory. BeginWrite does not return until no other threads are reading from or writing to the memory. After a thread calls BeginWrite, any other thread asking to access that memory by calling BeginRead (see page 4064) or BeginWrite must wait until the writing thread calls EndWrite (see page 4065).

Note that a write lock operation is not atomic. Data in protected memory will always be in a consistent state when BeginWrite returns, but is not necessarily in the same state as when BeginWrite was called. As a rule, a thread should always discard previous samples from protected memory after promoting a read lock to a write lock. However, the calling thread can determine whether a state change has occurred by examining return value of BeginWrite: true if protected memory has not be written to by another thread, false if another thread may have modified protected memory.

See Also

EndWrite (see page 4065)

BeginRead (see page 4064)

Using the Multi-read Exclusive-write Synchronizer

39.1.48.4.3 TMultiReadExclusiveWriteSynchronizer.EndRead Method

Removes a thread from the internal list of threads that are accessing the protected memory.

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

Call EndRead after reading from the memory protected by the multi-read exclusive-write synchronizer. Each call to BeginRead (see page 4064) must be matched by a call to EndRead, or other threads can never return from the BeginWrite (see page 4064) method.

See Also

BeginRead (see page 4064)

BeginWrite (see page 4064)

Using the Multi-read Exclusive-write Synchronizer

39.1.48.4.4 TMultiReadExclusiveWriteSynchronizer.EndWrite Method

Allows other threads to access the associated memory.

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

Call EndWrite after writing to the memory that is protected by the multi-read exclusive-write synchronizer. Each call to BeginWrite (see page 4064) must be matched by a call to EndWrite, or other threads can never return from the BeginRead (see page 4064) or BeginWrite (see page 4064) methods.

See Also

BeginWrite (see page 4064)

BeginRead (see page 4064)

Using the Multi-read Exclusive-write Synchronizer

39.1.48.5 TMultiReadExclusiveWriteSynchronizer Properties

The properties of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Properties

	Name	Description
	RevisionLevel (see page 4066)	Changes revision level of protected memory.

39.1.48.5.1 TMultiReadExclusiveWriteSynchronizer.RevisionLevel Property

Changes revision level of protected memory.

Pascal

```
property RevisionLevel: Cardinal;
```

C++

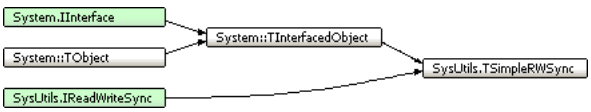
```
__property unsigned RevisionLevel;
```

Description

RevisionLevel changes each time a write lock is granted. This property should be used for identity comparisons only, not for determining revision sequence.

39.1.49 TSimpleRWSync Class

Class Hierarchy



Pascal

```
TSimpleRWSync = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TSimpleRWSync : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.TSimpleRWSync.

39.1.49.1 TSimpleRWSync Members

The following tables list the members exposed by TSimpleRWSync.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	BeginRead (see page 4069)	This is BeginRead, a member of class TSimpleRWSync.
	BeginWrite (see page 4069)	This is BeginWrite, a member of class TSimpleRWSync.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4068)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 4068)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndRead (see page 4070)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 4070)	This is EndWrite, a member of class TSimpleRWSync.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

39.1.49.2 TSimpleRWSync.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSimpleRWSync();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

39.1.49.3 TSimpleRWSync.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleRWSync();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil

before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

39.1.49.4 TSimpleRWSync Methods

The methods of the TSimpleRWSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4069)	This is BeginRead, a member of class TSimpleRWSync.
	BeginWrite (see page 4069)	This is BeginWrite, a member of class TSimpleRWSync.
	EndRead (see page 4070)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 4070)	This is EndWrite, a member of class TSimpleRWSync.

39.1.49.4.1 TSimpleRWSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class TSimpleRWSync.

39.1.49.4.2 TSimpleRWSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class TSimpleRWSync.

39.1.49.4.3 TSimpleRWSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class TSimpleRWSync.

39.1.49.4.4 TSimpleRWSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class TSimpleRWSync.

39.1.50 TStringBuilder Class

An editable array of characters whose contents has all the functionality of a string object.

Class Hierarchy

Pascal

```
TStringBuilder = class;
```

C++

```
class TStringBuilder;
```

File

SysUtils

Description

Use a TStringBuilder as an editable string.

The TStringBuilder class is completely compatable with the .Net framework's StringBuilder (MSDN) class. Use a TStringBuilder instance to hold a character array. This array can be modified after creation. Substrings can be appended, searched and replaced, or inserted. The character array can be queried by index or converted to a string for comparison.

The character array can be referenced directly by using the Chars (see page 4085) property. The Length (see page 4086) property contains the current length of the character array.

builder.Length (see page 4086) = builder.ToString.Length (see page 4086);

The Capacity (see page 4085) property holds the current allocated storage space for this TStringBuilder instance. Capacity (see page 4085) can be increased up to the value of MaxCapacity (see page 4086) and down to the current value of Length (

see page 4086). Capacity (see page 4085) is increased if Length (see page 4086) is increased, or the character array is altered by appending to increase it's length.

The Append (see page 4078) method converts a multitude of possible types to a string and appends that to the current character array. AppendFormat (see page 4079) allows you to append several objects to the character array, each object with it's own format.

The Equals (see page 4081) method compares a specified object or another TStringBuilder to this instance of TStringBuilder.

The Append (see page 4078), CopyTo (see page 4080), Insert (see page 4082), Remove (see page 4083) and Replace (see page 4084) methods access the character array by index or search the character as a string. The ToString (see page 4084) method converts the character array to a string to provide string functionality.

TStringBuilder is an implementation of the .NET StringBuilder (MSDN) class.

C++ Examples:

```
/*
This example exercises many of the members of the TStringBuilder class.
*/
template <class T>
void testAppend(T t)
{
    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
    builder->Append(t);
    UnicodeString s1 = builder->ToString();
    UnicodeString s2(t);
    assert(s1 == s2);
}


#if defined(_DELPHI_STRING_UNICODE)
const wchar_t* ext_chars =
    L"Ð• Ð‘ Ð’ Ð“ Ð„ Ð… Ð† Ð‡ Ð†"
    L"Ðš Ð› Ðœ Ð• Ðž ÐŸ Ò€ Ð  Ð¡ Ð¢ Ñ¸"
    L"Ð¤ Ð¥ Ñ  Ð¦ Ð§ Ð¨ Ð© Ðª Ð« Ð¬ Ñ¢"
    L"Ð® Ð†Ð• Ñ¦ Ñ¨ Ñª Ñ¬ Ñ® Ñ° Ñ² Ñ´ Ñ¤";
#else
const char* ext_chars =
    "Ã€ Ã• Ã‚ Ãƒ Ã„ Ã… Ã† Ã‡ Ãˆ Ã‰ Ã Ã‹ Ã• ÃŽ Ã•"
    "Ã• Ã‘ Ã’ Ã“ Ã” Ã• Ã– Ã— Ã˜ Ã™ Ãš Ã› Ãœ Ã• Ãž ÃŸ"
    "Ã  Ã¡ Ã¢ Ã£ Ã¤ Ã¥ Ã¦ Ã§ Ã¨ Ã© Ãª Ã« Ã¬ Ã- Ã® Ã¯"
    "Ã° Ã± Ã² Ã³ Ã´ Ãµ Ã¶ Ã· Ã¸ Ã¹ Ãº Ã» Ã¼ Ã½ Ã¾ Ã¿";
#endif

//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    {
        // Test starts out as empty
        std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
        assert(builder->ToString() == System::String());
    }

    {
        // Append
        short s = 0xffe;
        testAppend(s);

        unsigned short us = 0xfffe;
        testAppend(us);
    }

    {
        const _DCHAR* p = ext_chars;
```

```
        std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
        System::String str1(p);

        // Appending of wchar_t
        while (*p)
        {
            builder->Append(*p++);
        }

        // Length property
        assert(builder->Length == str1.Length());

        // Subscript operator
        for (int i=0; i<str1.Length(); i++)
        {
            // NOTE: strings are 1-based
            assert((*(builder.get()))[i] == str1[i+1]);
        }

        // ToString()
        System::String str2 = builder->ToString();
        Edit1->Text = str1;
        Edit2->Text = str2;
        assert(str2 == str1);

        // Replace
        builder->Replace(' ', '.');
        UnicodeString str3;
        str3 = builder->ToString();
        Edit3->Text = str3;
        assert(str3 != str2);

        // Replace back
        builder->Replace(L'.', L' ');
        System::String str4 = builder->ToString();
        Edit4->Text = str4;
        assert(str2 == str4);
    }

    {
        System::String str(ext_chars);

        std::auto_ptr<TStringBuilder> builder(new TStringBuilder());

        // Insert at start when empty, at start when not empty and at end
        builder->Insert(0, str);
        builder->Insert(0, str);
        builder->Insert(builder->Length, str);

        // Concatenate string 3 times
        System::String str2;
        str2 += str;
        str2 += str;
        str2 += str;

        str = builder->ToString();

        Edit5->Text = str;
        assert(builder->ToString() == str2);
    }
    MessageDlg("All assertions passed in the TStringBuilder Test test.",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
```

```
{
    const _DCHAR* p = ext_chars;

    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());

    // Appending of chars
    while (*p)
    {
        builder->Append(*p++);
    }

    System::String str1(ext_chars);
    System::String str2 = builder->ToString();
    assert(str1 == str2);

    builder->Replace(_D(" "), _D("."));
    System::String str3;
    str3 = builder->ToString();
    assert(str1 != str3);

    builder->Replace(_D("."), _D(" "));
    System::String str4(builder->ToString());
    assert(str1 == str4);
    MessageDlg("All assertions passed in the TStringBuilder Replace test.",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

See Also

Append (see page 4078)

AppendFormat (see page 4079)

AppendLine (see page 4080)

CopyTo (see page 4080)

EnsureCapacity (see page 4080)

Equals (see page 4081)

Insert (see page 4082)

Remove (see page 4083)

Replace (see page 4084)

ToString (see page 4084)

Capacity (see page 4085)

Chars (see page 4085)

Length (see page 4086)

MaxCapacity (see page 4086)

Andreano Lanusse's blog about TStringBuilder

StringBuilder (MSDN)

39.1.50.1 TStringBuilder Members

The following tables list the members exposed by TStringBuilder.

Protected Fields

	Name	Description
	FData (see page 4076)	This is FData, a member of class TStringBuilder.
	FLength (see page 4076)	This is FLength, a member of class TStringBuilder.
	FMaxCapacity (see page 4076)	This is FMaxCapacity, a member of class TStringBuilder.

Public Methods

	Name	Description
	Append (see page 4077)	This is the overview for the Append method overload.
	AppendFormat (see page 4079)	Appends a number of objects to the current character array in this TStringBuilder (see page 4070) instance.
	AppendLine (see page 4079)	This is the overview for the AppendLine method overload.
	CopyTo (see page 4080)	Copys a part of the instance character array into a destination string.
	Create (see page 4074)	This is the overview for the Create constructor overload.
	EnsureCapacity (see page 4080)	Sets the Capacity (see page 4085) property to the supplied value, if the new capacity is larger than the old capacity.
	Equals (see page 4081)	Compares the character array in the Chars (see page 4085) property, the Capacity (see page 4085) property, and the MaxCapacity (see page 4086) property of this instance to that of a supplied TStringBuilder (see page 4070) instance.
	Insert (see page 4081)	This is the overview for the Insert method overload.
	Remove (see page 4083)	Removes a substring from the character array of this instance.
	Replace (see page 4083)	This is the overview for the Replace method overload.
V	ToString (see page 4084)	This is the overview for the ToString method overload.

Public Properties

	Name	Description
	Capacity (see page 4085)	Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4070) instance.
	Chars (see page 4085)	Used to get or set the characters stored in this TStringBuilder (see page 4070) instance.
	Length (see page 4086)	Used to get or set the size of the character array stored in this TStringBuilder (see page 4070) instance.
R	MaxCapacity (see page 4086)	Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4070) instance.

39.1.50.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TStringBuilder.Create () (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).
	TStringBuilder.Create (Integer) (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).
	TStringBuilder.Create (Integer, Integer) (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).
	TStringBuilder.Create (string) (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).
	TStringBuilder.Create (string, Integer) (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).

	TStringBuilder.Create (string, Integer, Integer, Integer) (see page 4075)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).

39.1.50.2.1 TStringBuilder.Create Constructor ()

Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4070).

Pascal

```
constructor Create; overload;
constructor Create(aCapacity: Integer); overload;
constructor Create(const Value: string); overload;
constructor Create(aCapacity: Integer; aMaxCapacity: Integer); overload;
constructor Create(const Value: string; aCapacity: Integer); overload;
constructor Create(const Value: string; StartIndex: Integer; Length: Integer; aCapacity:
Integer); overload;
```

C++

```
__fastcall TStringBuilder();
__fastcall TStringBuilder(int aCapacity);
__fastcall TStringBuilder(const UnicodeString Value);
__fastcall TStringBuilder(int aCapacity, int aMaxCapacity);
__fastcall TStringBuilder(const UnicodeString Value, int aCapacity);
__fastcall TStringBuilder(const UnicodeString Value, int StartIndex, int Length, int
aCapacity);
```

Description

Use Create to Create an instance of the TStringBuilder (see page 4070) class.

aCapacity specifies the initial value of the Capacity (see page 4085) property, the maximum number of characters this TStringBuilder (see page 4070) instance can hold. The value of the Capacity (see page 4085) property can be changed.

Value specifies the initial character array to be stored in this TStringBuilder (see page 4070) instance.

aMaxCapacity is the maximum value that the Capacity (see page 4085) property can be for this instance of the TStringBuilder (see page 4070) class.

StartIndex specifies the index of the first character in the Value string to be copied into the initial character array of this TStringBuilder (see page 4070) instance.

Length (see page 4086) is the number of characters to copy from the Value string into the initial character array of this TStringBuilder (see page 4070) instance.

See Also

Capacity

StringBuilder.Create (MSDN)

39.1.50.3 TStringBuilder Fields

The fields of the TStringBuilder class are listed here.

Protected Fields

	Name	Description
	FData (see page 4076)	This is FData, a member of class TStringBuilder.
	FLength (see page 4076)	This is FLength, a member of class TStringBuilder.
	FMaxCapacity (see page 4076)	This is FMaxCapacity, a member of class TStringBuilder.

39.1.50.3.1 TStringBuilder.FData Field

Pascal

```
FData: TCharArray;
```

C++

```
TCharArray FData;
```

Description

This is FData, a member of class TStringBuilder.

39.1.50.3.2 TStringBuilder.FLength Field

Pascal

```
FLength: Integer;
```

C++

```
int FLength;
```

Description

This is FLength, a member of class TStringBuilder.

39.1.50.3.3 TStringBuilder.FMaxCapacity Field

Pascal

```
FMaxCapacity: Integer;
```

C++

```
int FMaxCapacity;
```

Description

This is FMaxCapacity, a member of class TStringBuilder.

39.1.50.4 TStringBuilder Methods

The methods of the TStringBuilder class are listed here.

Public Methods

	Name	Description
	Append (see page 4077)	This is the overview for the Append method overload.
	AppendFormat (see page 4079)	Appends a number of objects to the current character array in this TStringBuilder (see page 4070) instance.
	AppendLine (see page 4079)	This is the overview for the AppendLine method overload.
	CopyTo (see page 4080)	Copys a part of the instance character array into a destination string.
	EnsureCapacity (see page 4080)	Sets the Capacity (see page 4085) property to the supplied value, if the new capacity is larger than the old capacity.
	Equals (see page 4081)	Compares the character array in the Chars (see page 4085) property, the Capacity (see page 4085) property, and the MaxCapacity (see page 4086) property of this instance to that of a supplied TStringBuilder (see page 4070) instance.
	Insert (see page 4081)	This is the overview for the Insert method overload.
	Remove (see page 4083)	Removes a substring from the character array of this instance.

	Replace (see page 4083)	This is the overview for the Replace method overload.
	ToString (see page 4084)	This is the overview for the ToString method overload.

39.1.50.4.1 Append Method

This is the overview for the Append method overload.

Overload List

	Name	Description
	TStringBuilder.Append (Boolean) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Byte) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Cardinal) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Char) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Char, Integer) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Currency) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Double) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Int64) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Integer) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Shortint) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Single) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Smallint) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (string) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (string, Integer, Integer) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.

	TStringBuilder.Append (TCharArray) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (TCharArray, Integer, Integer) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (TObject) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (UInt64) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.
	TStringBuilder.Append (Word) (see page 4078)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.

39.1.50.4.1.1 TStringBuilder.Append Method (Boolean)

Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.

Pascal

```
function Append(const Value: Boolean): TStringBuilder; overload;
function Append(const Value: Byte): TStringBuilder; overload;
function Append(const Value: Char): TStringBuilder; overload;
function Append(const Value: Currency): TStringBuilder; overload;
function Append(const Value: Double): TStringBuilder; overload;
function Append(const Value: Smallint): TStringBuilder; overload;
function Append(const Value: Integer): TStringBuilder; overload;
function Append(const Value: Int64): TStringBuilder; overload;
function Append(const Value: TObject): TStringBuilder; overload;
function Append(const Value: Shortint): TStringBuilder; overload;
function Append(const Value: Single): TStringBuilder; overload;
function Append(const Value: string): TStringBuilder; overload;
function Append(const Value: UInt64): TStringBuilder; overload;
function Append(const Value: TCharArray): TStringBuilder; overload;
function Append(const Value: Word): TStringBuilder; overload;
function Append(const Value: Cardinal): TStringBuilder; overload;
function Append(const Value: Char; RepeatCount: Integer): TStringBuilder; overload;
function Append(const Value: TCharArray; StartIndex: Integer; CharCount: Integer):
TStringBuilder; overload;
function Append(const Value: string; StartIndex: Integer; Count: Integer): TStringBuilder;
overload;
```

C++

```
__fastcall TStringBuilder Append(const Boolean Value);
__fastcall TStringBuilder Append(const Byte Value);
__fastcall TStringBuilder Append(const Char Value);
__fastcall TStringBuilder Append(const Currency Value);
__fastcall TStringBuilder Append(const Double Value);
__fastcall TStringBuilder Append(const Smallint Value);
__fastcall TStringBuilder Append(const int Value);
__fastcall TStringBuilder Append(const Int64 Value);
__fastcall TStringBuilder Append(const TObject * Value);
__fastcall TStringBuilder Append(const Shortint Value);
__fastcall TStringBuilder Append(const Single Value);
__fastcall TStringBuilder Append(const UnicodeString Value);
__fastcall TStringBuilder Append(const UInt64 Value);
__fastcall TStringBuilder Append(const TCharArray Value);
```

```
__fastcall TStringBuilder Append(const Word Value);
__fastcall TStringBuilder Append(const unsigned Value);
__fastcall TStringBuilder Append(const Char Value, int RepeatCount);
__fastcall TStringBuilder Append(const TCharArray Value, int StartIndex, int CharCount);
__fastcall TStringBuilder Append(const UnicodeString Value, int StartIndex, int Count);
```

Description

Use Append to append a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4070) instance.

In the case where **Value** is a Char, **Value** is a Unicode character and repeatedly appended to the end of theTStringBuilder string **Count** number of times.

The two overloads that use the parameters **CharCount** or **Count** append a substring. **StartIndex** indicates the first character is the string **Value** to append, and **CharCount** or **Count** specifies the number of characters to append.

See Also

StringBuilder.Append (MSDN)

39.1.50.4.2 TStringBuilder.AppendFormat Method

Appends a number of objects to the current character array in this TStringBuilder (see page 4070) instance.

Pascal

```
function AppendFormat(const Format: string; const Args: array of const): TStringBuilder;
overload;
```

C++

```
__fastcall TStringBuilder AppendFormat(const UnicodeString Format, const array of const Args);
```

Description

Use AppendFormat to append a number of objects to the current character array in this TStringBuilder (see page 4070) instance. The objects are represented by a string of format specifications and an array of objects to format. There is a one to one relationship between the format specifications and the objects in the objects array.

Format (see page 4232) is a string of format specifications.

Args is the corresponding array of objects to format.

See Also

StringBuilder.AppendFormat (MSDN)

39.1.50.4.3 AppendLine Method

This is the overview for the AppendLine method overload.

Overload List

	Name	Description
	TStringBuilder.AppendLine () (see page 4080)	Appends a line terminator or a string and a line terminator to the current character array in this instance.
	TStringBuilder.AppendLine (string) (see page 4080)	Appends a line terminator or a string and a line terminator to the current character array in this instance.

39.1.50.4.3.1 TStringBuilder.AppendLine Method ()

Appends a line terminator or a string and a line terminator to the current character array in this instance.

Pascal

```
function AppendLine: TStringBuilder; overload;
function AppendLine(const Value: string): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder AppendLine();
__fastcall TStringBuilder AppendLine(const UnicodeString Value);
```

Description

Use AppendLine to append a line terminator or a string and a line terminator to the current character array in this instance.

See Also

StringBuilder.AppendLine (MSDN)

39.1.50.4.4 TStringBuilder.CopyTo Method

Copys a part of the instance character array into a destination string.

Pascal

```
procedure CopyTo(SourceIndex: Integer; const Destination: TCharArray; DestinationIndex:
Integer; Count: Integer);
```

C++

```
__fastcall CopyTo(int SourceIndex, const TCharArray Destination, int DestinationIndex, int
Count);
```

Description

Use CopyTo to copy a part of the instance character array into a destination string.

SourceIndex specifies the first character in the instance character array to copy.

Destination indicates the character array to copy to.

DestinationIndex specifies the position in the destination string to insert the substring.

Count is the number of characters to copy.

See Also

StringBuilder.CopyTo (MSDN)

39.1.50.4.5 TStringBuilder.EnsureCapacity Method

Sets the Capacity (see page 4085) property to the supplied value, if the new capacity is larger than the old capacity.

Pascal

```
function EnsureCapacity(aCapacity: Integer): Integer;
```

C++

```
__fastcall int EnsureCapacity(int aCapacity);
```

Description

Sets the Capacity (see page 4085) property to the supplied value, if the new capacity is larger than the old capacity. Also

allocates more memory for the character array if needed. Returns the new capacity.

See Also

Capacity (see page 4085)

StringBuilder.EnsureCapacity (MSDN)

39.1.50.4.6 TStringBuilder.Equals Method

Compares the character array in the Chars (see page 4085) property, the Capacity (see page 4085) property, and the MaxCapacity (see page 4086) property of this instance to that of a supplied TStringBuilder (see page 4070) instance.

Pascal

```
function Equals(StringBuilder: TStringBuilder): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TStringBuilder StringBuilder);
```

Description

Use Equals to compare the character array in the Chars (see page 4085) property, the Capacity (see page 4085) property, and the MaxCapacity (see page 4086) property of this instance to that of a supplied TStringBuilder (see page 4070) instance.

See Also

Capacity (see page 4085)

Chars (see page 4085)

MaxCapacity (see page 4086)

StringBuilder.Equals (MSDN)

39.1.50.4.7 Insert Method

This is the overview for the Insert method overload.

Overload List

	Name	Description
	TStringBuilder.Insert (Integer, Boolean) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Byte) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Cardinal) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Char) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Currency) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Double) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Int64) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.

	TStringBuilder.Insert (Integer, Integer) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Shortint) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Single) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Smallint) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, string) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, string, Integer) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TCharArray) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TCharArray, Integer, Integer) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TObject) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, UInt64) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Word) (see page 4082)	Inserts a string representation of the object parameter at the Index provided.

39.1.50.4.7.1 TStringBuilder.Insert Method (Integer, Boolean)

Inserts a string representation of the object parameter at the Index provided.

Pascal

```
function Insert(Index: Integer; const Value: Boolean): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Byte): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Char): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Currency): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Double): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Smallint): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Integer): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TCharArray): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Int64): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TObject): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Shortint): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Single): TStringBuilder; overload;
function Insert(Index: Integer; const Value: string): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Word): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Cardinal): TStringBuilder; overload;
function Insert(Index: Integer; const Value: UInt64): TStringBuilder; overload;
function Insert(Index: Integer; const Value: string; count: Integer): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TCharArray; startIndex: Integer; charCount:
Integer): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder Insert(int Index, const Boolean Value);
__fastcall TStringBuilder Insert(int Index, const Byte Value);
__fastcall TStringBuilder Insert(int Index, const Char Value);
__fastcall TStringBuilder Insert(int Index, const Currency Value);
__fastcall TStringBuilder Insert(int Index, const Double Value);
__fastcall TStringBuilder Insert(int Index, const Smallint Value);
```

```
__fastcall TStringBuilder Insert(int Index, const int Value);
__fastcall TStringBuilder Insert(int Index, const TCharArray Value);
__fastcall TStringBuilder Insert(int Index, const Int64 Value);
__fastcall TStringBuilder Insert(int Index, const TObject * Value);
__fastcall TStringBuilder Insert(int Index, const Shortint Value);
__fastcall TStringBuilder Insert(int Index, const Single Value);
__fastcall TStringBuilder Insert(int Index, const UnicodeString Value);
__fastcall TStringBuilder Insert(int Index, const Word Value);
__fastcall TStringBuilder Insert(int Index, const unsigned Value);
__fastcall TStringBuilder Insert(int Index, const UInt64 Value);
__fastcall TStringBuilder Insert(int Index, const UnicodeString Value, int count);
__fastcall TStringBuilder Insert(int Index, const TCharArray Value, int startIndex, int
charCount);
```

Description

Use Insert to insert a string representation of the object parameter at the Index provided.

In the overload that uses the parameter **count**, **count** provides a way to insert the object **count** number of times.

startIndex and **charCount** provide a way to insert a substring of the **Value** TCharArray (see page 4381) into the character array of this instance.

See Also

StringBuilder.Insert (MSDN)

39.1.50.4.8 TStringBuilder.Remove Method

Removes a substring from the character array of this instance.

Pascal

```
function Remove(StartIndex: Integer; RemLength: Integer): TStringBuilder;
```

C++

```
__fastcall TStringBuilder Remove(int StartIndex, int RemLength);
```

Description

Use Remove to remove a substring from the character array of this instance.

StartIndex indicates the index of the first character to remove.

RemLength indicates the number of characters to remove.

See Also

StringBuilder.Remove (MSDN)

39.1.50.4.9 Replace Method

This is the overview for the Replace method overload.

Overload List

	Name	Description
	TStringBuilder.Replace (Char, Char) (see page 4084)	Replaces all occurences of a specified string with another.
	TStringBuilder.Replace (Char, Char, Integer, Integer) (see page 4084)	Replaces all occurences of a specified string with another.

	TStringBuilder.Replace (string, string) (see page 4084)	Replaces all occurences of a specified string with another.
	TStringBuilder.Replace (string, string, Integer, Integer) (see page 4084)	Replaces all occurences of a specified string with another.

39.1.50.4.9.1 TStringBuilder.Replace Method (Char, Char)

Replaces all occurences of a specified string with another.

Pascal

```
function Replace(const OldChar: Char; const NewChar: Char): TStringBuilder; overload;
function Replace(const OldValue: string; const NewValue: string): TStringBuilder; overload;
function Replace(const OldChar: Char; const NewChar: Char; StartIndex: Integer; Count:
Integer): TStringBuilder; overload;
function Replace(const OldValue: string; const NewValue: string; StartIndex: Integer; Count:
Integer): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder Replace(const Char OldChar, const Char NewChar);
__fastcall TStringBuilder Replace(const UnicodeString OldValue, const UnicodeString NewValue);
__fastcall TStringBuilder Replace(const Char OldChar, const Char NewChar, int StartIndex, int
Count);
__fastcall TStringBuilder Replace(const UnicodeString OldValue, const UnicodeString NewValue,
int StartIndex, int Count);
```

Description

Use Replace to replace all occurences of a specified string with another. Replace can replace a single **OldChar** with a **NewChar** everywhere, an **OldValue string** with a **NewValue** string, an **OldChar** with a **NewChar** everywhere in a specified substring, or an **OldValue** string with a **NewValue** string everywhere in a specified substring.

See Also

StringBuilder.Replace (MSDN)

39.1.50.4.10 ToString Method

This is the overview for the ToString method overload.

Overload List

	Name	Description
	TStringBuilder.ToString () (see page 4084)	Returns a string representation of the character array contained in this TStringBuilder (see page 4070) instance.
	TStringBuilder.ToString (Integer, Integer) (see page 4084)	Returns a string representation of the character array contained in this TStringBuilder (see page 4070) instance.

39.1.50.4.10.1 TStringBuilder.ToString Method ()

Returns a string representation of the character array contained in this TStringBuilder (see page 4070) instance.

Pascal

```
function ToString: string; override; overload;
function ToString(StartIndex: Integer; StrLength: Integer): string; overload; reintroduce;
```

C++

```
virtual __fastcall UnicodeString ToString();
__fastcall UnicodeString ToString(int StartIndex, int StrLength);
```

Description

Use ToString to create a string representation of the character array contained in this TStringBuilder (see page 4070) instance.

StartIndex and **StrLength** specify a substing of the character array to convert.

See Also

StringBuilder.ToString (MSDN)

39.1.50.5 TStringBuilder Properties

The properties of the TStringBuilder class are listed here.

Public Properties

	Name	Description
	Capacity (see page 4085)	Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4070) instance.
	Chars (see page 4085)	Used to get or set the characters stored in this TStringBuilder (see page 4070) instance.
	Length (see page 4086)	Used to get or set the size of the character array stored in this TStringBuilder (see page 4070) instance.
R	MaxCapacity (see page 4086)	Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4070) instance.

39.1.50.5.1 TStringBuilder.Capacity Property

Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4070) instance.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Use Capacity to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4070) instance. Capacity cannot be set higher than the value of MaxCapacity (see page 4086) and cannot be set lower than the current value of Length (see page 4086).

See Also

MaxCapacity (see page 4086)

Length (see page 4086)

StringBuilder.Capacity (MSDN)

39.1.50.5.2 TStringBuilder.Chars Property

Used to get or set the characters stored in this TStringBuilder (see page 4070) instance.

Pascal

```
property Chars [index: Integer]: Char;
```

C++

```
__property Char Chars[int index];
```

Description

Use Chars to get or set the characters stored in this TStringBuilder (see page 4070) instance.

index specifies the position of the character to access.

The value of this property is the character stored at location index.

See Also

StringBuilder.Chars (MSDN)

39.1.50.5.3 TStringBuilder.Length Property

Used to get or set the size of the character array stored in this TStringBuilder (see page 4070) instance.

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

Use Length to get or set the size of the character array stored in this TStringBuilder (see page 4070) instance. If the value of **Length** is lowered, the character array is truncated. If the value is raised, the character array is padded with spaces and the Capacity (see page 4085) property is increased. An error occurs if an attempt is made to increase **Length** larger than the MaxCapacity (see page 4086) property.

See Also

Capacity (see page 4085)

MaxCapacity (see page 4086)

StringBuilder.Length (MSDN)

39.1.50.5.4 TStringBuilder.MaxCapacity Property

Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4070) instance.

Pascal

```
property MaxCapacity: Integer;
```

C++

```
__property int MaxCapacity;
```

Description

Use MaxCapacity to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4070) instance. **MaxCapacity** is set when the instance of TStringBuilder (see page 4070) is created and determines the maximum value to which the Capacity (see page 4085) property can be set.

See Also

Capacity (see page 4085)

StringBuilder.MaxCapacity (MSDN)

39.1.51 TThreadLocalCounter Class

Class Hierarchy

Pascal

```
TThreadLocalCounter = class;
```

C++

```
class TThreadLocalCounter;
```

File

SysUtils

Description

This is class SysUtils.TThreadLocalCounter.

39.1.51.1 TThreadLocalCounter Members

The following tables list the members exposed by TThreadLocalCounter.

Public Methods

	Name	Description
	Close (see page 4088)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 4088)	This is Delete, a member of class TThreadLocalCounter.
	Destroy (see page 4087)	This is Destroy, a member of class TThreadLocalCounter.
	Open (see page 4088)	This is Open, a member of class TThreadLocalCounter.

39.1.51.2 TThreadLocalCounter.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThreadLocalCounter();
```

Description

This is Destroy, a member of class TThreadLocalCounter.

39.1.51.3 TThreadLocalCounter Methods

The methods of the TThreadLocalCounter class are listed here.

Public Methods

	Name	Description
	Close (see page 4088)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 4088)	This is Delete, a member of class TThreadLocalCounter.
	Open (see page 4088)	This is Open, a member of class TThreadLocalCounter.

39.1.51.3.1 TThreadLocalCounter.Close Method

Pascal

```
procedure Close(var Thread: PThreadInfo);
```

C++

```
__fastcall Close(PThreadInfo Thread);
```

Description

This is Close, a member of class TThreadLocalCounter.

39.1.51.3.2 TThreadLocalCounter.Delete Method

Pascal

```
procedure Delete(var Thread: PThreadInfo);
```

C++

```
__fastcall Delete(PThreadInfo Thread);
```

Description

This is Delete, a member of class TThreadLocalCounter.

39.1.51.3.3 TThreadLocalCounter.Open Method

Pascal

```
procedure Open(var Thread: PThreadInfo);
```

C++

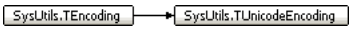
```
__fastcall Open(PThreadInfo Thread);
```

Description

This is Open, a member of class TThreadLocalCounter.

39.1.52 TUnicodeEncoding Class

Class Hierarchy



Pascal

```
TUnicodeEncoding = class(TEncoding);
```

C++

```
class TUnicodeEncoding : public TEncoding;
```

File

SysUtils

Description

This is class SysUtils.TUnicodeEncoding.

39.1.52.1 TUnicodeEncoding Members

The following tables list the members exposed by TUnicodeEncoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
V	Create (see page 4090)	This is Create, a member of class TUnicodeEncoding.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4092)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4092)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4093)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4090)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4091)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4091)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 4092)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.52.2 TUnicodeEncoding.Create Constructor

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TUnicodeEncoding();
```

Description

This is Create, a member of class TUnicodeEncoding.

39.1.52.3 TUnicodeEncoding Methods

The methods of the TUnicodeEncoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 4092)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4092)	Calculates the maximum number of characters generated to decode a specified number of bytes.
	GetPreamble (see page 4093)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4090)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 4091)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 4091)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 4092)	Decodes a set of bytes into a set of characters.

39.1.52.3.1 TUnicodeEncoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

39.1.52.3.2 TUnicodeEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

39.1.52.3.3 TUnicodeEncoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

39.1.52.3.4 TUnicodeEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

39.1.52.3.5 TUnicodeEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

39.1.52.3.6 TUnicodeEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

39.1.52.3.7 TUnicodeEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

39.1.53 TUTF7Encoding Class

Class Hierarchy

Pascal

```
TUTF7Encoding = class(TMBCSEncoding);
```

C++

```
class TUTF7Encoding : public TMBCSEncoding;
```

File

SysUtils

Description

This is class SysUtils.TUTF7Encoding.

39.1.53.1 TUTF7Encoding Members

The following tables list the members exposed by TUTF7Encoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
V	Create (see page 4094)	This is Create, a member of class TUTF7Encoding.
V	Create (see page 4057)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4097)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4097)	Calculates the maximum number of characters generated to decode a specified number of bytes.
A	GetPreamble (see page 4047)	Returns a preamble in a byte array that specifies the encoding used.
V	GetPreamble (see page 4061)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4095)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4095)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4096)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 4096)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.53.2 TUTF7Encoding.Create Constructor

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TUTF7Encoding();
```

Description

This is Create, a member of class TUTF7Encoding.

39.1.53.3 TUTF7Encoding Methods

The methods of the TUTF7Encoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 4097)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4097)	Calculates the maximum number of characters generated to decode a specified number of bytes.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4095)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 4095)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 4096)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 4096)	Decodes a set of bytes into a set of characters.

39.1.53.3.1 TUTF7Encoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

39.1.53.3.2 TUTF7Encoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

39.1.53.3.3 TUTF7Encoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

39.1.53.3.4 TUTF7Encoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

39.1.53.3.5 TUTF7Encoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

39.1.53.3.6 TUTF7Encoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

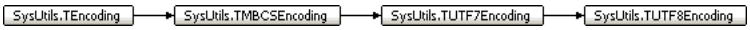
The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

39.1.54 TUTF8Encoding Class

Class Hierarchy



Pascal

```
TUTF8Encoding = class(TUTF7Encoding);
```

C++

```
class TUTF8Encoding : public TUTF7Encoding;
```

File

SysUtils

Description

This is class SysUtils.TUTF8Encoding.

39.1.54.1 TUTF8Encoding Members

The following tables list the members exposed by TUTF8Encoding.

Public Methods

	Name	Description
	Convert (see page 4040)	This is the overview for the Convert method overload.
V	Create (see page 4099)	This is Create, a member of class TUTF8Encoding.
V	Create (see page 4057)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 4041)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 4042)	Determines the encoding for a given buffer.
	GetEncoding (see page 4046)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4099)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4100)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4100)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 4047)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 4048)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 4048)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 4049)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 4049)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 4049)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 4050)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 4050)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 4050)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 4039)	This is FIsSingleByte, a member of class TEncoding.

	FMaxCharSize (see page 4040)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4058)	Returns the number of bytes generated by encoding Chars.
V	GetByteCount (see page 4095)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 4042)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4059)	Encodes a set of characters into a sequence of bytes.
V	GetBytes (see page 4095)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 4043)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4059)	Calcuates a number of characters generated by decoding a sequence of bytes.
V	GetCharCount (see page 4096)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 4044)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 4060)	Decodes a set of bytes into a set of characters.
V	GetChars (see page 4096)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 4045)	This is the overview for the GetChars method overload.

39.1.54.2 TUTF8Encoding.Create Constructor

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TUTF8Encoding();
```

Description

This is Create, a member of class TUTF8Encoding.

39.1.54.3 TUTF8Encoding Methods

The methods of the TUTF8Encoding class are listed here.

Public Methods

	Name	Description
V	GetMaxByteCount (see page 4099)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4100)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4100)	Returns a preamble in a byte array that specifies the encoding used.

39.1.54.3.1 TUTF8Encoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

39.1.54.3.2 TUTF8Encoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

39.1.54.3.3 TUTF8Encoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

39.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IReadWriteSync (see page 4101)	This is class SysUtils.IReadWriteSync.

39.2.1 IReadWriteSync Interface

Class Hierarchy

SysUtils.IReadWriteSync

Pascal

```
IReadWriteSync = interface;
```

C++

```
__interface IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.IReadWriteSync.

39.2.1.1 IReadWriteSync Members

The following tables list the members exposed by IReadWriteSync.

Public Methods

	Name	Description
	BeginRead (see page 4102)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 4102)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 4102)	This is EndRead, a member of class IReadWriteSync.
	EndWrite (see page 4102)	This is EndWrite, a member of class IReadWriteSync.

39.2.1.2 IReadWriteSync Methods

The methods of the IReadWriteSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4102)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 4102)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 4102)	This is EndRead, a member of class IReadWriteSync.

	EndWrite (see page 4102)	This is EndWrite, a member of class IReadWriteSync.

39.2.1.2.1 IReadWriteSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class IReadWriteSync.

39.2.1.2.2 IReadWriteSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class IReadWriteSync.

39.2.1.2.3 IReadWriteSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class IReadWriteSync.

39.2.1.2.4 IReadWriteSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class IReadWriteSync.

39.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Abort (see page 4111)	Raises a silent exception.
	AddExitProc (see page 4112)	Adds a procedure to the exit list.
	AddTerminateProc (see page 4112)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 4113)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 4113)	Compares file names based on the current locale.
	AnsiCompareStr (see page 4114)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 4115)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 4115)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 4116)	Converts a quoted string into an unquoted string.
	AnsiExtractQuotedStr (see page 4116)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 4116)	Returns a pointer to the last character in a string.
	AnsiLastChar (see page 4116)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 4117)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 4117)	Converts a file name to lower case.
	AnsiPos (see page 4118)	Locates the position of a substring.
	AnsiQuotedStr (see page 4118)	Returns the quoted version of a string.
	AnsiSameStr (see page 4119)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 4119)	Compares strings based on the current locale without case sensitivity.
	AnsiStrAlloc (see page 4120)	This is function SysUtils.AnsiStrAlloc.
	AnsiStrComp (see page 4120)	Compares null-terminated character strings.
	AnsiStrComp (see page 4120)	Compares null-terminated character strings.
	AnsiStrIComp (see page 4121)	Compares null terminated character strings case insensitively.
	AnsiStrIComp (see page 4121)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 4121)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLastChar (see page 4121)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 4122)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLComp (see page 4122)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 4122)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLIComp (see page 4122)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 4123)	Converts all characters in a null-terminated string to lower case.
	AnsiStrLower (see page 4123)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 4124)	Returns a pointer to the first occurrence of a substring.
	AnsiStrPos (see page 4124)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 4124)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrRScan (see page 4124)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 4125)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrScan (see page 4125)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 4125)	Converts all characters in a null-terminated string to upper case.
	AnsiStrUpper (see page 4125)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 4126)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 4126)	Converts a file name to upper case.
	AppendStr (see page 4127)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 4127)	Assigns a new dynamically allocated string to a specified pointer.

	Beep (see page 4127)	Emits a message beep.
	BoolToStr (see page 4130)	Converts a Boolean value to a string.
	ByteLength (see page 4132)	This is function SysUtils.ByteLength.
	BytesOf (see page 4132)	This is function SysUtils.BytesOf.
	BytesOf (see page 4133)	This is function SysUtils.BytesOf.
	BytesOf (see page 4133)	This is function SysUtils.BytesOf.
	BytesOf (see page 4133)	This is function SysUtils.BytesOf.
	ByteToCharIndex (see page 4134)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharIndex (see page 4134)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 4134)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteToCharLen (see page 4134)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 4135)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	ByteType (see page 4135)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 4136)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 4136)	Changes the extension of a file name.
	ChangeFilePath (see page 4137)	Changes the path of a file name.
	CharInSet (see page 4138)	This is function SysUtils.CharInSet.
	CharInSet (see page 4138)	This is function SysUtils.CharInSet.
	CharLength (see page 4139)	Returns number of bytes used by a character.
	CharLength (see page 4139)	Returns number of bytes used by a character.
	CharToByteIndex (see page 4139)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteIndex (see page 4139)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 4140)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToByteLen (see page 4140)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToElementIndex (see page 4140)	This is function SysUtils.CharToElementIndex.
	CharToElementIndex (see page 4140)	This is function SysUtils.CharToElementIndex.
	CharToElementLen (see page 4141)	This is function SysUtils.CharToElementLen.
	CharToElementLen (see page 4141)	This is function SysUtils.CharToElementLen.
	CheckWin32Version (see page 4141)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3165).
	CompareMem (see page 4142)	Performs a binary comparison of two memory images.
	CompareStr (see page 4142)	Compares two strings case sensitively.
	CompareStr (see page 4142)	Compares two strings case sensitively.
	CompareText (see page 4143)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 4143)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 4147)	Creates a new directory.
	CreateGUID (see page 4148)	Creates a globally unique identifier.
	CurrentYear (see page 4150)	Returns the current year.
	CurrToStr (see page 4150)	Formats a Currency (see page 3242) value as a string.
	CurrToStr (see page 4150)	Formats a Currency (see page 3242) value as a string.
	CurrToStrF (see page 4151)	Converts a Currency (see page 3242) value to a string, using a specified format.
	CurrToStrF (see page 4151)	Converts a Currency (see page 3242) value to a string, using a specified format.
	Date (see page 4151)	Returns the current date.
	DateTimeToFileDate (see page 4152)	Converts a TDateTime (see page 3335) value to an OS timestamp value.
	DateTimeToStr (see page 4153)	Converts a TDateTime (see page 3335) value to a string.
	DateTimeToStr (see page 4153)	Converts a TDateTime (see page 3335) value to a string.
	DateTimeToString (see page 4153)	Converts a TDateTime (see page 3335) value to a string using a specified Format (see page 4232).
	DateTimeToString (see page 4153)	Converts a TDateTime (see page 3335) value to a string using a specified Format (see page 4232).

	DateTimeToSystemTime (see page 4155)	Converts a TDateTime (see page 3335) value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 4156)	Converts a TDateTime (see page 3335) value into the corresponding TTimeStamp (see page 4375) value.
	DateToStr (see page 4156)	Converts a TDateTime (see page 3335) value to a string.
	DateToStr (see page 4156)	Converts a TDateTime (see page 3335) value to a string.
	DayOfWeek (see page 4158)	Returns the day of the week for a specified date.
	DecodeDate (see page 4159)	Returns Year, Month, and Day values for a TDateTime (see page 3335) value.
	DecodeDateFully (see page 4160)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3335) value.
	DecodeTime (see page 4161)	Breaks a TDateTime (see page 3335) value into hours, minutes, seconds, and milliseconds.
	DelegatesEqual (see page 4162)	This is function SysUtils.DelegatesEqual.
	DeleteFile (see page 4162)	Deletes a file from disk.
	DirectoryExists (see page 4164)	Determines whether a specified directory exists.
	DiskFree (see page 4165)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 4166)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 4167)	Disposes of a string pointer that was allocated using NewStr (see page 4271).
	ElementToCharIndex (see page 4168)	This is function SysUtils.ElementToCharIndex.
	ElementToCharIndex (see page 4168)	This is function SysUtils.ElementToCharIndex.
	ElementToCharLen (see page 4168)	This is function SysUtils.ElementToCharLen.
	ElementToCharLen (see page 4169)	This is function SysUtils.ElementToCharLen.
	EncodeDate (see page 4169)	Returns a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.
	EncodeTime (see page 4170)	Returns a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 4171)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 4172)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 4173)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 4173)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 4174)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 4175)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 4176)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 4176)	Returns the drive portion of a file name.
	ExtractFileExt (see page 4177)	Returns the extension portions of a file name.
	ExtractFileName (see page 4177)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 4179)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 4181)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 4182)	Converts a file name to the short 8.3 form.
	FileAge (see page 4183)	Returns the OS timestamp of a file.
	FileAge (see page 4183)	Returns the OS timestamp of a file.
	FileClose (see page 4185)	Closes a specified file.
	FileCreate (see page 4188)	Creates a new file.
	FileCreate (see page 4188)	Creates a new file.
	FileDateToDateTime (see page 4191)	Converts an OS timestamp value to TDateTime (see page 3335) value.
	FileExists (see page 4193)	Tests if a specified file exists.
	FileGetAttr (see page 4197)	Returns the file attributes of FileName.
	FileGetDate (see page 4199)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 4200)	Reports whether a file is read-only.
	FileOpen (see page 4200)	Opens a specified file using a specified access mode.
	FileRead (see page 4204)	Reads a specified number of bytes from a file.
	FileSearch (see page 4206)	Searches a specified directory path for a file.
	FileSeek (see page 4207)	Repositions read/write point.
	FileSeek (see page 4207)	Repositions read/write point.
	FileSetAttr (see page 4209)	Sets the file attributes of a specified file.
	FileSetDate (see page 4212)	Sets the OS time stamp for a specified file.

	FileSetDate (see page 4212)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 4212)	Sets the file permissions to read-only.
	FileWrite (see page 4213)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 4216)	Finalizes the specified package (.bpl file).
	FindClose (see page 4216)	Releases memory allocated by FindFirst (see page 4218).
	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4218)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 4218)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 4221)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4218).
	FloatToCurr (see page 4223)	Converts a floating-point value to a Currency (see page 3242) value.
	FloatToDateTime (see page 4223)	Converts a floating-point value to a TDateTime (see page 3335) value.
	FloatToDecimal (see page 4224)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 4224)	Converts a floating point value to a string.
	FloatToStr (see page 4224)	Converts a floating point value to a string.
	FloatToStrF (see page 4226)	Converts a floating point value to a string, using a specified Format (see page 4232), Precision, and Digits.
	FloatToStrF (see page 4226)	Converts a floating point value to a string, using a specified Format (see page 4232), Precision, and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToText (see page 4227)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4228)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 4229)	Returns formatted output using a resourced format string.
	FmtStr (see page 4229)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 4229)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 4230)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 4232)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 4232)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4237)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 4238)	Formats a Currency (see page 3242) object.
	FormatCurr (see page 4238)	Formats a Currency (see page 3242) object.
	FormatDateTime (see page 4239)	Formats a TDateTime (see page 3335) value.
	FormatDateTime (see page 4239)	Formats a TDateTime (see page 3335) value.
	FormatFloat (see page 4241)	Formats a floating point value.

	FormatFloat (see page 4241)	Formats a floating point value.
	FreeAndNil (see page 4245)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 4246)	Get Delphi Access Licence.
	GetCurrentDir (see page 4246)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 4248)	Retrieves an environment variable value.
	GetFileVersion (see page 4249)	Returns file version.
	GetFormatSettings (see page 4250)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 4250)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 4250)	Populates a TFormatSettings (see page 4367) data structure.
	GetLocaleStr (see page 4251)	Retrieves a localisation value.
	GetModuleName (see page 4251)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 4252)	Returns the description resource from a package.
	GetPackageInfo (see page 4253)	Enumerates all the units and required packages for a package.
	GetTime (see page 4253)	Returns the current time.
	GUIDToString (see page 4254)	Converts a class identifier GUID to a string.
	HashName (see page 4256)	This is function SysUtils.HashName.
	IncAMonth (see page 4256)	Increments date data by one month.
	IncludeTrailingBackslash (see page 4257)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 4257)	Ensures path name ends with delimiter.
	IncMonth (see page 4258)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 4259)	Validates and initializes a given package.
	InitializePackage (see page 4259)	Validates and initializes a given package.
	IntToHex (see page 4259)	Returns the hex representation of an integer.
	IntToHex (see page 4259)	Returns the hex representation of an integer.
	IntToStr (see page 4260)	Converts an integer to a string.
	IntToStr (see page 4260)	Converts an integer to a string.
	IsAssembly (see page 4261)	Determines whether a dll is an Assembly.
	IsDelimiter (see page 4262)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 4262)	Indicates whether two TGUID (see page 3800) values are the same.
	IsLeadChar (see page 4264)	This is function SysUtils.IsLeadChar.
	IsLeadChar (see page 4264)	This is function SysUtils.IsLeadChar.
	IsLeapYear (see page 4264)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 4266)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 4266)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 4267)	Lists the locales for which support is available.
	LastDelimiter (see page 4267)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 4268)	Loads a specified package.
	LoadPackage (see page 4268)	Loads a specified package.
	LoadStr (see page 4268)	Loads a string resource from the application's executable file.
	LowerCase (see page 4269)	Converts an ASCII string to lowercase.
	LowerCase (see page 4269)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 4270)	Converts a specified number of milliseconds into a TTimeStamp (see page 4375) value.
	NewStr (see page 4271)	Allocates a string on the heap.
	NextCharIndex (see page 4271)	Returns the byte index of the next character.
	NextCharIndex (see page 4271)	Returns the byte index of the next character.
	Now (see page 4271)	Returns the current date and time.
	OutOfMemoryError (see page 4272)	Raises an EOutOfMemory (see page 4008) exception.
	PlatformBytesOf (see page 4273)	This is function SysUtils.PlatformBytesOf.
	PlatformStringOf (see page 4273)	This is function SysUtils.PlatformStringOf.
	QuotedStr (see page 4273)	Returns the quoted version of a string.
	RaiseLastOSError (see page 4274)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 4274)	Raises an exception for the last occurring OS or system library error.

	RaiseLastWin32Error (see page 4275)	Raises an exception for the last occurring Win32 error.
	RCS (see page 4276)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 4276)	Deletes an existing empty directory.
	RenameFile (see page 4277)	Changes a file name.
	ReplaceDate (see page 4279)	Replaces the date portion of a TDateTime (see page 3335) value with a specified date.
	ReplaceTime (see page 4280)	Replaces the time portion of a TDateTime (see page 3335) value with a specified time.
	RPR (see page 4281)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 4281)	Safe and simple library loading.
	SameFileName (see page 4282)	Compares file names based on the current locale.
	SameStr (see page 4282)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 4282)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 4283)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 4283)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 4283)	Sets the current directory.
	ShowException (see page 4284)	Displays an exception message with its physical address
	Sleep (see page 4284)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 4285)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 4285)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4285).
	StrBufSize (see page 4285)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4285).
	StrByteType (see page 4286)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrByteType (see page 4286)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 4286)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCat (see page 4286)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 4288)	Returns the length, in bytes, of a character.
	StrCharLength (see page 4288)	Returns the length, in bytes, of a character.
	StrComp (see page 4288)	Compares two strings with case sensitivity.
	StrComp (see page 4288)	Compares two strings with case sensitivity.
	StrCopy (see page 4289)	Copies a null-terminated string.
	StrCopy (see page 4289)	Copies a null-terminated string.
	StrDispose (see page 4291)	Disposes of a string.
	StrDispose (see page 4291)	Disposes of a string.
	StrECopy (see page 4292)	Copies null-terminated string.
	StrECopy (see page 4292)	Copies null-terminated string.
	StrEnd (see page 4293)	Returns a pointer to the end of a null terminated string.
	StrEnd (see page 4293)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrFmt (see page 4294)	Formats entries in an array.
	StrIComp (see page 4295)	Compares two strings without case sensitivity.
	StrIComp (see page 4295)	Compares two strings without case sensitivity.
	StringOf (see page 4296)	This is function SysUtils.StringOf.
	StringReplace (see page 4297)	Returns a string with occurrences of one substring replaced by another substring.
	StringToGUID (see page 4297)	Converts a string to a GUID.
	StrLCat (see page 4299)	Appends up to a specified maximum number of characters to string.
	StrLCat (see page 4299)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 4300)	Compares up to a specified maximum number of characters in two strings.
	StrLComp (see page 4300)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 4302)	Copies up to a specified maximum number of characters from Source to Dest.

	StrLCopy (see page 4302)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 4303)	Returns number of characters in a string excluding the null terminator.
	StrLen (see page 4303)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4304)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 4305)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLIComp (see page 4305)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 4307)	Converts a string to lowercase.
	StrLower (see page 4307)	Converts a string to lowercase.
	StrMove (see page 4308)	Copies specified number of characters to string.
	StrMove (see page 4308)	Copies specified number of characters to string.
	StrNew (see page 4310)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNew (see page 4310)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 4311)	Returns pointer to following character.
	StrNextChar (see page 4311)	Returns pointer to following character.
	StrPas (see page 4311)	Converts null-terminated string to an AnsiString (long string).
	StrPas (see page 4311)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 4313)	Copies an AnsiString (long string) to a null-terminated string.
	StrPCopy (see page 4313)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 4314)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPLCopy (see page 4314)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 4314)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrPos (see page 4314)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 4315)	Returns a pointer to the last occurrence of a specified character in a string.
	StrRScan (see page 4315)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 4317)	Returns a pointer to first occurrence of a specified character in a string.
	StrScan (see page 4317)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 4318)	Converts a string to a Boolean value.
	StrToBoolDef (see page 4318)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 4319)	Converts a string to a Currency (see page 3242) value.
	StrToCurr (see page 4319)	Converts a string to a Currency (see page 3242) value.
	StrToCurrDef (see page 4320)	Converts a string to a Currency (see page 3242) value, with error default.
	StrToCurrDef (see page 4320)	Converts a string to a Currency (see page 3242) value, with error default.
	StrToDate (see page 4320)	Converts a string to a TDateTime (see page 3335) value.
	StrToDate (see page 4320)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateDef (see page 4322)	Converts a string to a TDateTime (see page 3335) value, with error default.
	StrToDateDef (see page 4322)	Converts a string to a TDateTime (see page 3335) value, with error default.
	StrToDateTime (see page 4322)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateTime (see page 4322)	Converts a string to a TDateTime (see page 3335) value.
	StrToDateTimeDef (see page 4323)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToDateTimeDef (see page 4323)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToFloat (see page 4324)	Converts a given string to a floating-point value.
	StrToFloat (see page 4324)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 4327)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 4327)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 4328)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 4329)	Converts a string that represents an integer (decimal or hex notation) to a number.

	StrToInt64Def (see page 4329)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 4330)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 4332)	Converts a string to a TDateTime (see page 3335) value.
	StrToTime (see page 4332)	Converts a string to a TDateTime (see page 3335) value.
	StrToTimeDef (see page 4333)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrToTimeDef (see page 4333)	Converts a string to a TDateTime (see page 3335) value with error default.
	StrUpper (see page 4334)	Returns a string in upper case.
	StrUpper (see page 4334)	Returns a string in upper case.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4335)	Indicates whether a given object or interface supports a specified interface.
	SysErrorMessage (see page 4336)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 4336)	Converts a system time value into a TDateTime (see page 3335) value.
	TextPos (see page 4337)	This is function SysUtils.TextPos.
	TextPos (see page 4337)	This is function SysUtils.TextPos.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4338)	Converts a null-terminated string to a floating-point value.
	Time (see page 4338)	Returns the current time.
	TimeStampToDateTime (see page 4339)	Converts a TTimeStamp (see page 4375) value into the corresponding TDateTime (see page 3335) value.
	TimeStampToMSecs (see page 4340)	Converts a TTimeStamp (see page 4375) value into an absolute number of milliseconds.
	TimeToStr (see page 4340)	Returns a string that represents a TDateTime (see page 3335) value.
	TimeToStr (see page 4340)	Returns a string that represents a TDateTime (see page 3335) value.
	Trim (see page 4342)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 4343)	Trims leading spaces and control characters from a string.
	TrimRight (see page 4343)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 4343)	Creates a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 4344)	Creates a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 4345)	Converts a floating-point value to a Currency (see page 3242) value.
	TryFloatToDateTime (see page 4345)	Converts a floating-point value to a TDateTime (see page 3335) value.
	TryStrToBool (see page 4346)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 4347)	Converts a string to a Currency (see page 3242) value, with a Boolean success code.
	TryStrToCurr (see page 4347)	Converts a string to a Currency (see page 3242) value, with a Boolean success code.
	TryStrToDate (see page 4347)	Converts a string to a TDateTime (see page 3335) value, with a Boolean success code.
	TryStrToDate (see page 4347)	Converts a string to a TDateTime (see page 3335) value, with a Boolean success code.
	TryStrToDateTime (see page 4348)	Converts a string to a TDateTime (see page 3335) value with a Boolean success code.
	TryStrToDateTime (see page 4348)	Converts a string to a TDateTime (see page 3335) value with a Boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4349)	Converts a given string to a floating-point value with a boolean success code.

	TryStrToInt (see page 4349)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 4350)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 4351)	Converts a string to a TDateTime (see page 3335) value with an error default.
	TryStrToTime (see page 4351)	Converts a string to a TDateTime (see page 3335) value with an error default.
	UIntToStr (see page 4351)	This is function SysUtils.UIntToStr.
	UIntToStr (see page 4352)	This is function SysUtils.UIntToStr.
	UnloadPackage (see page 4352)	Unloads a package.
	UpperCase (see page 4352)	Returns a copy of a string in uppercase.
	UpperCase (see page 4352)	Returns a copy of a string in uppercase.
	WideBytesOf (see page 4354)	This is function SysUtils.WideBytesOf.
	WideCompareStr (see page 4354)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 4355)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 4355)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 4355)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 4356)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 4356)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 4356)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 4356)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 4357)	Returns Unicode string converted to lower case.
	WideSameStr (see page 4357)	Compares Unicode strings based on the current locale with case sensitivity.
	WideSameText (see page 4358)	Compares Unicode strings based on the current locale without case sensitivity.
	WideStrAlloc (see page 4358)	This is function SysUtils.WideStrAlloc.
	WideStringOf (see page 4359)	This is function SysUtils.WideStringOf.
	WideUpperCase (see page 4359)	Returns Unicode string converted to upper case.
	Win32Check (see page 4359)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 4360)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 4360)	Splits a string into multiple lines as its length approaches a specified size.

39.3.1 SysUtils.Abort Function

Raises a silent exception.

Pascal

```
procedure Abort;
```

C++

```
Abort();
```

File

SysUtils

Description

Use Abort to escape from an execution path without reporting an error.

Abort raises a special "silent exception" (EAbort (see page 3982)), which operates like any other exception, but does not display an error message to the end user. Abort redirects execution to the end of the last exception block.

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

EAbort (see page 3982)

39.3.2 SysUtils.AddExitProc Function

Adds a procedure to the exit list.

Pascal

```
procedure AddExitProc(Proc: TProcedure);
```

C++

```
AddExitProc(TProcedure Proc);
```

File

SysUtils

Description

AddExitProc adds Proc to the Delphi exit procedure list, so that Proc is automatically executed when the application terminates. AddExitProc is not compatible with CLX package support and is provided for backward compatibility only. Do not use AddExitProc in new applications.

39.3.3 SysUtils.AddTerminateProc Function

Adds a terminate procedure to the system list of termination procedures.

Pascal

```
procedure AddTerminateProc(TermProc: TTerminateProc);
```

C++

```
AddTerminateProc(TTerminateProc TermProc);
```

File

SysUtils

Description

Call AddTerminateProc in Delphi code to add a procedure that is called when the application is about to terminate. The run time library calls all termination procedures before the application terminates. The supplied termination procedure (TermProc) should return true if the application can safely terminate or false if the application cannot safely terminate. If any termination procedure returns false, the application does not terminate.

See Also

CallTerminateProcs (see page 4136)

39.3.4 SysUtils.AdjustLineBreaks Function

Adjusts line break characters to reflect Windows or Linux style.

Pascal

```
function AdjustLineBreaks(const S: string; Style: TTextLineBreakStyle = tlbsCRLF): string;
overload;
```

C++

```
UnicodeString AdjustLineBreaks(const UnicodeString S, TTextLineBreakStyle Style = tlbsCRLF);
```

File

SysUtils

Description

AdjustLineBreaks returns a text string with its line breaks adjusted to fit a specified style.

S is the string to convert.

Style indicates whether line breaks should all be Windows-style line breaks with a carriage return and linefeed (tlbsCRLF), or Linux-style breaks with a single linefeed character (tlbsLF). If Style is omitted, the style for the compile target is used.

See Also

TrimLeft (see page 4343)

TrimRight (see page 4343)

39.3.5 SysUtils.AnsiCompareFileName Function

Compares file names based on the current locale.

Pascal

```
function AnsiCompareFileName(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareFileName(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareFileName compares two file names. The current locale controls the compare operation. Comparison is case-insensitive under Windows, case-sensitive under Linux.

The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

AnsiCompareFileName supports the idiosyncrasies of MS-DOS file names that use Zenkaku Moji, a character encoding used in some Asian locales. Under Windows locales that do not use MBCS, AnsiCompareFileName is identical to AnsiCompareText (see page 4115). Under Linux, AnsiCompareFileName is identical to AnsiCompareStr (see page 4114).

See Also

SameFileName (see page 4282)

AnsiCompareText (see page 4115)

39.3.6 SysUtils.AnsiCompareStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiCompareStr(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareStr(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. Thus, setting S1 to 'a' and S2 to 'A' causees AnsiCompareStr to return a value less than zero, while CompareStr (see page 4142), with the same arguments, returns a value greater than zero.

See Also

AnsiCompareText (see page 4115)

AnsiStrComp (see page 4120)

AnsiSameStr (see page 4119)

39.3.7 SysUtils.AnsiCompareText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiCompareText(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareText(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. AnsiCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

AnsiCompareStr (see page 4114)

AnsiStrIComp (see page 4121)

AnsiSameText (see page 4119)

39.3.8 SysUtils.AnsiDequotedStr Function

Converts a quoted string into an unquoted string

Pascal

```
function AnsiDequotedStr(const S: string; AQuote: Char): string; overload;
```

C++

```
UnicodeString AnsiDequotedStr(const UnicodeString S, Char AQuote);
```

File

SysUtils

Description

AnsiDequotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The AQuote parameter defines what character to use as a quote character. If the S parameter does not begin and end with the quote character, AnsiDequotedStr returns S unchanged.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 4118)

AnsiExtractQuotedStr (see page 4116)

39.3.9 SysUtils.AnsiExtractQuotedStr Function

Converts a quoted string into an unquoted string.

Pascal

```
function AnsiExtractQuotedStr(var Src: PAnsiChar; Quote: AnsiChar): AnsiString; overload;
function AnsiExtractQuotedStr(var Src: PWidechar; Quote: WideChar): UnicodeString; overload;
```

C++

```
AnsiString AnsiExtractQuotedStr(PAnsiChar Src, AnsiChar Quote);
UnicodeString AnsiExtractQuotedStr(PWidechar Src, WideChar Quote);
```

File

SysUtils

Description

AnsiExtractQuotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The Quote parameter defines what character to use as a quote character. If the first character in Src is not the value of the Quote parameter, AnsiExtractQuotedStr returns an empty string.

The function copies characters from Src to the result string until the second solitary quote character or the first null character in Src. The Src parameter is updated to point to the first character following the quoted string. If Src does not contain a matching end quote character, the Src parameter is updated to point to the terminating null character.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 4118)

39.3.10 SysUtils.AnsiLastChar Function

Returns a pointer to the last character in a string.

Pascal

```
function AnsiLastChar(const S: AnsiString): PAnsiChar; overload;
function AnsiLastChar(const S: UnicodeString): PWideChar; overload;
```

C++

```
PAnsiChar AnsiLastChar(const AnsiString S);
PWideChar AnsiLastChar(const UnicodeString S);
```

File

SysUtils

Description

Call AnsiLastChar to obtain the last character in a string. This function supports multi-byte character sets (MBCS).

See Also

AnsiStrLastChar (see page 4121)

AnsiStrRScan (see page 4124)

39.3.11 SysUtils.AnsiLowerCase Function

Returns a string that is a copy of the given string converted to lower case.

Pascal

```
function AnsiLowerCase(const S: string): string; overload;
```

C++

```
UnicodeString AnsiLowerCase(const UnicodeString S);
```

File

SysUtils

Description

AnsiLowerCase returns a string that is a copy of the given string converted to lower case. The conversion uses the current locale. This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCaseFileName (see page 4117)

AnsiUpperCase (see page 4126)

LowerCase (see page 4269)

39.3.12 SysUtils.AnsiLowerCaseFileName Function

Converts a file name to lower case.

Pascal

```
function AnsiLowerCaseFileName(const S: string): string; overload;
```

C++

```
UnicodeString AnsiLowerCaseFileName(const UnicodeString S);
```

File

SysUtils

Description

AnsiLowerCaseFileName returns a string that is a copy of S converted to lower case. The conversion uses the current locale.

Note: On Windows, AnsiLowerCaseFileName supports lowercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux, or in locales that do not use multi byte character systems (MBCS), AnsiLowerCaseFileName is identical to AnsiLowerCase (see page 4117).

See Also

AnsiLowerCase (see page 4117)

AnsiUpperCaseFileName (see page 4126)

39.3.13 SysUtils.AnsiPos Function

Locates the position of a substring.

Pascal

```
function AnsiPos(const Substr: string; const S: string): Integer; overload;
```

C++

```
int AnsiPos(const UnicodeString Substr, const UnicodeString S);
```

File

SysUtils

Description

Call AnsiPos to obtain the byte offset of the Substr parameter, as it appears in the string S. For example, if Substr is the string "AB", and S is the string "ABCDE", AnsiPos returns 1. If Substr does not appear in S, AnsiPos returns 0.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiStrPos (see page 4124)

39.3.14 SysUtils.AnsiQuotedStr Function

Returns the quoted version of a string.

Pascal

```
function AnsiQuotedStr(const S: string; Quote: Char): string; overload;
```

C++

```
UnicodeString AnsiQuotedStr(const UnicodeString S, Char Quote);
```

File

SysUtils

Description

Use AnsiQuotedStr to convert a string (S) to a quoted string, using the provided Quote character. A Quote character is inserted at the beginning and end of S, and each Quote character in the string is doubled.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiExtractQuotedStr (see page 4116)

QuotedStr (see page 4273)

39.3.15 SysUtils.AnsiSameStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiSameStr(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameStr(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameText (see page 4119)

AnsiCompareStr (see page 4114)

AnsiSameCaption

39.3.16 SysUtils.AnsiSameText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiSameText(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameText(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameStr (see page 4119)

AnsiCompareText (see page 4115)

AnsiSameCaption

39.3.17 SysUtils.AnsiStrAlloc Function

Pascal

```
function AnsiStrAlloc(Size: Cardinal): PAnsiChar;
```

C++

```
PAnsiChar AnsiStrAlloc(unsigned Size);
```

File

SysUtils

Description

This is function SysUtils.AnsiStrAlloc.

39.3.18 SysUtils.AnsiStrComp Function

Compares null-terminated character strings.

Pascal

```
function AnsiStrComp(S1: PAnsiChar; S2: PAnsiChar): Integer;
function AnsiStrComp(S1: PWideChar; S2: PWideChar): Integer;
```

C++

```
int AnsiStrComp(PAnsiChar S1, PAnsiChar S2);
int AnsiStrComp(PWideChar S1, PWideChar S2);
```

File

SysUtils

Description

AnsiStrComp compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareStr (see page 4114)

AnsiStrIComp (see page 4121)

AnsiStrLComp (see page 4122)

39.3.19 SysUtils.AnsiStrIComp Function

Compares null terminated character strings case insensitively.

Pascal

```
function AnsiStrIComp(S1: PAnsiChar; S2: PAnsiChar): Integer;
function AnsiStrIComp(S1: PWideChar; S2: PWideChar): Integer;
```

C++

```
int AnsiStrIComp(PAnsiChar S1, PAnsiChar S2);
int AnsiStrIComp(PWideChar S1, PWideChar S2);
```

File

SysUtils

Description

AnsiStrIComp compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is the same as for CompareStr (see page 4142).

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareText (see page 4115)

AnsiStrComp (see page 4120)

39.3.20 SysUtils.AnsiStrLastChar Function

Returns a pointer to the last character in a null-terminated string.

Pascal

```
function AnsiStrLastChar(P: PAnsiChar): PAnsiChar; overload;
function AnsiStrLastChar(P: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrLastChar(PAnsiChar P);
PWideChar AnsiStrLastChar(PWideChar P);
```

File

SysUtils

Description

Call AnsiStrLastChar to obtain a pointer to the last full character in a null-terminated sequence. When working with multi-byte character sets (MBCS), the last character will not be the same as the last char value if the last two bytes represent a single double byte character.

See Also

AnsiLastChar (see page 4116)

39.3.21 SysUtils.AnsiStrLComp Function

Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.

Pascal

```
function AnsiStrLComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: Cardinal): Integer; overload;
function AnsiStrLComp(S1: PWideChar; S2: PWideChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLComp(PAnsiChar S1, PAnsiChar S2, unsigned MaxLen);
int AnsiStrLComp(PWideChar S1, PWideChar S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLComp compares S1 to S2, with case sensitivity. If S1 or S2 is longer than MaxLen bytes, AnsiStrLComp only compares the first MaxLen bytes. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

Note: Most locales consider lowercase characters to come before corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters come after uppercase characters.

See Also

AnsiStrComp (see page 4120)

AnsiStrLIComp (see page 4122)

StrLComp (see page 4300)

39.3.22 SysUtils.AnsiStrLIComp Function

Compares two strings, case-insensitively, up to the first MaxLen bytes.

Pascal

```
function AnsiStrLIComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: Cardinal): Integer; overload;
function AnsiStrLIComp(S1: PWideChar; S2: PWideChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLIComp(PAnsiChar S1, PAnsiChar S2, unsigned MaxLen);
int AnsiStrLIComp(PWideChar S1, PWideChar S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLIComp compares S1 to S2, without case sensitivity. If S1 or S2 is longer than MaxLen characters, AnsiStrLIComp only compares up to the first MaxLen characters. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLIComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

See Also

AnsiStrIComp (see page 4121)

AnsiStrLComp (see page 4122)

StrLIComp (see page 4305)

39.3.23 SysUtils.AnsiStrLower Function

Converts all characters in a null-terminated string to lower case.

Pascal

```
function AnsiStrLower(Str: PAnsiChar): PAnsiChar; overload;
function AnsiStrLower(Str: PWideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrLower(PAnsiChar Str);
PWideChar AnsiStrLower(PWideChar Str);
```

File

SysUtils

Description

AnsiStrLower returns a null-terminated character sequence where all characters are lower case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 4117)

AnsiLowerCaseFileName (see page 4117)

AnsiStrUpper (see page 4125)

39.3.24 SysUtils.AnsiStrPos Function

Returns a pointer to the first occurrence of a substring.

Pascal

```
function AnsiStrPos(Str: PAnsiChar; SubStr: PAnsiChar): PAnsiChar; overload;
function AnsiStrPos(Str: PWideChar; SubStr: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrPos(PAnsiChar Str, PAnsiChar SubStr);
PWideChar AnsiStrPos(PWideChar Str, PWideChar SubStr);
```

File

SysUtils

Description

Call AnsiStrPos to locate the first occurrence of SubStr in Str (see page 3764). If SubStr does not occur in Str (see page 3764), AnsiStrPos returns nil (Delphi) or NULL (C++).

Unlike the StrPos (see page 4314) function, AnsiStrPos works with multi-byte characters sets (MBCS).

See Also

AnsiPos (see page 4118)

StrPos (see page 4314)

39.3.25 SysUtils.AnsiStrRScan Function

Returns a pointer to the last occurrence of a specified character in a specified string.

Pascal

```
function AnsiStrRScan(Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function AnsiStrRScan(Str: PWideChar; Chr: WideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrRScan(PAnsiChar Str, AnsiChar Chr);
PWideChar AnsiStrRScan(PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

Call AnsiStrRScan to obtain a pointer to the last occurrence of Chr (see page 3640) in Str (see page 3764). If Chr (see page 3640) does not occur in Str (see page 3764), AnsiStrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrRScan (see page 4315) function, AnsiStrRScan works with multi-byte character systems (MBCS).

See Also

AnsiStrScan (see page 4125)

StrRScan (see page 4315)

39.3.26 SysUtils.AnsiStrScan Function

Returns a pointer to first occurrence of a character in a string.

Pascal

```
function AnsiStrScan(Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function AnsiStrScan(Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrScan(PAnsiChar Str, AnsiChar Chr);
PWideChar AnsiStrScan(PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

Call AnsiStrScan to obtain a pointer to the first occurrence of Chr (see page 3640) in Str (see page 3764). If Chr (see page 3640) does not occur in Str (see page 3764), AnsiStrScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrScan (see page 4317) function, AnsiStrScan works with multi-byte character systems (MBCS).

See Also

AnsiStrRScan (see page 4124)

StrScan (see page 4317)

39.3.27 SysUtils.AnsiStrUpper Function

Converts all characters in a null-terminated string to upper case.

Pascal

```
function AnsiStrUpper(Str: PAnsiChar): PAnsiChar; overload;
function AnsiStrUpper(Str: PWideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrUpper(PAnsiChar Str);
PWideChar AnsiStrUpper(PWideChar Str);
```

File

SysUtils

Description

AnsiStrUpper returns a null-terminated string where all characters are upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiUpperCase (see page 4126)

AnsiUpperCaseFileName (see page 4126)

AnsiStrLower (see page 4123)

39.3.28 SysUtils.AnsiUpperCase Function

Converts a string to upper case.

Pascal

```
function AnsiUpperCase(const S: string): string; overload;
```

C++

```
UnicodeString AnsiUpperCase(const UnicodeString S);
```

File

SysUtils

Description

AnsiUpperCase returns a string that is a copy of S, converted to upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 4117)

UpperCase (see page 4352)

39.3.29 SysUtils.AnsiUpperCaseFileName Function

Converts a file name to upper case.

Pascal

```
function AnsiUpperCaseFileName(const S: string): string; overload;
```

C++

```
UnicodeString AnsiUpperCaseFileName(const UnicodeString S);
```

File

SysUtils

Description

AnsiUpperCaseFileName returns a string that is a copy of S converted to upper case. The conversion uses the current locale.

Note: On Windows, AnsiUpperCaseFileName supports uppercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux or in locales that do not use multi byte character systems (MBCS), AnsiUpperCaseFileName is identical to AnsiUpperCase (see page 4126).

See Also

AnsiLowerCaseFileName (see page 4117)

AnsiUpperCase (see page 4126)

39.3.30 SysUtils.AppendStr Function

Appends a dynamically allocated string to an existing string.

Pascal

```
procedure AppendStr(var Dest: AnsiString; const S: AnsiString); deprecated;
```

C++

```
AppendStr(AnsiString Dest, const AnsiString S);
```

File

SysUtils

Description

AppendStr appends S to the end of Dest. It is provided for backwards compatibility only. Use the + operator instead:

Dest := Dest + S;

Dest = Dest + S;

39.3.31 SysUtils.AssignStr Function

Assigns a new dynamically allocated string to a specified pointer.

Pascal

```
procedure AssignStr(var P: PAnsiString; const S: AnsiString); deprecated;
```

C++

```
AssignStr(PAnsiString P, const AnsiString S);
```

File

SysUtils

Description

Do not use AssignStr in new applications. It is provided for backward compatibility only.

See Also

DisposeStr (see page 4167)

39.3.32 SysUtils.Beep Function

Emits a message beep.

Pascal

```
procedure Beep;
```

C++

```
Beep();
```

File

SysUtils

Description

Beep generates a conventional message beep.

Note: On Windows, Beep calls the Windows API MessageBeep.

Note: On Linux, to minimize library dependencies in the run time library, the SysUtils (see page 3967) version of Beep has limited Linux functionality. For cross-platform GUI applications, use the QControls version of Beep instead.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection. Notice that the
"Match whole word only" and "Match case" work.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes = TSearchTypes();
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
```

```
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following example beeps once for each "beep" passed in
  on the command line.  The example terminates the application
  if  "exit" is passed in on the command line.  Build the
  project to generate a .exe file and then execute that file
  from a command line: "ParamCount_proj beep beep exit".
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
  //  Use print statements since you can't use the debugger.
  //  MessageDlg(
  //    'System ParamCount = ' + IntToStr(System.ParamCount),
  //    mtConfirmation, [mbOK], 0);
    for i := 1 to System.ParamCount do
    begin
      if LowerCase(ParamStr(i)) = 'beep' then
        Beep
      else if LowerCase(ParamStr(i)) = 'exit' then
        Application.Terminate;
      Sleep(250);
    end;
  end;
```

39.3.33 SysUtils.BoolToStr Function

Converts a Boolean value to a string.

Pascal

```
function BoolToStr(B: Boolean; UseBoolStrs: Boolean = False): string;
```

C++

```
UnicodeString BoolToStr(Boolean B, Boolean UseBoolStrs = False);
```

File

SysUtils

Description

BoolToStr converts the given Boolean value to a string as follows:

Value of B	Value of UseBoolStrs	Value of returned string
true	false	'-1'
true	true	The first string in TrueBoolStrs (see page 4403) (default 'True');
false	false	'0'
false	true	The first string in FalseBoolStrs (see page 4392) (default 'False');

Note: The TrueBoolStrs (see page 4403) and FalseBoolStrs (see page 4392) variables are undefined when an application starts. Any use of StrToBool (see page 4318), BoolToStr or TryStrToBool (see page 4346) will cause these variables to be initialised with the default values above.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.t_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.t_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
      Memo1->Lines->Add(buffer);
      buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
      Memo1->Lines->Add(buffer);
      Memo1->Lines->Add("");
    }
  }
  __finally
  {
    Memo1->Lines->EndUpdate();
  }
}
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
```

```
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

StrToBool (see page 4318)

TryStrToBool (see page 4346)

TrueBoolStrs (see page 4403)

FalseBoolStrs (see page 4392)

39.3.34 SysUtils.ByteLength Function

Pascal

```
function ByteLength(const S: string): Integer;
```

C++

```
int ByteLength(const UnicodeString S);
```

File

SysUtils

Description

This is function SysUtils.ByteLength.

39.3.35 SysUtils.BytesOf Function

Pascal

```
function BytesOf(const Val: AnsiChar): TBytes; overload;
```

C++

```
TBytes BytesOf(const AnsiChar Val);
```

File

SysUtils

Description

This is function SysUtils.BytesOf.

39.3.36 SysUtils.BytesOf Function

Pascal

```
function BytesOf(const Val: RawByteString): TBytes; overload;
```

C++

```
TBytes BytesOf(const RawByteString Val);
```

File

SysUtils

Description

This is function SysUtils.BytesOf.

39.3.37 SysUtils.BytesOf Function

Pascal

```
function BytesOf(const Val: UnicodeString): TBytes; overload;
```

C++

```
TBytes BytesOf(const UnicodeString Val);
```

File

SysUtils

Description

This is function SysUtils.BytesOf.

39.3.38 SysUtils.BytesOf Function

Pascal

```
function BytesOf(const Val: WideChar): TBytes; overload;
```

C++

```
TBytes BytesOf(const WideChar Val);
```

File

SysUtils

Description

This is function SysUtils.BytesOf.

39.3.39 SysUtils.ByteToCharIndex Function

Returns the position of the character that contains a specified byte in a string.

Pascal

```
function ByteToCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
function ByteToCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int ByteToCharIndex(const AnsiString S, int Index);
int ByteToCharIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call ByteToCharIndex to determine which character contains a particular byte in a string, where 1 specifies the first byte, 2 specifies the second byte, and so on. If the Index parameter specifies a byte that is not in the AnsiString (Index < 0 or Index > Length (see page 3702)(S)), ByteToCharIndex returns 0.

If the system does not use a multi-byte character system (MBCS), ByteToCharIndex simply returns the value of Index, because there is a 1:1 correspondence between bytes and characters. Otherwise, ByteToCharIndex returns the position of the character, where 1 specifies the first character, 2 specifies the second character, and so on.

For example, if an AnsiString contains two characters, a single-byte character followed by a double byte character, then ByteToCharIndex returns 1 when Index is 1, it returns 2 when Index is 2 or 3, and it returns 0 for any other value of Index.

See Also

ByteToCharLen (see page 4134)

CharToByteIndex (see page 4139)

39.3.40 SysUtils.ByteToCharLen Function

Returns the number of characters fully contained in the first MaxLen bytes of a string.

Pascal

```
function ByteToCharLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
function ByteToCharLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int ByteToCharLen(const AnsiString S, int MaxLen);
int ByteToCharLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

Call ByteToCharLen to determine the number of characters represented by a portion of a string. ByteToCharLen counts characters until it reaches MaxLen bytes or the end of the string, whichever comes first.

When working with multi-byte character systems (MBCS), the value returned by ByteToCharLen may be less than MaxLen or the length of the string.

Note: ByteToCharLen will count a partial character if it appears as the last of the first MaxLen bytes.

See Also

ByteToCharIndex (see page 4134)

CharToByteIndex (see page 4139)

CharToByteLen (see page 4140)

39.3.41 SysUtils.ByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function ByteType(const S: AnsiString; Index: Integer): TMbcsByteType; overload;
function ByteType(const S: UnicodeString; Index: Integer): TMbcsByteType; overload;
```

C++

```
TMbcsByteType ByteType(const AnsiString S, int Index);
TMbcsByteType ByteType(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call ByteType to determine whether a specified byte in a string is a single-byte character, the first byte of a multibyte character, or one of the trailing bytes.

AnsiString is the string that contains the byte in question.

Index identifies the byte for which you want to know the byte type. Bytes are numbered from 1.

If the system is not using a multi-byte character system (MBCS), ByteType always returns mbSingleByte. Otherwise, ByteType returns mbSingleByte if the indicated byte represents a complete character in S, mbLeadByte if it represents the first byte of a double byte character, and mbTrailByte if it represents a subsequent byte of a multibyte character.

Note: No checking is done to ensure that Index is less than the length of S. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

LeadBytes (see page 4394)

StrByteType (see page 4286)

39.3.42 SysUtils.CallTerminateProcs Function

Calls all of the functions in the termination procedure list.

Pascal

```
function CallTerminateProcs: Boolean;
```

C++

```
Boolean CallTerminateProcs();
```

File

SysUtils

Description

CallTerminateProcs is called internally when an application is about to terminate. It returns true only if all of the functions in the application's terminate procedure list return true. Do not call this function directly. Its use is automatic in Delphi application.

See Also

AddTerminateProc (see page 4112)

39.3.43 SysUtils.ChangeFileExt Function

Changes the extension of a file name.

Pascal

```
function ChangeFileExt(const FileName: string; const Extension: string): string; overload;
```

C++

```
UnicodeString ChangeFileExt(const UnicodeString FileName, const UnicodeString Extension);
```

File

SysUtils

Description

ChangeFileExt takes the file name passed in FileName and changes the extension of the file name to the extension passed in Extension. Extension specifies the new extension, including the initial dot character.

ChangeFileExt does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/

#include <memory>       //for STL auto_ptr class
```

```
void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    std::auto_ptr<TIcon> pi(new Graphics::TIcon);
    AnsiString as;
    pi->LoadFromFile(OpenDialog1->FileName);
    as = ChangeFileExt(OpenDialog1->FileName,".BMP");
    Image1->Width = pi->Width;
    Image1->Height = pi->Height;
    Image1->Canvas->Draw(0, 0, pi.get());
    Image1->Picture->SaveToFile(as);
    ShowMessage(OpenDialog1->FileName + " Saved to " + as);
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
```

See Also

ExtractFileExt (see page 4177)

39.3.44 SysUtils.ChangeFilePath Function

Changes the path of a file name.

Pascal

```
function ChangeFilePath(const FileName: string; const Path: string): string; overload;
```

C++

```
UnicodeString ChangeFilePath(const UnicodeString FileName, const UnicodeString Path);
```

File

SysUtils

Description

ChangeFilePath replaces the drive and directory parts of the passed FileName with the passed Path value. If Path is not blank, it must specify a drive (terminated by a colon) and/or a directory (terminated by a backslash).

ChangeFilePath does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

See Also

ExtractFilePath (see page 4179)

39.3.45 SysUtils.CharInSet Function

Pascal

```
function CharInSet(C: AnsiChar; const CharSet: TSysCharSet): Boolean; overload;
```

C++

```
Boolean CharInSet(AnsiChar C, const TSysCharSet CharSet);
```

File

SysUtils

Description

This is function SysUtils.CharInSet.

39.3.46 SysUtils.CharInSet Function

Pascal

```
function CharInSet(C: WideChar; const CharSet: TSysCharSet): Boolean; overload;
```

C++

```
Boolean CharInSet(WideChar C, const TSysCharSet CharSet);
```

File

SysUtils

Description

This is function SysUtils.CharInSet.

39.3.47 SysUtils.CharLength Function

Returns number of bytes used by a character.

Pascal

```
function CharLength(const S: AnsiString; Index: Integer): Integer; overload;
function CharLength(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharLength(const AnsiString S, int Index);
int CharLength(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call CharLength to determine the size in bytes of the character located in S at Index.

If the system is not using a multi-byte character system (MBCS), CharLength always returns 1.

39.3.48 SysUtils.CharToByteIndex Function

Returns the 1-based index of the first byte of a specified character in a string.

Pascal

```
function CharToByteIndex(const S: AnsiString; Index: Integer): Integer; overload;
function CharToByteIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharToByteIndex(const AnsiString S, int Index);
int CharToByteIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call CharToByteIndex to obtain the byte offset for the character at a specified position in the string S, where 1 specifies the first character, 2 specifies the second character, and so on. CharToByteIndex returns zero if Index is out of range (Index <= 0 or S contains fewer than Index characters). If the Index parameter indicates a double-byte character, CharToByteIndex returns the offset of the first byte.

If the system is not using a multi-byte character system (MBCS), CharToByteIndex returns the value of Index, because there is a 1:1 correspondence between bytes and characters.

See Also

ByteToCharIndex (see page 4134)

CharToByteLen (see page 4140)

39.3.49 SysUtils.CharToByteLen Function

Returns the length, in bytes, of the first MaxLen characters of a string.

Pascal

```
function CharToByteLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
function CharToByteLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToByteLen(const AnsiString S, int MaxLen);
int CharToByteLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

Call CharToByteLen to convert the length of a string in characters to the corresponding length in bytes. CharToByteLen returns the number of bytes required by the first MaxLen characters of S. If S contains fewer than MaxLen characters, CharToByteLen returns the byte length of S.

If the system is not using a multi-byte character system (MBCS), CharToByteLen returns the length of S or MaxLen, whichever is smaller.

See Also

ByteToCharLen (see page 4134)

CharToByteIndex (see page 4139)

39.3.50 SysUtils.CharToElementIndex Function

Pascal

```
function CharToElementIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int CharToElementIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.CharToElementIndex.

39.3.51 SysUtils.CharToElementIndex Function

Pascal

```
function CharToElementIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharToElementIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.CharToElementIndex.

39.3.52 SysUtils.CharToElementLen Function

Pascal

```
function CharToElementLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToElementLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.CharToElementLen.

39.3.53 SysUtils.CharToElementLen Function

Pascal

```
function CharToElementLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToElementLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.CharToElementLen.

39.3.54 SysUtils.CheckWin32Version Function

Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3165).

Pascal

```
function CheckWin32Version(AMajor: Integer; AMinor: Integer = 0): Boolean;
```

C++

```
Boolean CheckWin32Version(int AMajor, int AMinor = 0);
```

File

SysUtils

Description

The Windows Operating System (see page 3165) Major.Minor version number is compared with the passed AMajor and AMinor values. AMinor defaults to 0 if not supplied.

CheckWin32Version Returns true if the Windows OS Major.Minor version number is >= the passed AMajor.AMinor value.

Note: Prior to Delphi 7, this routine returned a different result.

See Also

Win32MajorVersion (see page 4405)Win32MinorVersion (see page 4406)

39.3.55 SysUtils.CompareMem Function

Performs a binary comparison of two memory images.

Pascal

```
function CompareMem(P1: Pointer; P2: Pointer; Length: Integer): Boolean;
```

C++

```
Boolean CompareMem(void * P1, void * P2, int Length);
```

File

SysUtils

Description

CompareMem performs a binary compare of Length (see page 3702) bytes of memory referenced by P1 to that of P2. CompareMem returns true if the memory referenced by P1 is identical to that of P2.

See Also

CompareStr (see page 4142)

CompareText (see page 4143)

CollectionsEqual

39.3.56 SysUtils.CompareStr Function

Compares two strings case sensitively.

Pascal

```
function CompareStr(const S1: string; const S2: string): Integer; overload;
function CompareStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareStr(const UnicodeString S1, const UnicodeString S2);
int CompareStr(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareStr compares S1 to S2, with case-sensitivity. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2. The compare operation is based on the 8-bit ordinal value of each character and is not affected by the current locale.

C++ Examples:

```
/*
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString String1 = "STEVE";
  AnsiString String2 = "Steve";
  int I = CompareStr(String1, String2);  // the value of I is < 0
  Edit1->Text = IntToStr(I);
  if (I != 0)
    ShowMessage ("The strings are not equal");
}
```

Delphi Examples:

```
{
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  String1, String2 : string;
  I : Integer;
begin
  String1 := 'STEVE';
  String2 := 'STEVe';
  I := SysUtils.CompareStr(String1, String2);  { the value of I is < 0 }
  Edit1.Text := InttoStr(I);
  if I <> 0 then
    MessageDlg('The strings are not equal', mtWarning, [mbOK], 0)
end;
```

See Also

CompareText (see page 4143)

AnsiCompareStr (see page 4114)

39.3.57 SysUtils.CompareText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function CompareText(const S1: string; const S2: string): Integer; overload;
function CompareText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareText(const UnicodeString S1, const UnicodeString S2);
int CompareText(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareText compares S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. CompareText is not case sensitive and is not affected by the current locale.

C++ Examples:

```
/*
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
*/
int __stdcall CustomSortProc(long Item1, long Item2, long ParamSort)
{
  return -CompareText((reinterpret_cast<TListItem *>(Item1))->Caption,
                      (reinterpret_cast<TListItem *>(Item2))->Caption);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListView1->CustomSort(CustomSortProc, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
```

```
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  dynamic_cast<TCustomListView *>(Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
}
function CustomSortProc(Item1, Item2: TListItem; ParamSort: integer): integer; stdcall;
begin
  Result := -CompareText(Item1.Caption,Item2.Caption);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListView1.CustomSort(@CustomSortProc, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
  var ListItem : TListItem;
  begin
    ListView1.ViewStyle := vsList;
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Apples';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Oranges';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Pears';
  end;
  {
  This example shows how to use the OnColumnClick and
  OnCompare events of a list view to let users sort the
  columns in a report-style list view by clicking on the
  column headers. This requires a global variable to keep
  track of the column that was clicked.  The OnColumnClick
  event handler sets the global variable to indicate the
  column to sort and calls AlphaSort:
  }
  var ColumnToSort: Integer;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
    ListItem: TListItem;
    NewColumn: TListColumn;
  const
    imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
  begin
    Col2Array[0]:= 'Documentation for the C++ icon.';
    Col2Array[1]:= 'Borland icon.';
    Col2Array[2]:= 'Delphi icon.';
    // Create a ListView item for each image in the ImageList
    with ListView1 do
    begin
      SmallImages := ImageList1;
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := imagenames[I];
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      ListView1.ViewStyle := vsReport;
    end;
  end;

  procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
  begin
    ColumnToSort := Column.Index;
    (Sender as TCustomListView).AlphaSort;
  end;

  {
  The OnCompare event handler causes the list view to sort on
  the selected column:
  }
  procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
```

```
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

CompareStr (see page 4142)

AnsiCompareText (see page 4115)

CompareMem (see page 4142)

SameText (see page 4283)

39.3.58 SysUtils.CreateDir Function

Creates a new directory.

Pascal

```
function CreateDir(const Dir: string): Boolean;
```

C++

```
Boolean CreateDir(const UnicodeString Dir);
```

File

SysUtils

Description

CreateDir creates a new directory. The return value is true if a new directory was successfully created, or false if an error occurred.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
```

```
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 4230)

ExtractFileDir (see page 4176)

GetCurrentDir (see page 4246)

RemoveDir (see page 4276)

SetCurrentDir (see page 4283)

ForceDirectories (see page 4230)

39.3.59 SysUtils.CreateGUID Function

Creates a globally unique identifier.

Pascal

```
function CreateGUID(out Guid: TGUID): HResult; stdcall;
```

C++

```
__stdcall HRESULT CreateGUID(TGUID Guid);
```

File

SysUtils

Description

CreateGUID sets Guid to a new-created Globally Unique Identifier.

Note: Under Windows, CreateGUID calls the Windows API CoCreateGUID.

Note: Under Linux, CreateGUID calls uuid_generate_time.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
```

```
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
```

```
    CreateGUID(MyGuid0);

    { Testing if 2 guids are equal }
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
  end;
```

39.3.60 SysUtils.CurrentYear Function

Returns the current year.

Pascal

```
function CurrentYear: Word;
```

C++

```
Word CurrentYear();
```

File

SysUtils

Description

CurrentYear returns the current year on the Gregorian calendar. The value returned is the complete four-digit year.

39.3.61 SysUtils.CurrToStr Function

Formats a Currency (see page 3242) value as a string.

Pascal

```
function CurrToStr(Value: Currency): string; overload;
function CurrToStr(Value: Currency; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString CurrToStr(Currency Value);
UnicodeString CurrToStr(Currency Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

CurrToStr converts the Currency (see page 3242) value given by Value to its string representation. The conversion uses the ffGeneral floating point conversion rules.

The first form of CurrToStr is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.62 SysUtils.CurrToStrF Function

Converts a Currency (see page 3242) value to a string, using a specified format.

Pascal

```
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer): string; overload;
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer; const
FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString CurrToStrF(Currency Value, TFloatFormat Format, int Digits);
UnicodeString CurrToStrF(Currency Value, TFloatFormat Format, int Digits, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

CurrToStrF converts the Currency (see page 3242) value given by Value to its string representation using the indicated float format convention. A call to CurrToStrF corresponds to a call to FloatToStrF (see page 4226) with an implied precision of 19 digits.

The first form of CurrToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.63 SysUtils.Date Function

Returns the current date.

Pascal

```
function Date: TDateTime;
```

C++

```
TDateTime Date();
```

File

SysUtils

Description

Use Date to obtain the current local date as a TDateTime (see page 3335) value. The time portion of the value is 0 (midnight).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
```

See Also

DateToStr (see page 4156)

DecodeDate (see page 4159)

Now (see page 4271)

Time (see page 4338)

Today

Tomorrow

Yesterday

IsToday

39.3.64 SysUtils.DateTimeToFileDate Function

Converts a TDateTime (see page 3335) value to an OS timestamp value.

Pascal

```
function DateTimeToFileDate(DateTime: TDateTime): Integer;
```

C++

```
int DateTimeToFileDate(TDateTime DateTime);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the Operating System (see page 3165) to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use DateTimeToFileDate to convert a TDateTime (see page 3335) value to an OS timestamp value.

DateTimeToFileDate is particularly useful for converting the Time (see page 4338) field of the TSearchRec (see page 4371) type used by the FindFirst (see page 4218) and FindNext (see page 4221) functions.

See Also

FileDateToDateTime (see page 4191)

FindFirst (see page 4218)

FindNext (see page 4221)

TSearchRec (see page 4371)

39.3.65 SysUtils.DateTimeToStr Function

Converts a TDateTime (see page 3335) value to a string.

Pascal

```
function DateTimeToStr(const DateTime: TDateTime): string; overload;
function DateTimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings):
string; overload;
```

C++

```
UnicodeString DateTimeToStr(const TDateTime DateTime);
UnicodeString DateTimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

The first form of DateTimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

39.3.66 SysUtils.DateTimeToString Function

Converts a TDateTime (see page 3335) value to a string using a specified Format (see page 4232).

Pascal

```
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime);
```

```
overload;
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime;
const FormatSettings: TFormatSettings); overload;
```

C++

```
DateTimeToString(UnicodeString Result, const UnicodeString Format, TDateTime DateTime);
DateTimeToString(UnicodeString Result, const UnicodeString Format, TDateTime DateTime, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

DateTimeToString converts the TDateTime (see page 3335) value given by DateTime using the format string given by Format (see page 4232) into the string variable given by Result. See the table below for information about the supported format strings.

The first form of DateTimeToString is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToString, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToString, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

Date (see page 4151) Time (see page 4338) Format (see page 4232) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 4398) global variable, followed by the time using the format given by the LongTimeFormat (see page 4396) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 4399) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 4395) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 4398) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 4394) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3652)) using the strings given by the ShortMonthNames (see page 4399) global variable.

mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 4395) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
t	Displays the time using the format given by the ShortTimeFormat (see page 4400) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 4396) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 4402) global variable for any hour before noon, and the contents of the TimePMString (see page 4402) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 4389) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 4403) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

See Also

DateTimeToStr (see page 4153)

GetLocaleFormatSettings (see page 4250)

39.3.67 SysUtils.DateTimeToSystemTime Function

Converts a TDateTime (see page 3335) value into the Win32 API's system time type.

Pascal

```
procedure DateTimeToSystemTime(const DateTime: TDateTime; var SystemTime: TSystemTime);
```

C++

```
DateTimeToSystemTime(const TDateTime DateTime, TSystemTime SystemTime);
```

File

SysUtils

Description

Use DateTimeToSystemTime to convert a TDateTime (see page 3335) value into a TSystemTime (Delphi) or _SYSTEMTIME (C++) value. This procedure is primarily used to convert TDateTime (see page 3335) values for making a Windows API call.

See Also

SystemTimeToDateTime (see page 4336)

39.3.68 SysUtils.DateTimeToTimeStamp Function

Converts a TDateTime (see page 3335) value into the corresponding TTimeStamp (see page 4375) value.

Pascal

```
function DateTimeToTimeStamp(DateTime: TDateTime): TTimeStamp;
```

C++

```
TTimeStamp DateTimeToTimeStamp(TDateTime DateTime);
```

File

SysUtils

Description

Call DateTimeToTimeStamp to convert a TDateTime (see page 3335) value into a TTimeStamp (see page 4375) value.

See Also

TimeStampToDateTime (see page 4339)

39.3.69 SysUtils.DateToStr Function

Converts a TDateTime (see page 3335) value to a string.

Pascal

```
function DateToStr(const DateTime: TDateTime): string; overload;
function DateToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
UnicodeString DateToStr(const TDateTime DateTime);
UnicodeString DateToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use DateToStr to obtain a string representation of a date value that can be used for display purposes.

The first form of DateToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateToStr(Date());
}
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

39.3.70 SysUtils.DayOfWeek Function

Returns the day of the week for a specified date.

Pascal

```
function DayOfWeek(const DateTime: TDateTime): Word;
```

C++

```
Word DayOfWeek(const TDateTime DateTime);
```

File

SysUtils

Description

DayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

Note: DayOfWeek is not compliant with the ISO 8601 standard, which defines Monday as the first day of the week. For an ISO 8601 compliant version, use the DayOfTheWeek function instead.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

See Also

Date (see page 4151)

EncodeDate (see page 4169)

DayOfTheMonth

DayOfTheYear

DayOfTheWeek

39.3.71 SysUtils.DecodeDate Function

Returns Year, Month, and Day values for a TDateTime (see page 3335) value.

Pascal

```
procedure DecodeDate(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
Word);
```

C++

```
DecodeDate(const TDateTime DateTime, Word Year, Word Month, Word Day);
```

File

SysUtils

Description

The DecodeDate procedure breaks the value specified as the Date (see page 4151) parameter into Year, Month, and Day values. If the given TDateTime (see page 3335) value has a negative (BC) year, the year, month, and day return parameters are all set to zero.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
```

```
  the user clicks the button, the current date and time are
  reported in the captions of the two labels.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    Word Year, Month, Day, Hour, Min, Sec, MSec;
    TDateTime dtPresent = Now();
    DecodeDate(dtPresent, Year, Month, Day);
    Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
  IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
    DecodeTime(dtPresent, Hour, Min, Sec, MSec);
    Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
  ") + IntToStr(Hour);
  }
```

Delphi Examples:

```
  {
  This example uses a button and two labels on a form. When
  the user clicks the button, the current date and time are
  reported in the captions of the two labels.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Present: TDateTime;
    Year, Month, Day, Hour, Min, Sec, MSec: Word;
   begin
    Present:= Now;
    SysUtils.DecodeDate(Present, Year, Month, Day);
    Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
      + IntToStr(Month) + ' of Year ' + IntToStr(Year);
    SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
    Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
      + IntToStr(Hour);
  end;
```

See Also

DecodeDateFully (see page 4160)

DecodeTime (see page 4161)

DayOf

YearOf

MonthOf

DecodeDateTime

39.3.72 SysUtils.DecodeDateFully Function

Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3335) value.

Pascal

```
  function DecodeDateFully(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
  Word; var DOW: Word): Boolean;
```

C++

```
  Boolean DecodeDateFully(const TDateTime DateTime, Word Year, Word Month, Word Day, Word DOW);
```

File

SysUtils

Description

DecodeDateFully procedure breaks the value specified as the Date (see page 4151) parameter into Year, Month, Day, and DOW values. If Year is a leap year, the function return value is True; otherwise the function return value is false. If the given TDateTime (see page 3335) value has a negative (BC) year, the return parameters are all set to zero.

See Also

DecodeDate (see page 4159)

DecodeTime (see page 4161)

DayOf

YearOf

MonthOf

DecodeDateTime

39.3.73 SysUtils.DecodeTime Function

Breaks a TDateTime (see page 3335) value into hours, minutes, seconds, and milliseconds.

Pascal

```
procedure DecodeTime(const DateTime: TDateTime; var Hour: Word; var Min: Word; var Sec: Word;
var MSec: Word);
```

C++

```
DecodeTime(const TDateTime DateTime, Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

DecodeTime breaks the object specified as the Time (see page 4338) parameter into hours, minutes, seconds, and milliseconds.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Word Year, Month, Day, Hour, Min, Sec, MSec;
  TDateTime dtPresent = Now();
  DecodeDate(dtPresent, Year, Month, Day);
  Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
  DecodeTime(dtPresent, Hour, Min, Sec, MSec);
  Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
") + IntToStr(Hour);
}
```

Delphi Examples:

```
{
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  SysUtils.DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

See Also

DecodeDate (see page 4159)

EncodeTime (see page 4170)

Time (see page 4338)

HourOf

MinuteOf

SecondOf

MilliSecondOf

DecodeDateTime

39.3.74 SysUtils.DelegatesEqual Function

Pascal

```
function DelegatesEqual(A: Pointer; B: Pointer): Boolean;
```

C++

```
Boolean DelegatesEqual(void * A, void * B);
```

File

SysUtils

Description

This is function SysUtils.DelegatesEqual.

39.3.75 SysUtils.DeleteFile Function

Deletes a file from disk.

Pascal

```
function DeleteFile(const FileName: string): Boolean;
```

C++

```
Boolean DeleteFile(const UnicodeString FileName);
```

File

SysUtils

Description

DeleteFile deletes the file named by FileName from the disk. If the file cannot be deleted or does not exist, the function returns false.

C++ Examples:

```
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
/*
Click the button to open a TOpenDialog, then select a file
to delete.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
      DeleteFile(OpenDialog1->FileName);
  }
}
```

Delphi Examples:

```
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
```

```
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
```

39.3.76 SysUtils.DirectoryExists Function

Determines whether a specified directory exists.

Pascal

```
function DirectoryExists(const Directory: string): Boolean;
```

C++

```
Boolean DirectoryExists(const UnicodeString Directory);
```

File

SysUtils

Description

Call DirectoryExists to determine whether the directory specified by the **Directory** parameter exists. If the directory exists, the function returns true. If the directory does not exist, the function returns false.

If a full path name is entered, DirectoryExists searches for the directory along the designated path. Otherwise, the **Directory** parameter is interpreted as a relative path name from the current directory.

Note: The FileCtrl unit (Windows only) also contains a DirectoryExists function. However, the FileCtrl version is deprecated, and the SysUtils (see page 3967) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 4230)

SelectDirectory

FileExists

39.3.77 SysUtils.DiskFree Function

Returns the number of free bytes on a specified drive.

Pascal

```
function DiskFree(Drive: Byte): Int64;
```

C++

```
Int64 DiskFree(Byte Drive);
```

File

SysUtils

Description

DiskFree returns the number of free bytes on the specified drive, where 0 = Current, 1 = A, 2 = B, and so on.

DiskFree returns -1 if the drive number is invalid.

Note: DiskFree is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

```
/*
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ShowMessage(FormatFloat("#,##0 bytes", DiskFree(0)));
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

DiskSize (see page 4166)

39.3.78 SysUtils.DiskSize Function

Returns the size, in bytes, of a specified drive.

Pascal

```
function DiskSize(Drive: Byte): Int64;
```

C++

```
Int64 DiskSize(Byte Drive);
```

File

SysUtils

Description

DiskSize returns the size in bytes of the specified drive, where 0 = Current, 1 = A, 2 = B, etc. DiskSize returns -1 if the drive number is invalid.

Note: DiskSize is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
```

See Also

DiskFree (see page 4165)

39.3.79 SysUtils.DisposeStr Function

Disposes of a string pointer that was allocated using NewStr (see page 4271).

Pascal

```
procedure DisposeStr(P: PAnsiString); deprecated;
```

C++

```
DisposeStr(PAnsiString P);
```

File

SysUtils

Description

DisposeStr is only provided for backward compatibility with existing code that allocated a string using NewStr (see page 4271). Both DisposeStr and NewStr (see page 4271) are obsolete.

See Also

NewStr (see page 4271)

39.3.80 SysUtils.ElementToCharIndex Function

Pascal

```
function ElementToCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int ElementToCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharIndex.

39.3.81 SysUtils.ElementToCharIndex Function

Pascal

```
function ElementToCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int ElementToCharIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharIndex.

39.3.82 SysUtils.ElementToCharLen Function

Pascal

```
function ElementToCharLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
```

C++

```
int ElementToCharLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharLen.

39.3.83 SysUtils.ElementToCharLen Function

Pascal

```
function ElementToCharLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int ElementToCharLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharLen.

39.3.84 SysUtils.EncodeDate Function

Returns a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.

Pascal

```
function EncodeDate(Year: Word; Month: Word; Day: Word): TDateTime;
```

C++

```
TDateTime EncodeDate(Word Year, Word Month, Word Day);
```

File

SysUtils

Description

EncodeDate returns a TDateTime (see page 3335) value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

If the specified values are not within range, EncodeDate raises an EConvertError (see page 3985) exception.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
```

```
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

See Also

DateToStr (see page 4156)

DecodeDate (see page 4159)

EncodeTime (see page 4170)

EncodeDateTime

TryEncodeDate (see page 4343)

39.3.85 SysUtils.EncodeTime Function

Returns a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function EncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word): TDateTime;
```

C++

```
TDateTime EncodeTime(Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

EncodeTime encodes the given hour, minute, second, and millisecond into a TDateTime (see page 3335) value.

Valid Hour values are 0 through 23.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

If the specified values are not within range, EncodeTime raises an EConvertError (see page 3985) exception.

The resulting value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
```

See Also

DecodeTime (see page 4161)

EncodeDate (see page 4169)

EConvertError (see page 3985)

EncodeDateTime

39.3.86 SysUtils.ExceptionErrorMessage Function

Formats a standard error message.

Pascal

```
function ExceptionErrorMessage(ExceptObject: TObject; ExceptAddr: Pointer; Buffer: PChar;
```

```
  Size: Integer): Integer;
```

C++

```
int ExceptionErrorMessage(TObject * ExceptObject, void * ExceptAddr, const char * Buffer, int
Size);
```

File

SysUtils

Description

Call ExceptionErrorMessage to assemble an error message that indicates the object associated with the current exception and the location at which the exception was raised. The assembled message is written into the buffer provided by the Buffer parameter. Specify the size of this buffer using the Size parameter.

ExceptionErrorMessage returns the number of bytes in the assembled error message.

See Also

ExceptAddr (see page 3663)

ExceptObject (see page 3664)

ShowException (see page 4284)

39.3.87 SysUtils.ExcludeTrailingBackslash Function

Returns a path name without a trailing delimiter.

Pascal

```
function ExcludeTrailingBackslash(const S: string): string; overload; platform;
```

C++

```
UnicodeString ExcludeTrailingBackslash(const UnicodeString S);
```

File

SysUtils

Description

ExcludeTrailingBackslash returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. This function is included for backward compatibility only. New (see page 3712) programs should use ExcludeTrailingPathDelimiter (see page 4173).

Note: ExcludeTrailingBackslash works with multibyte character sets.

See Also

IncludeTrailingBackslash (see page 4257)

IsPathDelimiter (see page 4266)

ExcludeTrailingPathDelimiter (see page 4173)

39.3.88 SysUtils.ExcludeTrailingPathDelimiter Function

Returns a path name without a trailing delimiter.

Pascal

```
function ExcludeTrailingPathDelimiter(const S: string): string; overload;
```

C++

```
UnicodeString ExcludeTrailingPathDelimiter(const UnicodeString S);
```

File

SysUtils

Description

ExcludeTrailingPathDelimiter returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. If the last character in S is a delimiter, the return value is S with the last character removed. Otherwise, S is returned unchanged.

Note: ExcludeTrailingPathDelimiter works with multibyte character sets.

See Also

IncludeTrailingPathDelimiter (see page 4257)

IsPathDelimiter (see page 4266)

39.3.89 SysUtils.ExpandFileName Function

Returns the full path name for a relative file name.

Pascal

```
function ExpandFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExpandFileName(const UnicodeString FileName);
```

File

SysUtils

Description

ExpandFileName converts the relative file name into a fully qualified path name. ExpandFileName does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

Note: ExpandFileName does not support UNC format. To expand to a UNC file name, use ExpandUNCFileName (see page 4175).

C++ Examples:

```
/*
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
specified in the edit control is expanded to a fully
```

```
qualified filename and added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items->Add(ExpandFileName(Edit1->Text));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = ExtractFilePath(Application->ExeName);
}
```

Delphi Examples:

```
{
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
specified in the edit control is expanded to a fully
qualified filename and added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(SysUtils.ExpandFileName(Edit1.Text));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := ExtractFilePath(Application.ExeName);
end;
```

See Also

ExtractFileName (see page 4177)

ExpandUNCFileName (see page 4175)

ExpandFileNameCase (see page 4174)

39.3.90 SysUtils.ExpandFileNameCase Function

Returns the full path name for a relative file name on a case-sensitive file system.

Pascal

```
function ExpandFileNameCase(const FileName: string; out MatchFound: TFilenameCaseMatch):
string; overload;
```

C++

```
UnicodeString ExpandFileNameCase(const UnicodeString FileName, TFilenameCaseMatch MatchFound);
```

File

SysUtils

Description

ExpandFileNameCase expands a relative file name into a fully qualified path name, then attempts to find the file or a close case-insensitive match. This is useful when converting file name data from case-insensitive conventions (Windows) to case-sensitive conventions (Linux). It is also useful for validating file names entered by users unfamiliar with Linux conventions.

FileName is the relative file name to expand.

MatchFound indicates what match, if any, was found for FileName.

The return value is the fully qualified path name generated by ExpandFileNameCase. Its relationship to FileName depends on the value of MatchFound, as indicated in the following table:

MatchFound	Return value
mkNone	The fully qualified path name, as returned by ExpandFileName (see page 4173).
mkExactMatch	The fully qualified path name for the single case-sensitive match to FileName. This is the same value as the one returned when MatchFound is mkNone.
mkSingleMatch	(Linux only.) The fully qualified path name for the single file that matches FileName case-insensitively.
mkAmbiguous	(Linux only.) The fully qualified path name for one of the case-insensitve matches to FileName. This path name is not the same as the specified file, and not necessarily the closest match.

Note: ExpandFileNameCase is much slower than ExpandFileName (see page 4173), particularly when the specified file name is ambiguous or does not match an actual file.

See Also

ExpandFileName (see page 4173)

39.3.91 SysUtils.ExpandUNCFileName Function

Returns the full path of a file name in UNC format, if appropriate.

Pascal

```
function ExpandUNCFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExpandUNCFileName(const UnicodeString FileName);
```

File

SysUtils

Description

ExpandUNCFileName returns the fully-qualified file name for a specified file name.

Note: On Windows, ExpandUNCFileName returns a fully qualified path name using Universal Naming Convention for network files. For files on local disks, ExpandUNCFileName is equivalent to ExpandFileName (see page 4173). In a UNC file name, the drive prefix is replaced with the name of the network resource to which the drive is mapped. For example, if the working directory is J:\ourdir and drive J is mapped to \\ourserver\ourshare, ExpandUNCFileName('ourfile') returns "\\ourserver\ourshare\ourdir\ourfile".

Note: On Linux, ExpandUNCFileName is always equivalent to ExpandFileName (see page 4173).

See Also

ExpandFileName (see page 4173)

ExtractFileDrive (see page 4176)

39.3.92 SysUtils.ExtractFileDir Function

Extracts the drive and directory parts from FileName.

Pascal

```
function ExtractFileDir(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileDir(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is a directory name suitable for passing to the CreateDir (see page 4147), GetCurrentDir (see page 4246), RemoveDir (see page 4276), and SetCurrentDir (see page 4283) functions. This string is empty if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

See Also

CreateDir (see page 4147)

GetCurrentDir (see page 4246)

RemoveDir (see page 4276)

SetCurrentDir (see page 4283)

39.3.93 SysUtils.ExtractFileDrive Function

Returns the drive portion of a file name.

Pascal

```
function ExtractFileDrive(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileDrive(const UnicodeString FileName);
```

File

SysUtils

Description

Note: On Windows, ExtractFileDrive returns a string containing the drive portion of a fully qualified path name for the file passed in the FileName. For file names with drive letters, the result is in the form "'<drive>'". For file names with a UNC path the result is in the form "'\\<servername>\<sharename>'". If the given path contains neither style of path prefix, the result is an empty string.

Note: On Linux, ExtractFileDrive always returns a zero-length string.

See Also

ExpandFileName (see page 4173)

ExpandUNCFileName (see page 4175)

39.3.94 SysUtils.ExtractFileExt Function

Returns the extension portions of a file name.

Pascal

```
function ExtractFileExt(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileExt(const UnicodeString FileName);
```

File

SysUtils

Description

Use ExtractFileExt to obtain the extension from a file name. For example, the following code returns the extension of the file name specified by a variable named MyFileName:

MyFilesExtension := ExtractFileExt(MyFileName); // Delphi

MyFilesExtension = ExtractFileExt(MyFileName); // C++

The resulting string includes the period character that separates the name and extension parts. This string is empty if the given file name has no extension.

Note: This function works for multi-byte character systems (MBCS).

See Also

ExtractFileName (see page 4177)

39.3.95 SysUtils.ExtractFileName Function

Extracts the name and extension parts of a file name.

Pascal

```
function ExtractFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileName(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is the rightmost characters of FileName, starting with the first character after the colon or backslash that separates the path information from the name and extension. The resulting string is equal to FileName if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
```

```
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

ExpandFileName (see page 4173)

ExtractFilePath (see page 4179)

39.3.96 SysUtils.ExtractFilePath Function

Returns the drive and directory portions of a file name.

Pascal

```
function ExtractFilePath(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFilePath(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is the leftmost characters of FileName, up to and including the colon or backslash that separates the path information from the name and extension. The resulting string is empty if FileName contains no drive and directory parts.

This function works for multi-byte character systems (MBCS).

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
```

```
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  temp1 := Edit4.Text;
  temp2 := ExtractFileName(Edit1.Text);
```

```
    NewFileName := temp1 + temp2;
    OldFileName :=
      Edit3.Text + ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
  {
  The following code prompts for confirmation before deleting
  a file.  Create a file and then enter the path and file name
  in the textfields.  Click the button to delete it.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var FileName: string;
  begin
    Filename:= Edit1.Text;
    if SysUtils.FileExists(FileName) then
    begin
      if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
  mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
        DeleteFile(FileName);
    end
    else
      MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
  [mbOK], 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Edit2.Text:= ExtractFilePath(Application.ExeName);
  end;
```

See Also

ExtractFileName (see page 4177)

39.3.97 SysUtils.ExtractRelativePath Function

Returns a relative path name, relative to a specific base directory.

Pascal

```
function ExtractRelativePath(const BaseName: string; const DestName: string): string; overload;
```

C++

```
UnicodeString ExtractRelativePath(const UnicodeString BaseName, const UnicodeString DestName);
```

File

SysUtils

Description

Call ExtractRelativePath to convert a fully qualified path name into a relative path name. The `DestName` parameter specifies file name (including path) to be converted. `BaseName` is the fully qualified name of the base directory to which the returned path name should be relative. `BaseName` may or may not include a file name, but it must include the final path delimiter.

Note: If you send it what should be a directory without the trailing backslash (c:\foo

instead of `c:\foo\`), the function treats `foo` as a file and returns undesirable results. You should correctly call it like this:

```
ExtractRelativePath('c:\foo\', 'c:\bar\')
```

so that it would return

```
'..\bar\'
```

.

```
ExtractRelativePath('c:\foo\', 'c:\bar')
```

would return the incorrect result, probably like

```
'..'
```

.

See Also

ExpandFileName (see page 4173)

39.3.98 SysUtils.ExtractShortPathName Function

Converts a file name to the short 8.3 form.

Pascal

```
function ExtractShortPathName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractShortPathName(const UnicodeString FileName);
```

File

SysUtils

Description

ExtractShortPathName converts the file name, FileName, to the short 8.3 form. ExtractShortPathName returns an empty string if the file or directory does not exist.

The first parameter, FileName, can be a full path name. For example, ExtractShortPathName would convert the full path name:

C:\Program Files\MyCompany\MyApp\MyApp.exe

to the short form:

C:\Progra~1\MyComp~1\MyApp\MyApp.exe

Note: ExtractShortPathName is only available on Windows.

See Also

ExpandFileName (see page 4173)

ExpandUNCFileName (see page 4175)

39.3.99 SysUtils.FileAge Function

Returns the OS timestamp of a file.

Pascal

```
function FileAge(const FileName: string): Integer; overload; deprecated;
function FileAge(const FileName: string; out FileDateTime: TDateTime): Boolean; overload;
```

C++

```
int FileAge(const UnicodeString FileName);
Boolean FileAge(const UnicodeString FileName, TDateTime FileDateTime);
```

File

SysUtils

Description

Call FileAge to obtain the OS timestamp of the file specified by FileName. The return value can be converted to a TDateTime (see page 3335) object using the FileDateToDateTime (see page 4191) function. The return value is -1 if the file does not exist.

Note: :Under Linux, -1 is a valid timestamp. Use FileExists (see page 4193) to verify that the file does not exist.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
```

```
        NewAttributes = NewAttributes | faReadOnly;
      else
         NewAttributes = NewAttributes & ~faReadOnly;

      if (FileAttrDlg->Archive->Checked)
        NewAttributes = NewAttributes | faArchive;
      else
        NewAttributes = NewAttributes & ~faArchive;

      if (FileAttrDlg->System->Checked)
        NewAttributes = NewAttributes | faSysFile;
      else
        NewAttributes = NewAttributes & ~faSysFile;

      if (FileAttrDlg->Hidden->Checked)
        NewAttributes = NewAttributes | faHidden;
      else
        NewAttributes = NewAttributes  & ~faHidden;
      if (NewAttributes != Attributes)
        FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
```

```
        if Hidden.Checked then
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

FileDateToDateTime (see page 4191)

FileGetDate (see page 4199)

FileExists (see page 4193)

39.3.100 SysUtils.FileClose Function

Closes a specified file.

Pascal

```
procedure FileClose(Handle: Integer);
```

C++

```
FileClose(int Handle);
```

File

SysUtils

Description

FileClose closes a file given its handle. The handle is obtained when the file is opened using FileOpen (see page 4200) or FileCreate (see page 4188).

Note: : When working with Delphi language file variables, use the CloseFile (see page 3643) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
```

```
    {
      fnsplit(AnsiString(SaveDialog1->FileName).t_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
```

```
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  This example uses a file listbox and a regular listbox on a
  form. The following routine scans through the files listed
  in the file listbox and lists the sizes of any selected
  files to the regular list box.  To exercise the error
  condition create a file in the Debug directory, start this
  application and then delete the file.  Now try to list the
  size of the deleted file.  Set the MultiSelect and
  ExtendedSelect properties on the FileListBox.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
```

```
    F: File;
    i, filehandle: Integer;
  begin
    for i := 0 to (FileListBox1.Items.Count - 1) do begin
    try
      if FileListBox1.Selected[i] then
      begin
        if not FileExists(FileListBox1.Items.Strings[i]) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' not found', mtError, [mbOk], 0);
          Continue;
        end;
        filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
     finally
     { do something here }
     end;
    end;
  end;
```

See Also

FileCreate (see page 4188)

FileOpen (see page 4200)

CloseFile (see page 3643)

39.3.101 SysUtils.FileCreate Function

Creates a new file.

Pascal

```
function FileCreate(const FileName: string): Integer; overload;
function FileCreate(const FileName: string; Rights: Integer): Integer; overload;
```

C++

```
int FileCreate(const UnicodeString FileName);
int FileCreate(const UnicodeString FileName, int Rights);
```

File

SysUtils

Description

FileCreate creates a new file with the specified name. If the return value is positive, the function was successful and the value is

the file handle of the new file. A return value of -1 indicates that an error occurred.

Note: On Windows, the FileAccessRights variable and Rights parameter are ignored.

Note: On Linux, a file has a set of permission bits controlling access to the file. The first form of FileCreate takes the permission bits from the global FileAccessRights variable. The second form takes the permission bits from the Rights parameter. In either case, the permission bits are masked by the current process's umask value.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).t_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
```

```
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

FileClose (see page 4185)

FileOpen (see page 4200)

AssignFile (see page 3629)

39.3.102 SysUtils.FileDateToDateTime Function

Converts an OS timestamp value to TDateTime (see page 3335) value.

Pascal

```
function FileDateToDateTime(FileDate: Integer): TDateTime;
```

C++

```
TDateTime FileDateToDateTime(int FileDate);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use FileDateToDateTime to convert a timestamp to a TDateTime (see page 3335) value.

You may need to use FileDateToDateTime to convert the Time (see page 4338) field of the TSearchRec (see page 4371) type used by the FindFirst (see page 4218) and FindNext (see page 4221) functions.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
```

```
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
```

```
      Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
      { read file attributes }
      ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
      Archive.Checked := (Attributes and faArchive) = faArchive;
      System.Checked := (Attributes and faSysFile) = faSysFile;
      Hidden.Checked := (Attributes and faHidden) = faHidden;
      if ShowModal <> id_Cancel then    { execute dialog box }
      begin
        NewAttributes := Attributes;
        { start with original attributes }
        if ReadOnly.Checked then
          NewAttributes := NewAttributes or SysUtils.faReadOnly
        else
          NewAttributes := NewAttributes and not SysUtils.faReadOnly;
        if Archive.Checked then
          NewAttributes := NewAttributes or faArchive
        else
          NewAttributes := NewAttributes and not faArchive;
        if System.Checked then
          NewAttributes := NewAttributes or faSysFile
        else
          NewAttributes := NewAttributes and not faSysFile;
        if Hidden.Checked then
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

DateTimeToFileDate (see page 4152)

FindFirst (see page 4218)

FindNext (see page 4221)

39.3.103 SysUtils.FileExists Function

Tests if a specified file exists.

Pascal

```
function FileExists(const FileName: string): Boolean;
```

C++

```
Boolean FileExists(const UnicodeString FileName);
```

File

SysUtils

Description

FileExists returns true if the file specified by FileName exists. If the file does not exist, FileExists returns false.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
```

```
  Save dialog box on a form. When the button is clicked, the
  user is prompted for a filename. When the user clicks OK,
  the contents of the string grid are written to the specified
  file. Additional information is also written to the file so
  that it can be read easily with the FileRead function.
  */

  #include <dir.h>

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    char szFileName[MAXFILE+4];
    int iFileHandle;
    int iLength;
    if (SaveDialog1->Execute())
    {
      if (FileExists(SaveDialog1->FileName))
      {
        fnsplit(AnsiString(SaveDialog1->FileName).t_str(), 0, 0, szFileName, 0);
        strcat(szFileName, ".BAK");
        RenameFile(SaveDialog1->FileName, szFileName);
      }
      iFileHandle = FileCreate(SaveDialog1->FileName);
      // Write out the number of rows and columns in the grid.
      FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
  sizeof(StringGrid1->ColCount));
      FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
  sizeof(StringGrid1->RowCount));
      for (int x=0;x<StringGrid1->ColCount;x++)
      {
        for (int y=0;y<StringGrid1->RowCount;y++)
        {
          // Write out the length of each string, followed by the string itself.
          iLength = StringGrid1->Cells[x][y].Length();
          FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
          FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
  StringGrid1->Cells[x][y].Length());
        }
      }
      FileClose(iFileHandle);
    }
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    StringGrid1->Cells[1][0] = "Column 1";
    StringGrid1->Cells[2][0] = "Column 2";
    StringGrid1->Cells[3][0] = "Column 3";
    StringGrid1->Cells[4][0] = "Column 4";
    StringGrid1->Cells[0][1] = "Row 1";
    StringGrid1->Cells[1][1] = "Object";
    StringGrid1->Cells[2][1] = "Pascal";
    StringGrid1->Cells[3][1] = "is";
    StringGrid1->Cells[4][1] = "excellent";
    StringGrid1->Cells[0][2] = "Row 2";
    StringGrid1->Cells[1][2] = "Delphi";
    StringGrid1->Cells[2][2] = "is";
    StringGrid1->Cells[4][2] = "RAD";
  };
  /*
  The following code prompts for confirmation before deleting
  a file.  Create a file and then click on the button to select
  the file with a TOpenDialog.  Click Yes to delete it.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
```

```
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
```

```
            Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
```

See Also

FileSearch (see page 4206)

DirectoryExists (see page 4164)

39.3.104 SysUtils.FileGetAttr Function

Returns the file attributes of FileName.

Pascal

```
function FileGetAttr(const FileName: string): Integer; platform;
```

C++

```
int FileGetAttr(const UnicodeString FileName);
```

File

SysUtils

Description

FileGetAttr returns the attributes of the file as a string of bits. This value is the same as the Attr field of a TSearchRec (see page 4371) type. Check for individual attributes with code such as the following:

Attrs := FileGetAttr('MyFile.sys');

```
if Attrs and faHidden <> 0 then
  FileSetAttr('MyFile.sys', Attrs – faHidden);
```

int Attrs = FileGetAttr("MyFile.sys");

```
if (Attrs & faHidden)
  FileSetAttr("MyFile.sys", Attrs & !faHidden);
```

A return value of INVALID_FILE_ATTRIBUTES indicates that an error occurred.

Note: See TSearchRec (see page 4371) for a description of the individual attribute constants.

Note: FileGetAttr is Windows-only.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
```

```
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
```

```
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

FileSetAttr (see page 4209)

39.3.105 SysUtils.FileGetDate Function

Returns an OS timestamp for a specified file.

Pascal

```
function FileGetDate(Handle: Integer): Integer;
```

C++

```
int FileGetDate(int Handle);
```

File

SysUtils

Description

Use FileGetDate to determine the timestamp of a file given its handle. The return value is -1 if the handle is invalid. The FileDateToDateTime (see page 4191) function can be used to convert the returned value to a TDateTime (see page 3335) value.

Note: :FileGetDate uses a file handle rather than a native Delphi file variable. To obtain the timestamp of a file that has not been opened using the FileOpen (see page 4200) or FileCreate (see page 4188) method, use the FileAge (see page 4183) function instead.

Note: :Under Linux, -1 is a valid timestamp. Use FileExists (see page 4193) to verify that the file does not exist.

See Also

FileSetDate (see page 4212)

FileDateToDateTime (see page 4191)

FileAge (see page 4183)

FileExists (see page 4193)

39.3.106 SysUtils.FileIsReadOnly Function

Reports whether a file is read-only.

Pascal

```
function FileIsReadOnly(const FileName: string): Boolean;
```

C++

```
Boolean FileIsReadOnly(const UnicodeString FileName);
```

File

SysUtils

Description

FileIsReadOnly returns **True** if FileName can be found and opened, but the current process does not have permission to write to it.

Note: On Windows XP, FileIsReadOnly returns **True** if the file does **not** exist.

39.3.107 SysUtils.FileOpen Function

Opens a specified file using a specified access mode.

Pascal

```
function FileOpen(const FileName: string; Mode: LongWord): Integer;
```

C++

```
int FileOpen(const UnicodeString FileName, LongWord Mode);
```

File

SysUtils

Description

Use FileOpen to open a file and obtain a file handle. The access mode value is constructed by or-ing one of the fmOpen constants with one of the fmShare constants defined in File open mode constants. If the return value is 0 or greater, the function was successful and the value is the file handle of the opened file. A return value of -1 indicates that an error occurred.

Note: Use of the non-native Delphi language file handlers such as FileOpen is not encouraged. These routines map to system routines and return OS file handles, not normal Delphi file variables. These are low-level file access routines. For normal file operations use AssignFile (see page 3629), Rewrite (see page 3736), and Reset (see page 3734) instead.

C++ Examples:

```
/*
This example requires a TEdit, a button and a string grid.
Enter the file name in the TEdit and click the button to open
that file.  Notice that the file cannot be read.  Remove the
fmOpenWrite and the fmShareDenyNone access and the string
grid loads fine.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  try
  {
    iFileHandle = FileOpen(Edit1->Text, fmOpenRead | fmOpenWrite | fmShareDenyNone);
    if (iFileHandle > 0) {
      iFileLength = FileSeek(iFileHandle, 0, 2);
      FileSeek(iFileHandle,0,0);
      std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);

      for (int i = 0; i < iBytesRead; i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    else
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
  catch(...)
  {
    Application->MessageBox(
      L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
      L"File Error", IDOK);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
```

```
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example requires a TEdit, a button and a string grid.
Enter the file name in the TEdit and click the button to open
```

```
that file.  Notice that the file cannot be read.  Remove the
fmOpenWrite and the fmShareDenyNone access and the string
grid loads fine.
}
procedure OpenForShare(const FileName: String);
var
  FileHandle : Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  FileHandle := SysUtils.FileOpen(FileName, fmOpenWrite or fmShareDenyNone or fmOpenRead);
  if FileHandle > 0 then
  begin
    try
      MessageDlg('Valid file handle.', mtInformation, [mbOk], 0);
      iFileLength := SysUtils.FileSeek(FileHandle,0,2);
      FileSeek(FileHandle, 0, 0);
      Buffer := PAnsiChar(System.AllocMem(iFileLength + 1));
      iBytesRead := SysUtils.FileRead(FileHandle, Buffer^, iFileLength);
      for i := 0 to iBytesRead-1 do
      begin
        Form1.StringGrid1.RowCount := Form1.StringGrid1.RowCount + 1;
        Form1.StringGrid1.Cells[1, i + 1] := Buffer[i];
        Form1.StringGrid1.Cells[2, i + 1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end
  else
    MessageDlg('Open error: FileHandle = negative DOS error code.', mtInformation, [mbOk], 0);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenForShare(Edit1.Text);
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
```

```
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

FileClose (see page 4185)

FileCreate (see page 4188)

AssignFile (see page 3629)

Rewrite (see page 3736)

Reset (see page 3734)

39.3.108 SysUtils.FileRead Function

Reads a specified number of bytes from a file.

Pascal

```
function FileRead(Handle: Integer; var Buffer; Count: LongWord): Integer;
```

C++

```
int FileRead(int Handle,  Buffer, LongWord Count);
```

File

SysUtils

Description

FileRead reads Count bytes from the file specified by Handle into the buffer. The Count parameter indicates the size, in bytes, of the buffer. The function result is the actual number of bytes read, which may be less than Count. The Handle that is passed to FileRead must be opened with FileOpen (see page 4200) or FileCreate (see page 4188).

Note: :In Delphi, do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To read from a file specified by a Delphi file variable, use the BlockRead (see page 3633) procedure or the Read (see page 3727) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/
```

```
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
```

```
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
```

See Also

BlockRead (see page 3633)

FileOpen (see page 4200)

FileCreate (see page 4188)

FileSeek (see page 4207)

FileWrite (see page 4213)

Read (see page 3727)

Write (see page 3789)

39.3.109 SysUtils.FileSearch Function

Searches a specified directory path for a file.

Pascal

```
function FileSearch(const Name: string; const DirList: string): string;
```

C++

```
UnicodeString FileSearch(const UnicodeString Name, const UnicodeString DirList);
```

File

SysUtils

Description

FileSearch searches through the directories passed in DirList for a file named Name. DirList is a list of path names delimited by semicolons on Windows, colons on Linux). If FileSearch locates a file matching Name, it returns a string specifying a path name for that file. If no matching file exists, FileSearch returns an empty string.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString asFileName = FileSearch(Edit1->Text, GetCurrentDir());
  if (asFileName.IsEmpty())
    ShowMessage(AnsiString("Couldn't find ") + Edit1->Text + ".");
  else
    ShowMessage(AnsiString("Found ") + asFileName + ".");
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = GetCurrentDir();
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FileToFind: string;
begin
  FileToFind := SysUtils.FileSearch(Edit1.Text, GetCurrentDir);
  if FileToFind = '' then
    ShowMessage('Couldn''t find ' + Edit1.Text + '.')
  else
    ShowMessage('Found ' + FileToFind + '.');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := GetCurrentDir;
end;
```

See Also

FileExists (see page 4193)

39.3.110 SysUtils.FileSeek Function

Repositions read/write point.

Pascal

```
function FileSeek(Handle: Integer; Offset: Integer; Origin: Integer): Integer; overload;
function FileSeek(Handle: Integer; const Offset: Int64; Origin: Integer): Int64; overload;
```

C++

```
int FileSeek(int Handle, int Offset, int Origin);
Int64 FileSeek(int Handle, const Int64 Offset, int Origin);
```

File

SysUtils

Description

Use FileSeek to reposition the read/write point in a file that was opened with FileOpen (see page 4200) or FileCreate (see page 4188). Handle is the file handle that was returned by FileOpen (see page 4200) or FileCreate (see page 4188).

Offset specifies the number of bytes from Origin where the file pointer should be positioned. Origin is a code with three possible values, denoting the beginning of the file, the end of the file, and the current position of the file pointer.

Origin	Action
0	The file pointer is positioned Offset bytes from the beginning of the file.
1	The file pointer is positioned Offset bytes from its current position.

2	The file pointer is positioned Offset bytes from the end of the file.

If FileSeek is successful, it returns the new position of the file pointer; otherwise, it returns -1.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To move the file pointer in a file specified by a Delphi file variable, use the Seek (see page 3741) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
```

```
  grid. The first column contains the character values in the
  buffer. The second column contains the numeric values of the
  characters in the buffer.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    iFileHandle: Integer;
    iFileLength: Integer;
    iBytesRead: Integer;
    Buffer: PAnsiChar;
    i: Integer;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
        Buffer := System.AllocMem(iFileLength + 1);
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

FileOpen (see page 4200)

FileCreate (see page 4188)

FileRead (see page 4204)

FileWrite (see page 4213)

Seek (see page 3741)

39.3.111 SysUtils.FileSetAttr Function

Sets the file attributes of a specified file.

Pascal

```
function FileSetAttr(const FileName: string; Attr: Integer): Integer; platform;
```

C++

```
int FileSetAttr(const UnicodeString FileName, int Attr);
```

File

SysUtils

Description

FileSetAttr sets the file attributes of the file given by FileName to the value given by Attr. The value of Attr is formed by combining the appropriate file attribute constants, as in the following:

```
FileSetAttr('MyFile.sys', faReadOnly or faSysFile); // Delphi
FileSetAttr("MyFile.sys", faReadOnly | faSysFile); // C++
```

FileSetAttr returns zero if the function was successful. Otherwise the return value is an error code.

Note: See TSearchRec (see page 4371) for a description of the file attribute constants.

Note: FileSetAttr is only available on Windows.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

FileGetAttr (see page 4197)

TSearchRec (see page 4371)

39.3.112 SysUtils.FileSetDate Function

Sets the OS time stamp for a specified file.

Pascal

```
function FileSetDate(const FileName: string; Age: Integer): Integer; overload;
function FileSetDate(Handle: Integer; Age: Integer): Integer; overload; platform;
```

C++

```
int FileSetDate(const UnicodeString FileName, int Age);
int FileSetDate(int Handle, int Age);
```

File

SysUtils

Description

FileSetDate sets the OS timestamp of the specified file to the value given by Age. The DateTimeToFileDate (see page 4152) function can be used to convert a TDateTime (see page 3335) value to a OS timestamp.

Handle is the Windows file handle of the file to alter. This syntax is only available on Windows.

FileName is the name of the file to alter.

Age is the timestamp to apply to the specified file.

The return value is zero if the function was successful. Otherwise the return value is an error code.

See Also

FileCreate (see page 4188)

FileGetDate (see page 4199)

FileOpen (see page 4200)

DateTimeToFileDate (see page 4152)

39.3.113 SysUtils.FileSetReadOnly Function

Sets the file permissions to read-only.

Pascal

```
function FileSetReadOnly(const FileName: string; ReadOnly: Boolean): Boolean;
```

C++

```
Boolean FileSetReadOnly(const UnicodeString FileName, Boolean ReadOnly);
```

File

SysUtils

Description

FileSetReadOnly attempts to set the read-only status of the specified file.

FileName is the name of the file to alter.

ReadOnly indicates the status to assign to the file.

FileSetReadOnly returns true if the operation was successful and false if an error occurred.

Note: On Windows, the ReadOnly parameter is assigned to the file's read-only flag.

Note: On Linux, the inverse of the ReadOnly parameter is assigned to the file's user write, group write, and other write permission bits. The active user must have permission to perform these operations.

FileSetReadOnly is designed to support cross-platform code. To manipulate permissions more specifically than FileSetReadOnly allows, use platform-specific functions.

39.3.114 SysUtils.FileWrite Function

Writes the contents of a buffer to the current position in a file.

Pascal

```
function FileWrite(Handle: Integer; const Buffer; Count: LongWord): Integer;
```

C++

```
int FileWrite(int Handle, const  Buffer, LongWord Count);
```

File

SysUtils

Description

FileWrite writes Count bytes to the file given by Handle from the buffer specified by Buffer. Handle is a file handle returned by the FileOpen (see page 4200) or FileCreate (see page 4188) method.

The return value is the number of bytes actually written, or -1 if an error occurred.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To write to a file specified by a Delphi file variable, use Write (see page 3789), Writeln, or BlockWrite (see page 3634) instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
```

```
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).t_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
```

```
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

FileRead (see page 4204)

FileSeek (see page 4207)

Writeln

Write (see page 3789)

BlockWrite (see page 3634)

39.3.115 SysUtils.FinalizePackage Function

Finalizes the specified package (.bpl file).

Pascal

```
procedure FinalizePackage(Module: HMODULE);
```

C++

```
FinalizePackage(HMODULE Module);
```

File

SysUtils

Description

Most applications do not need to call FinalizePackage, because UnloadPackage (see page 4352) calls FinalizePackage internally. FinalizePackage executes any finalization code in the specified package.

See Also

UnloadPackage (see page 4352)

39.3.116 SysUtils.FindClose Function

Releases memory allocated by FindFirst (see page 4218).

Pascal

```
procedure FindClose(var F: TSearchRec);
```

C++

```
FindClose(TSearchRec F);
```

File

SysUtils

Description

FindClose terminates a FindFirst (see page 4218)/FindNext (see page 4221) sequence.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
```

```
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
```

```
          if (sr.Attr and FileAttrs) = sr.Attr then
          begin
          RowCount := RowCount + 1;
          Cells[1,RowCount-1] := sr.Name;
          Cells[2,RowCount-1] := IntToStr(sr.Size);
          end;
        until FindNext(sr) <> 0;
        FindClose(sr);
      end;
    end;
  end;
```

See Also

FindFirst (see page 4218)

FindNext (see page 4221)

39.3.117 SysUtils.FindCmdLineSwitch Function

Determines whether a string was passed as a command line argument to the application.

Pascal

```
function FindCmdLineSwitch(const Switch: string; const Chars: TSysCharSet; IgnoreCase:
Boolean): Boolean; overload;
function FindCmdLineSwitch(const Switch: string): Boolean; overload;
function FindCmdLineSwitch(const Switch: string; IgnoreCase: Boolean): Boolean; overload;
```

C++

```
Boolean FindCmdLineSwitch(const UnicodeString Switch, const TSysCharSet Chars, Boolean
IgnoreCase);
Boolean FindCmdLineSwitch(const UnicodeString Switch);
Boolean FindCmdLineSwitch(const UnicodeString Switch, Boolean IgnoreCase);
```

File

SysUtils

Description

FindCmdLineSwitch determines whether a string represents a command-line switch, based on the first character of the string.

Switch is a command line parameter, typically provided by the ParamStr (see page 3718) function.

Chars is the set of characters that distinguish switches from other parameters. If SwitchChars (see page 4436) is omitted, it defaults to '/' and '-' on Windows, and to '-' on Linux.

IgnoreCase determines whether case-insensitive character matching is used. The default is true on Windows, false on Linux.

See Also

ParamCount (see page 3716)

ParamStr (see page 3718)

39.3.118 SysUtils.FindFirst Function

Searches for the first instance of a file name with a given set of attributes in a specified directory.

Pascal

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
```

C++

```
int FindFirst(const UnicodeString Path, int Attr, TSearchRec F);
```

File

SysUtils

Description

FindFirst searches the directory specified by Path for the first file that matches the file name implied by Path and the attributes specified by the Attr parameter. The result is returned in the F parameter. Use the fields of this search record to extract the information needed. FindFirst returns 0 if a file was successfully located, otherwise, it returns an error code.

The Path constant parameter is the directory and file name mask, including wildcard characters. For example, '.\test*.*' specifies all files in the test subdirectory.

The Attr parameter specifies the special files to include in addition to all normal files. Choose from these file attribute constants when specifying the Attr parameter:

Constant	Description
faReadOnly (see page 4414)	Read (see page 3727)-only files
faHidden (see page 4413)	Hidden files
faSysFile (see page 4415)	System (see page 3165) files
faVolumeID (see page 4416)	Volume ID files
faDirectory (see page 4412)	Directory files
faArchive (see page 4412)	Archive files
faAnyFile (see page 4411)	Any file

Attributes can be combined by adding (Delphi) or or-ing (C++) their constants or values. For example, to search for read-only and hidden files in addition to normal files, pass (faReadOnly (see page 4414) + faHidden (see page 4413)) in Delphi or (faReadOnly (see page 4414) | faHidden (see page 4413)) in C++ as the Attr parameter. To include only normal files, pass zero for the Attr parameter.

Note: FindFirst allocates resources (memory) which must be released by calling FindClose (see page 4216).

Note: Some of the file attribute constants are not valid on all platforms. For example, faVolumeID (see page 4416) and faArchive (see page 4412) will not work on Linux.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
```

```
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
```

```
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
      FindClose(sr);
    end;
  end;
end;
```

See Also

FindClose (see page 4216)

FindNext (see page 4221)

39.3.119 SysUtils.FindNext Function

Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4218).

Pascal

```
function FindNext(var F: TSearchRec): Integer;
```

C++

```
int FindNext(TSearchRec F);
```

File

SysUtils

Description

FindNext returns the next entry that matches the name and attributes specified in a previous call to FindFirst (see page 4218). The search record must be one that was passed to FindFirst (see page 4218). The return value is zero if the function was successful. Otherwise the return value is an error code.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
```

```
    iAttributes |= faArchive * CheckBox6->Checked;
    iAttributes |= faAnyFile * CheckBox7->Checked;
    StringGrid1->RowCount = 0;
    if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
    {
      do
      {
        if ((sr.Attr & iAttributes) == sr.Attr)
        {
          StringGrid1->RowCount = StringGrid1->RowCount + 1;
          StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
          StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
        }
      } while (FindNext(sr) == 0);
      FindClose(sr);
    }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
```

```
      FindClose(sr);
    end;
  end;
end;
```

See Also

FindFirst (see page 4218)

39.3.120 SysUtils.FloatToCurr Function

Converts a floating-point value to a Currency (see page 3242) value.

Pascal

```
function FloatToCurr(const Value: Extended): Currency;
```

C++

```
Currency FloatToCurr(const Extended Value);
```

File

SysUtils

Description

FloatToCurr returns the Currency (see page 3242) value equivalent to Value. If Value is not within the valid range for a Currency (see page 3242) value, FloatToCurr raises an EConvertError (see page 3985) exception.

See Also

TryFloatToCurr (see page 4345)

Double

39.3.121 SysUtils.FloatToDateTime Function

Converts a floating-point value to a TDateTime (see page 3335) value.

Pascal

```
function FloatToDateTime(const Value: Extended): TDateTime;
```

C++

```
TDateTime FloatToDateTime(const Extended Value);
```

File

SysUtils

Description

FloatToDateTime returns the TDateTime (see page 3335) value equivalent to Value. If Value is not within the valid range for TDateTime (see page 3335), FloatToDateTime raises an EConvertError (see page 3985) exception.

See Also

TryFloatToDateTime (see page 4345)

MinDateTime (see page 4423)

EConvertError (see page 3985)

39.3.122 SysUtils.FloatToDecimal Function

Converts a floating-point value to a decimal representation.

Pascal

```
procedure FloatToDecimal(var Result: TFloatRec; const Value; ValueType: TFloatValue;
Precision: Integer; Decimals: Integer);
```

C++

```
FloatToDecimal(TFloatRec Result, const  Value, TFloatValue ValueType, int Precision, int
Decimals);
```

File

SysUtils

Description

FloatToDecimal converts a floating-point value to a decimal representation that is suited for further formatting.

The Value parameter must be a variable of type Extended or Currency (see page 3242), as indicated by the ValueType parameter.

For values of type Extended, the Precision parameter specifies the requested number of significant digits in the result--the allowed range is 1..18. For values of type Currency (see page 3242), the Precision parameter is ignored, and the implied precision of the conversion is 19 digits.

The Decimals parameter specifies the requested maximum number of digits to the left of the decimal point in the result. Precision and Decimals together control how the result is rounded. To produce a result that always has a given number of significant digits regardless of the magnitude of the number, specify 9999 for the Decimals parameter.

39.3.123 SysUtils.FloatToStr Function

Converts a floating point value to a string.

Pascal

```
function FloatToStr(Value: Extended): string; overload;
function FloatToStr(Value: Extended; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString FloatToStr(Extended Value);
UnicodeString FloatToStr(Extended Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStr converts the floating-point value given by Value to its string representation. The conversion uses general number format with 15 significant digits.

For greater control over the formatting of the string, use the FloatToStrF (see page 4226) function.

The first form of FloatToStr is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Extended ex = 16 * ArcTan(1.0/5.0) - 4.0 * ArcTan(1.0/239.0); // Machin's formula
  String mystring = FloatToStr(ex);
  Edit2->Text = mystring;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Text = "3.14159265358979";
}
```

Delphi Examples:

```
{
This example displays the compiler version number in the
text field.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text := FloatToStr(System.CompilerVersion);
end;
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

See Also

FloatToStrF (see page 4226)

FloatToText (see page 4227)

GetLocaleFormatSettings (see page 4250)

39.3.124 SysUtils.FloatToStrF Function

Converts a floating point value to a string, using a specified Format (see page 4232), Precision, and Digits.

Pascal

```
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer): string; overload;
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits);
UnicodeString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits,
const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStrF converts the floating-point value given by Value to its string representation.

The Value parameter is the value to convert.

The Precision parameter specifies the precision of the given value. It should be 7 or less for values of type Single, 15 or less for values of type Double, and 18 or less for values of type Extended.

The Digits and Format (see page 4232) parameters together control how the value is formatted into a string. For details, see the description of TFloatFormat (see page 4364).

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

The first form of FloatToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
```

```
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

39.3.125 SysUtils.FloatToText Function

Converts a floating-point value to an unterminated character string, using a specified Format (see page 4232), Precision and Digits.

Pascal

```
function FloatToText(BufferArg: PAnsiChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer): Integer; overload;
function FloatToText(BufferArg: PWideChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer): Integer; overload;
function FloatToText(BufferArg: PAnsiChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer; const FormatSettings: TFormatSettings):
Integer; overload;
function FloatToText(BufferArg: PWideChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer; const FormatSettings: TFormatSettings):
Integer; overload;
```

C++

```
int FloatToText(PAnsiChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits);
int FloatToText(PWideChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits);
int FloatToText(PAnsiChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits, const TFormatSettings FormatSettings);
int FloatToText(PWideChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToText converts the given floating-point value to its decimal representation using the specified format, precision, and digits. The Value parameter must be a variable of type Extended or Currency (see page 3242), as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

The first form of FloatToText is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToText, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToText, you must populate FormatSettings with localization information. To populate

FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

FloatToStrF (see page 4226)

GetLocaleFormatSettings (see page 4250)

39.3.126 SysUtils.FloatToTextFmt Function

Converts a floating-point value to an unterminated character string, using a specified format.

Pascal

```
function FloatToTextFmt(Buf: PAnsiChar; const Value; ValueType: TFloatValue; Format:
PAnsiChar): Integer; overload;
function FloatToTextFmt(Buf: PAnsiChar; const Value; ValueType: TFloatValue; Format:
PAnsiChar; const FormatSettings: TFormatSettings): Integer; overload;
function FloatToTextFmt(Buf: PWideChar; const Value; ValueType: TFloatValue; Format:
PWideChar): Integer; overload;
function FloatToTextFmt(Buf: PWideChar; const Value; ValueType: TFloatValue; Format:
PWideChar; const FormatSettings: TFormatSettings): Integer; overload;
```

C++

```
int FloatToTextFmt(PAnsiChar Buf, const  Value, TFloatValue ValueType, PAnsiChar Format);
int FloatToTextFmt(PAnsiChar Buf, const  Value, TFloatValue ValueType, PAnsiChar Format, const
TFormatSettings FormatSettings);
int FloatToTextFmt(PWideChar Buf, const  Value, TFloatValue ValueType, PWideChar Format);
int FloatToTextFmt(PWideChar Buf, const  Value, TFloatValue ValueType, PWideChar Format, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use FloatToTextFmt to format a floating-point value using a format string rather than using a predefined format. To use a predefined format, use the FloatToText (see page 4227) or FloatToStrF (see page 4226) function instead.

The Value parameter must be a variable of type Extended or Currency (see page 3242), as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

Formats supported are described as part of the FormatFloat (see page 4241) function.

The first form of FloatToTextFmt is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToTextFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToTextFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

FormatFloat (see page 4241)

FloatToStrF (see page 4226)

FloatToText (see page 4227)

GetLocaleFormatSettings (see page 4250)

39.3.127 SysUtils.FmtLoadStr Function

Returns formatted output using a resourced format string.

Pascal

```
function FmtLoadStr(Ident: Integer; const Args: array of const): string;
```

C++

```
UnicodeString FmtLoadStr(int Ident, const array of const Args);
```

File

SysUtils

Description

FmtLoadStr loads a string from a program's resource string table and uses that string, plus an array of arguments, as a parameter to Format (see page 4232).

Ident is the string resource ID of the desired format string.

Args is an array of arguments that replace the format specifiers in the string Ident designates.

Note: Args_Size is the index of the last element in the Args array (one less than the number of arguments).

FmtLoadStr returns the resourced string with the arguments applied. This is the same value that would result from calling Format (see page 4232), using the resourced string and the specified arguments.

See Also

Format (see page 4232)

39.3.128 SysUtils.FmtStr Function

Assembles a formatted string using a format string and an array of arguments.

Pascal

```
procedure FmtStr(var Result: string; const Format: string; const Args: array of const);
overload;
procedure FmtStr(var Result: string; const Format: string; const Args: array of const; const
FormatSettings: TFormatSettings); overload;
```

C++

```
FmtStr(UnicodeString Result, const UnicodeString Format, const array of const Args);
FmtStr(UnicodeString Result, const UnicodeString Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open (untyped) array.

StrResult returns the results of the operation.

Format (see page 4232) is the format string that determines how to generate StrResult. For information on the format strings,

see Format (see page 4232) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 4232).

Note: Args_Size specifies the index of the last element of Args (one less than the number of elements).

The first form of FmtStr is not thread-safe, because it uses localization information contained in global variables. The second form of FmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

FormatBuf (see page 4237)

StrFmt (see page 4294)

StrLFmt (see page 4304)

Format (see page 4232)

FmtLoadStr (see page 4229)

GetLocaleFormatSettings (see page 4250)

39.3.129 SysUtils.ForceDirectories Function

Creates a new directory, including the creation of parent directories as needed.

Pascal

```
function ForceDirectories(Dir: string): Boolean;
```

C++

```
Boolean ForceDirectories(UnicodeString Dir);
```

File

SysUtils

Description

ForceDirectories creates a new directory as specified in Dir, which must be a fully-qualified path name. If the directories given in the path do not yet exist, ForceDirectories attempts to create them.

ForceDirectories returns true if it successfully creates all necessary directories, false if it could not create a needed directory.

Warning: Important:Do not call ForceDirectories with an empty string. Doing so causes ForceDirectories to raise an exception.

Note: The FileCtrl unit (Windows only) also contains a ForceDirectories function. However, the FileCtrl version is deprecated, and the SysUtils (see page 3967) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following sample code sets the directory of
DirectoryListBox1 to c:\\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
c:\\WINDOWS\\mytemp is added, but note that it is not updated
```

```
in DirectoryListBox1 until Button2 is pressed and Update is
called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current directory to see the magic happen.  Also,
capitalization matters in C++ and the file spearators must
be "\\".
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ForceDirectories(Edit1->Text);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  DirectoryListBox1->Update();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  DirectoryListBox1->Directory = "c:\\WINDOWS";
}
/*
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString Dir = "C:\\Apps\\Sales\\Local";
  if (ForceDirectories(Dir))
    Label1->Caption = Dir + " was created";
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  Dir := 'C:\APPS\SALES\LOCAL';
  if SysUtils.ForceDirectories(Dir) then
    Label1.Caption := Dir + ' was created'
end;
```

See Also

ExtractFileDir (see page 4176)

GetCurrentDir (see page 4246)

RemoveDir (see page 4276)

SetCurrentDir (see page 4283)

CreateDir (see page 4147)

DirectoryExists (see page 4164)

SelectDirectory

39.3.130 SysUtils.Format Function

Returns a formatted string assembled from a format string and an array of arguments.

Pascal

```
function Format(const Format: string; const Args: array of const): string; overload;
function Format(const Format: string; const Args: array of const; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString Format(const UnicodeString Format, const array of const Args);
UnicodeString Format(const UnicodeString Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

The Format function formats the series of arguments in an open (untyped) array.

Format is the format string. For information on the format strings, see Format Strings, described in this topic.

Args is an array of arguments to apply to the format specifiers in Format. For C++, Args_Size specifies the index of the last element of Args (one less than the number of elements).

Format returns the results of applying the arguments in Args to the format string Format.

Two syntax forms are given for Format. The first form of Format is not thread-safe, because it uses localization information contained in global variables. The second form of Format, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of Format, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

Format Strings

Format strings specify required formats to general-purpose formatting routines. Format strings passed to the string formatting routines contain two types of objects -- literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form:

```
"%" [index ":"] ["-"] [width] ["." prec] type
```

A format specifier begins with a % character. After the % come the following elements, in this order:

An optional argument zero-offset index specifier (that is, the first item has index 0), `[index ":"]`

An optional left justification indicator, `["-"]`

An optional width specifier, `[width]`

An optional precision specifier, `["." prec]`

The conversion type character, `type`

The following table summarizes the possible values for type:

Value	Meaning
d	Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.
u	Unsigned decimal. Similar to 'd' but no sign is output.
e	Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f	Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.
g	General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.
n	Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.
m	Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 4389), CurrencyFormat (see page 4388), NegCurrFormat (see page 4396), ThousandSeparator (see page 4402), DecimalSeparator (see page 4390), and CurrencyDecimals (see page 4388) global variables or their equivalent in a TFormatSettings (see page 4367) data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals (see page 4388) global variable or its TFormatSettings (see page 4367) equivalent.
p	Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.
s	String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.
x	Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.

For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator (see page 4390) and ThousandSeparator (see page 4402) global variables or their TFormatSettings (see page 4367) equivalent.

Index, width, and precision specifiers can be specified directly using decimal digit string (for example "%10d"), or indirectly using an asterisk character (for example "%*.*f"). When using an asterisk, the next argument in the argument list (which must be an integer value) becomes the value that is actually used. For example,

```
Format ('%*.*f', [8, 2, 123.456]);
```

is equivalent to

```
Format ('%8.2f', [123.456]);
```

Similarly:

```
TVarRec args[3] = {8,2,123.456};
Format ("%*.*f", args, 2);
```

is equivalent to

```
TVarRec args[1] = {123.456};
Format ("%8.2f", args, 0);
```

A width specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the format specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

An index specifier sets the current argument list index to the specified value. The index of the first argument in the argument list is 0. Using index specifiers, it is possible to format the same argument multiple times. For example "Format('%d %d %0:d %1:d', [10, 20])" produces the string '10 20 10 20'.

Note: Setting the index specifier affects all subsequent formatting. That is, Format('%d %d %d %0:d %d', [1, 2, 3, 4]) returns '1 2 3 1 2', not '1 2 3 1 4'. To get the latter result, you must use Format('%d %d %d %0:d %3:d', [1, 2, 3, 4])

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
```

```
    Msg: string;
    NewFile: TFileStream;
    OldFile: TFileStream;
  begin
    temp1 := Edit4.Text;
    temp2 := ExtractFileName(Edit1.Text);
    NewFileName := temp1 + temp2;
    OldFileName :=
      Edit3.Text + ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
  {
  This example displays a message on the form's status bar
  indicating the record count after a record is deleted.  Set
  the StatusBar SimplePanel property to True.
  }
  procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
  begin
    StatusBar1.SimpleText := Format('There are now %d records in the table',
  [DataSet.RecordCount]);
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Table1.Delete;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Name := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
```

```
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

Date (see page 4151)

39.3.131 SysUtils.FormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const): Cardinal; overload;
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
function FormatBuf(Buffer: PWideChar; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
```

```
Args: array of const): Cardinal; overload;
function FormatBuf(var Buffer: UnicodeString; BufLen: Cardinal; const Format; FmtLen:
Cardinal; const Args: array of const): Cardinal; overload;
function FormatBuf(Buffer: PWideChar; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
function FormatBuf(var Buffer: UnicodeString; BufLen: Cardinal; const Format; FmtLen:
Cardinal; const Args: array of const; const FormatSettings: TFormatSettings): Cardinal;
overload;
```

C++

```
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
const Args);
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
const Args, const TFormatSettings FormatSettings);
unsigned FormatBuf(PWideChar Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const
array of const Args);
unsigned FormatBuf(UnicodeString Buffer, unsigned BufLen, const  Format, unsigned FmtLen,
const array of const Args);
unsigned FormatBuf(PWideChar Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const
array of const Args, const TFormatSettings FormatSettings);
unsigned FormatBuf(UnicodeString Buffer, unsigned BufLen, const  Format, unsigned FmtLen,
const array of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open array into a text buffer.

Buffer is the text buffer that receives the results.

BufLen is the number of bytes in Buffer.

Format (see page 4232) is the format string that determines how arguments are formatted. For information on the format strings, see Format (see page 4232) Strings.

FmtLen is the number of bytes in Format (see page 4232).

Args are the arguments that replace the format specifiers in Format (see page 4232).

Note: Args_Size is the index of the last element in Args (one less than the number of arguments).

FormatBuf function returns the length, in bytes, of the resulting string.

The first form of FormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of FormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

GetLocaleFormatSettings (see page 4250)

39.3.132 SysUtils.FormatCurr Function

Formats a Currency (see page 3242) object.

Pascal

```
function FormatCurr(const Format: string; Value: Currency): string; overload;
function FormatCurr(const Format: string; Value: Currency; const FormatSettings:
```

```
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatCurr(const UnicodeString Format, Currency Value);
UnicodeString FormatCurr(const UnicodeString Format, Currency Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatCurr formats the Currency (see page 3242) object given by Value using the format string given by Format (see page 4232).

For information on the format strings, see Format (see page 4232) Strings.

The first form of FormatCurr is not thread-safe, because it uses localization information contained in global variables. The second form of FormatCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

CurrToStr (see page 4150)

CurrToStrF (see page 4151)

StrToCurr (see page 4319)

FormatFloat (see page 4241)

GetLocaleFormatSettings (see page 4250)

39.3.133 SysUtils.FormatDateTime Function

Formats a TDateTime (see page 3335) value.

Pascal

```
function FormatDateTime(const Format: string; DateTime: TDateTime): string; overload;
function FormatDateTime(const Format: string; DateTime: TDateTime; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatDateTime(const UnicodeString Format, TDateTime DateTime);
UnicodeString FormatDateTime(const UnicodeString Format, TDateTime DateTime, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

FormatDateTime formats the TDateTime (see page 3335) value given by DateTime using the format given by Format (see page 4232). See the table below for information about the supported format strings.

If the string specified by the Format (see page 4232) parameter is empty, the TDateTime (see page 3335) value is formatted as if a 'c' format specifier had been given.

The first form of FormatDateTime is not thread-safe, because it uses localization information contained in global variables. The

second form of FormatDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

Date (see page 4151) Time (see page 4338) Format (see page 4232) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 4398) global variable, followed by the time using the format given by the LongTimeFormat (see page 4396) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 4399) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 4395) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 4398) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 4394) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3652)) using the strings given by the ShortMonthNames (see page 4399) global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 4395) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).

t	Displays the time using the format given by the ShortTimeFormat (see page 4400) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 4396) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 4402) global variable for any hour before noon, and the contents of the TimePMString (see page 4402) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 4389) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 4403) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

C++ Examples:

```
/*
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 1995 at 2:30 PM."
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString S = "The meeting is on ";
  S = S + FormatDateTime("dddd, mmmm d, yyyy ' at ' hh:mm AM/PM", Now() + 0.125);
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 1995 at 2:30 PM."
}
procedure TForm1.Button1Click(Sender: TObject);
var S : string;
begin
  S := SysUtils.FormatDateTime(
    '"The meeting is on " dddd, mmmm d, yyyy, " at " hh:mm AM/PM', Now + 0.125);
  Label1.Caption := S;
end;
```

39.3.134 SysUtils.FormatFloat Function

Formats a floating point value.

Pascal

```
function FormatFloat(const Format: string; Value: Extended): string; overload;
function FormatFloat(const Format: string; Value: Extended; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatFloat(const UnicodeString Format, Extended Value);
```

```
UnicodeString FormatFloat(const UnicodeString Format, Extended Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatFloat formats the floating-point value given by Value using the format string given by Format (see page 4232). The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator (see page 4390) global variable or its TFormatSettings (see page 4367) equivalent.
,	Thousand separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator (see page 4402) global variable or its TFormatSettings (see page 4367) equivalent.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 4226) with the ffGeneral format. General

floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The first form of FormatFloat is not thread-safe, because it uses localization information contained in global variables. The second form of FormatFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
/*
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  ShowMessage(FormatFloat("#,##0 bytes", DiskFree(0)));
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;

end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

FloatToStrF (see page 4226)

FloatToText (see page 4227)

FloatToTextFmt (see page 4228)

GetLocaleFormatSettings (see page 4250)

39.3.135 SysUtils.FreeAndNil Function

Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNil(var Obj);
```

C++

```
FreeAndNil( Obj);
```

File

SysUtils

Description

Use FreeAndNil to ensure that a variable is nil (Delphi) or NULL (C++) after you free the object it references. Pass any variable that represents an object as the Obj parameter.

Warning: Obj must be an instance of a TObject (see page 3364) descendant.

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
```

```
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Picture.LoadFromFile('factory.bmp');
  end;
```

See Also

TObject

39.3.136 SysUtils.GDAL Function

Get Delphi Access Licence.

Pascal

```
function GDAL: LongWord;
```

C++

```
LongWord GDAL();
```

File

SysUtils

Description

GDAL retreives the access licences resource. It checks if it is valid, if not an Access Licence Violation exception with the message 'Application is not licenced to use this feature' is raised.

The returned value is the decrypted first Access Licence (AL1).

See Also

RCS (see page 4276)

RPR (see page 4281)

39.3.137 SysUtils.GetCurrentDir Function

Returns the name of the current directory.

Pascal

```
function GetCurrentDir: string;
```

C++

```
UnicodeString GetCurrentDir();
```

File

SysUtils

Description

GetCurrentDir returns the fully qualified name of the current directory.

C++ Examples:

```
/*
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
*/
void __fastcall TMyChDir::Button1Click(TObject *Sender)
{
  try
  {
    System::ChDir(Edit1->Text);
  }
  catch (...)
  {
    MessageDlg("Cannot find directory", mtWarning, TMsgDlgButtons() << mbOK, 0);
  }
  Edit2->Text = GetCurrentDir();
  FileListBox1->Directory = GetCurrentDir();
  DirectoryListBox1->Directory = GetCurrentDir();
}

void __fastcall TMyChDir::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}

void __fastcall TMyChDir::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}
```

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;
```

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;
```

See Also

GetDir (see page 3679)

CreateDir (see page 4147)

ExtractFileDir (see page 4176)

RemoveDir (see page 4276)

SetCurrentDir (see page 4283)

39.3.138 SysUtils.GetEnvironmentVariable Function

Retrieves an environment variable value.

Pascal

```
function GetEnvironmentVariable(const Name: string): string; overload;
```

C++

```
UnicodeString GetEnvironmentVariable(const UnicodeString Name);
```

File

SysUtils

Description

Call GetEnvironmentVariable to retrieve the value of an environment variable, passed as `Name`, for the current process.

Note: The case of the specified environment variable `Name` is ignored.

The following table defines the standard environment variables for Microsoft Windows.

Environment variable name	Environment value
ALLUSERSPROFILE	Generic user profile.
APPDATA	Path of the application data folder.
CLIENTNAME	Name of Client machine.
COMMONPROGRAMFILES	Path of common program files folder.
COMPUTERNAME	Name of Computer code is running on.
COMSPEC	Path of the cmd.exe program.
HOMEDRIVE	Current home drive designation, such as 'C:'
HOMEPATH	Path to current location for document storage.

LOGONSERVER	
NUMBER_OF_PROCESSORS	Number of processors on current machine.
OS	Base name of the Operating System (see page 3165). Note that Windows XP is given as Windows_NT.
PATH	The current program path.
PATHEXT	
PROCESSOR_ARCHITECTURE	Type of CPU architecture. For example, X86 for Intel Pentium processors.
PROCESSOR_IDENTIFIER	
PROCESSOR_LEVEL	More detailed description of the CPU architecture.
PROCESSOR_REVISION	Processor revision level.
PROGRAMFILES	Path of the program files folder.
SESSIONNAME	Name of the current OS session.
SYSTEMDRIVE	Drive the OS operates from.
SYSTEMROOT	
TEMP	Path of the temporary files folder.
TMP	
USERDOMAIN	
USERNAME	Name of the current user.
USERPROFILE	Path of the folder holding the current user's information.
WINDIR	Path of the Windows folder.

39.3.139 SysUtils.GetFileVersion Function

Returns file version.

Pascal

```
function GetFileVersion(const AFileName: string): Cardinal;
```

C++

```
unsigned GetFileVersion(const UnicodeString AFileName);
```

File

SysUtils

Description

GetFileVersion returns the most significant 32 bits of a file binary version number. AFileName specifies the name of the file, which is searched using the same path as the LoadLibrary function.

GetFileVersion is only available on Windows and must not be used in cross-platform code.

39.3.140 SysUtils.GetFormatSettings Function

Resets the date and number format parameters to initial values.

Pascal

```
procedure GetFormatSettings;
```

C++

```
GetFormatSettings();
```

File

SysUtils

Description

GetFormatSettings resets all date and number format variables to their initial values.

Note: On Windows, GetFormatSettings is called automatically in response to WM_WININICHANGE messages.

39.3.141 SysUtils.GetLocaleChar Function

Retrieves the first character of a localisation value.

Pascal

```
function GetLocaleChar(Locale: Integer; LocaleType: Integer; Default: Char): Char; platform;
```

C++

```
Char GetLocaleChar(int Locale, int LocaleType, Char Default);
```

File

SysUtils

Description

GetLocaleChar retrieves the first character of the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $0809 is the Locale for United Kingdom English, and $0409 is the Locale for US English. For the LOCALE_SCURRENCY LocaleType, these locales would yield '£' and '$' respectively.

Note: This is an OS specific function.

See Also

GetLocaleStr (see page 4251)

39.3.142 SysUtils.GetLocaleFormatSettings Function

Populates a TFormatSettings (see page 4367) data structure.

Pascal

```
procedure GetLocaleFormatSettings(LCID: Integer; var FormatSettings: TFormatSettings);
```

C++

```
GetLocaleFormatSettings(int LCID, TFormatSettings FormatSettings);
```

File

SysUtils

Description

GetLocaleFormatSettings populates FormatSettings based on the Windows locale ID specified in LCID. A TFormatSettings (see page 4367) data structure must be initialized by a call to GetLocaleFormatSettings before it can be used with any string formatting routines.

39.3.143 SysUtils.GetLocaleStr Function

Retrieves a localisation value.

Pascal

```
function GetLocaleStr(Locale: Integer; LocaleType: Integer; const Default: string): string;
platform;
```

C++

```
UnicodeString GetLocaleStr(int Locale, int LocaleType, const UnicodeString Default);
```

File

SysUtils

Description

GetLocaleStr retrieves the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $040c is the Locale for France French, and $0409 is the Locale for US English. For the LOCALE_SMONTHNAME1 LocaleType, these locales would yield 'janvier' and 'January' respectively.

Note: This is an OS specific function.

See Also

GetLocaleChar (see page 4250)

39.3.144 SysUtils.GetModuleName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleName(Module: HMODULE): string;
```

C++

```
UnicodeString GetModuleName(HMODULE Module);
```

File

SysUtils

Description

Call GetModuleName to obtain the fully qualified name of a module, given its handle.

Module is the handle of the module whose file name is required.

Warning: :On Linux, GetModuleName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

C++ Examples:

```
/*
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
*/
AnsiString __fastcall GetUsersFileName(void)
{
  return GetModuleName(unsigned(HInstance));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = GetUsersFileName();
}
```

Delphi Examples:

```
{
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
}
function GetUsersFileName: string;
begin
  Result := SysUtils.GetModuleName(HInstance);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := GetUsersFileName;
end;
```

39.3.145 SysUtils.GetPackageDescription Function

Returns the description resource from a package.

Pascal

```
function GetPackageDescription(ModuleName: PChar): string;
```

C++

```
UnicodeString GetPackageDescription(const char * ModuleName);
```

File

SysUtils

Description

Call GetPackageDescription to obtain the description stored with the named package. If the package does not have a description

resource, GetPackageDescription returns an empty string.

See Also

GetPackageInfo (see page 4253)

39.3.146 SysUtils.GetPackageInfo Function

Enumerates all the units and required packages for a package.

Pascal

```
procedure GetPackageInfo(Module: HMODULE; Param: Pointer; var Flags: Integer; InfoProc:
TPackageInfoProc);
```

C++

```
GetPackageInfo(HMODULE Module, void * Param, int Flags, TPackageInfoProc InfoProc);
```

File

SysUtils

Description

Call GetPackageInfo to process the information in a package's information table. The InfoProc callback is called for every unit included in the package and for every package required by the package. Module is the module handle for the package. GetPackageInfo passes any application-supplied information passed in the Param parameter to the InfoProc callback. The package's information flags are passed to the InfoProc and returned in the Flags parameter.

See Also

LoadPackage (see page 4268)

GetPackageDescription (see page 4252)

39.3.147 SysUtils.GetTime Function

Returns the current time.

Pascal

```
function GetTime: TDateTime;
```

C++

```
TDateTime GetTime();
```

File

SysUtils

Description

GetTime and Time (see page 4338) return the current time as a TDateTime (see page 3335) value. The two functions are completely equivalent.

Note: Time (see page 4338) clashes with an Xlib function of the same name. Use GetTime in Linux and cross-platform code.

See Also

Time (see page 4338)

Date (see page 4151)

DecodeTime (see page 4161)

Now (see page 4271)

TimeToStr (see page 4340)

39.3.148 SysUtils.GUIDToString Function

Converts a class identifier GUID to a string.

Pascal

```
function GUIDToString(const GUID: TGUID): string;
```

C++

```
UnicodeString GUIDToString(const TGUID GUID);
```

File

SysUtils

Description

GUIDToString converts a GUID into a string of printable characters. Each GUID always converts to a unique string.

If GUIDToString succeeds it returns a string for the requested GUID. If GUIDToString fails, it raises EConvertError (see page 3985).

C++ Examples:

```
/*
The following example manufactures a GUID and displays it in
a dialog box after translating it to a string.
*/
#include <ActiveX.hpp>
#include <ComObj.hpp>
#include <Sysutils.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    TGUID g;
    OleCheck(CoCreateGuid(&g));
    ShowMessage(Sysutils::GUIDToString(g));
}
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
```

```
            "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
          ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
          myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
          myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
The following example manufactures a GUID and displays it in
a dialog box after translating it to a string.
}
uses ActiveX, ComObj;
procedure TForm1.Button1Click(Sender: TObject);
var
  G: TGUID;
begin
  OleCheck(CoCreateGuid(G));
  ShowMessage(SysUtils.GuidToString(G));
end;
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
```

```
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
  end;
```

See Also

StringToGUID (see page 4297)

EConvertError (see page 3985)

39.3.149 SysUtils.HashName Function

Pascal

```
function HashName(Name: PAnsiChar): Cardinal;
```

C++

```
unsigned HashName(PAnsiChar Name);
```

File

SysUtils

Description

This is function SysUtils.HashName.

39.3.150 SysUtils.IncAMonth Function

Increments date data by one month.

Pascal

```
procedure IncAMonth(var Year: Word; var Month: Word; var Day: Word; NumberOfMonths: Integer =
1);
```

C++

```
IncAMonth(Word Year, Word Month, Word Day, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncAMonth modifies Year, Month, and Day parameters so the date they describe is incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month.

See Also

IncMonth (see page 4258)

DecodeDate (see page 4159)

EncodeDate (see page 4169)

ReplaceDate (see page 4279)

IncDay

IncHour

IncMilliSecond

IncMinute

IncSecond

IncWeek

IncYear

39.3.151 SysUtils.IncludeTrailingBackslash Function

Ensures path name ends with delimiter

Pascal

```
function IncludeTrailingBackslash(const S: string): string; overload; platform;
```

C++

```
UnicodeString IncludeTrailingBackslash(const UnicodeString S);
```

File

SysUtils

Description

IncludeTrailingBackslash ensures that a path name ends with a trailing path delimiter ('\' on Windows, '/' on Linux). This function is included for backward-compatibility only. New (see page 3712) programs should use IncludeTrailingPathDelimiter (see page 4257).

Note: IncludeTrailingBackslash works with multibyte character sets.

See Also

ExcludeTrailingBackslash (see page 4172)

IsPathDelimiter (see page 4266)

39.3.152 SysUtils.IncludeTrailingPathDelimiter Function

Ensures path name ends with delimiter.

Pascal

```
function IncludeTrailingPathDelimiter(const S: string): string; overload;
```

C++

```
UnicodeString IncludeTrailingPathDelimiter(const UnicodeString S);
```

File

SysUtils

Description

IncludeTrailingPathDelimiter ensures that a path name ends with a trailing path delimiter ('\' on Windows, '/' on Linux). If S already ends with a trailing delimiter character, it is returned unchanged; otherwise S with appended delimiter character is returned.

Note: IncludeTrailingPathDelimiter works with multibyte character sets.

See Also

ExcludeTrailingPathDelimiter (see page 4173)

IsPathDelimiter (see page 4266)

39.3.153 SysUtils.IncMonth Function

Returns a date shifted by a specified number of months.

Pascal

```
function IncMonth(const DateTime: TDateTime; NumberOfMonths: Integer = 1): TDateTime;
```

C++

```
TDateTime IncMonth(const TDateTime DateTime, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncMonth returns the value of the Date (see page 4151) parameter, incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month. The time of day specified by the Date (see page 4151) parameter is copied to the result.

See Also

IncAMonth (see page 4256)

DecodeDate (see page 4159)

EncodeDate (see page 4169)

ReplaceDate (see page 4279)

IncDay

IncHour

IncMilliSecond

IncMinute

IncSecond

IncWeek

IncYear

39.3.154 SysUtils.InitializePackage Function

Validates and initializes a given package.

Pascal

```
procedure InitializePackage(Module: HMODULE); overload;
procedure InitializePackage(Module: HMODULE; AValidatePackage: TValidatePackageProc); overload;
```

C++

```
InitializePackage(HMODULE Module);
InitializePackage(HMODULE Module, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

InitializePackage checks for duplicate units in a package and calls the initialization blocks of the contained units.

Note: InitializePackage is called internally by the LoadPackage (see page 4268) function.

See Also

LoadPackage (see page 4268)

39.3.155 SysUtils.IntToHex Function

Returns the hex representation of an integer.

Pascal

```
function IntToHex(Value: Integer; Digits: Integer): string; overload;
function IntToHex(Value: Int64; Digits: Integer): string; overload;
```

C++

```
UnicodeString IntToHex(int Value, int Digits);
UnicodeString IntToHex(Int64 Value, int Digits);
```

File

SysUtils

Description

IntToHex converts a number into a string containing the number's hexadecimal (base 16) representation. Value is the number to convert. Digits indicates the minimum number of hexadecimal digits to return.

C++ Examples:

```
/*
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = "";
```

```
    for (int i = 1; i <= Edit1->Text.Length(); i++)
    {
      try
      {
        Label1->Caption =
          Label1->Caption +
          IntToHex(Byte(Edit1->Text[i]),2) + ' ';
      }
      catch (...)
      {
        Beep();
      }
    }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
    try
      Label1.Caption :=
        Label1.Caption +
        SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
    except
      Beep;
    end;
  end;
end;
```

See Also

IntToStr (see page 4260)

39.3.156 SysUtils.IntToStr Function

Converts an integer to a string.

Pascal

```
function IntToStr(Value: Integer): string; overload;
function IntToStr(Value: Int64): string; overload;
```

C++

```
UnicodeString IntToStr(int Value);
UnicodeString IntToStr(Int64 Value);
```

File

SysUtils

Description

IntToStr converts an integer into a string containing the decimal representation of that number.

C++ Examples:

```
/*
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  try
  {
    Label1->Caption = IntToStr(StrToInt(Edit1->Text) * StrToInt(Edit2->Text));
  }
  catch(...)
  {
    ShowMessage("You must specify integer values. Please try again.");
  }
}
```

Delphi Examples:

```
{
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    Label1.Caption :=
      SysUtils.IntToStr(StrToInt(Edit1.Text) * StrToInt(Edit2.Text));
  except
    ShowMessage('You must specify integer values. Please try again.');
  end;
end;
```

See Also

IntToHex (see page 4259)

StrToInt (see page 4328)

StrToIntDef (see page 4330)

FloatToStr (see page 4224)

DateToStr (see page 4156)

TimeToStr (see page 4340)

39.3.157 SysUtils.IsAssembly Function

Determines whether a dll is an Assembly.

Pascal

```
function IsAssembly(const FileName: string): Boolean;
```

C++

```
Boolean IsAssembly(const UnicodeString FileName);
```

File

SysUtils

Description

Call IsAssembly to determine whether the passed FileName is that of a .Net Assembly or a traditional WIN32 DLL. The result is true if FileName is an Assembly.

39.3.158 SysUtils.IsDelimiter Function

Indicates whether a specified character in a string matches one of a set of delimiters.

Pascal

```
function IsDelimiter(const Delimiters: string; const S: string; Index: Integer): Boolean;
overload;
```

C++

```
Boolean IsDelimiter(const UnicodeString Delimiters, const UnicodeString S, int Index);
```

File

SysUtils

Description

Call IsDelimiter to determine whether the character at byte offset Index in the string S is one of the delimiters in the string Delimiters. Index is the index of the byte in question, where 1 is the first byte of the string, 2 is the second byte, and so on.

When working with a multi-byte character system (MBCS), IsDelimiter checks to make sure the indicated byte is not part of a double byte character. The delimiters in the Delimiters parameter must all be single byte characters.

See Also

AnsiStrRScan (see page 4124)

AnsiStrScan (see page 4125)

IsPathDelimiter (see page 4266)

LastDelimiter (see page 4267)

39.3.159 SysUtils.IsEqualGUID Function

Indicates whether two TGUID (see page 3800) values are the same.

Pascal

```
function IsEqualGUID(const guid1: TGUID; const guid2: TGUID): Boolean; stdcall;
```

C++

```
__stdcall Boolean IsEqualGUID(const TGUID guid1, const TGUID guid2);
```

File

SysUtils

Description

IsEqualGUID returns true if the TGUID (see page 3800) values are equal.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7]))));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));
```

```
    { Auto-generate a random GUID at runtime }
    if CreateGUID(MyGuid1) <> 0 then
       Memo1.Lines.Add('Creating GUID failed!')
    else
       Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

    { Generating second random GUID }
    CreateGUID(MyGuid0);

    { Testing if 2 guids are equal }
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
end;
```

39.3.160 SysUtils.IsLeadChar Function

Pascal

```
function IsLeadChar(C: AnsiChar): Boolean; overload;
```

C++

```
Boolean IsLeadChar(AnsiChar C);
```

File

SysUtils

Description

This is function SysUtils.IsLeadChar.

39.3.161 SysUtils.IsLeadChar Function

Pascal

```
function IsLeadChar(C: WideChar): Boolean; overload;
```

C++

```
Boolean IsLeadChar(WideChar C);
```

File

SysUtils

Description

This is function SysUtils.IsLeadChar.

39.3.162 SysUtils.IsLeapYear Function

Indicates whether a specified year is a leap year.

Pascal

```
function IsLeapYear(Year: Word): Boolean;
```

C++

```
Boolean IsLeapYear(Word Year);
```

File

SysUtils

Description

Call IsLeapYear to determine whether the year specified by the Year parameter is a leap year. Year specifies the calendar year.

Use YearOf to obtain the value of Year for IsLeapYear from a TDateTime (see page 3335) value.

C++ Examples:

```
/*
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
*/
bool __fastcall IsThisLeapYear(int year)
{
  return IsLeapYear(year);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int year = StrToInt(Edit1->Text);
  if (IsThisLeapYear(year))
    ShowMessage(Edit1->Text + " is a leap year.");
  else
    ShowMessage(Edit1->Text + " is not a leap year.");
}
```

Delphi Examples:

```
{
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
}
function IsThisLeapYear(year : Word): Boolean;
begin
  Result := SysUtils.IsLeapYear(year);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  year : integer;
begin
  year := StrToInt(Edit1.Text);
  if (IsThisLeapYear(year)) then
    ShowMessage(Edit1.Text + ' is a leap year.')
  else
    ShowMessage(Edit1.Text + ' is not a leap year.');
end;
```

See Also

Date (see page 4151)

YearOf

IsInLeapYear

39.3.163 SysUtils.IsPathDelimiter Function

Indicates whether the byte at a specified position in a string is the path delimiter.

Pascal

```
function IsPathDelimiter(const S: string; Index: Integer): Boolean; overload;
```

C++

```
Boolean IsPathDelimiter(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call IsPathDelimiter to determine whether a position in the string S contains the path delimiter character ('/' on Linux, '\' on Windows). Positions are numbered from 1.

When working with a multi-byte character system (MBCS), IsPathDelimiter distinguishes between a true delimiter character, and the byte of the same value that can appear as the second byte of a double byte character.

To test for more characters than just the delimiter, use IsDelimiter (see page 4262).

See Also

IsDelimiter (see page 4262)

LastDelimiter (see page 4267)

39.3.164 SysUtils.IsValidIdent Function

Determines whether a string conforms to Delphi identifier rules.

Pascal

```
function IsValidIdent(const Ident: string; AllowDots: Boolean = False): Boolean;
```

C++

```
Boolean IsValidIdent(const UnicodeString Ident, Boolean AllowDots = False);
```

File

SysUtils

Description

Call IsValidIdent to determine whether the passed Ident is valid for use as an identifier, such as the name of a component.

An identifier is a sequence of letters, digits or underscores. Only the first 63 characters are used uniquely. Letters are limited to A-Z and a-z, although the case is ignored. The first character cannot be a digit.

39.3.165 SysUtils.Languages Function

Lists the locales for which support is available.

Pascal

```
function Languages: TLanguages;
```

C++

```
TLanguages Languages();
```

File

SysUtils

Description

Use the value returned by Languages to obtain information about the locales that the system supports. This function is only available on Windows.

See Also

TSysLocale (see page 4374)

39.3.166 SysUtils.LastDelimiter Function

Returns the byte index of the last character that matches any character in a specified set of delimiters.

Pascal

```
function LastDelimiter(const Delimiters: string; const S: string): Integer; overload;
```

C++

```
int LastDelimiter(const UnicodeString Delimiters, const UnicodeString S);
```

File

SysUtils

Description

Call LastDelimiter to locate the last delimiter in a specified string.

Delimiters is a string where each character is a valid delimiter.

S is the string to search for delimiters.

For example, the line

MyIndex := LastDelimiter('\.:','c:\filename.ext'); // Delphi

MyIndex = LastDelimiter("\\.:","c:\\filename.ext"); // C++

sets MyIndex to 12.

When working with multi-byte character sets (MBCS), S may contain double byte characters, but the delimiters listed in the Delimiters parameter must all be single byte non-null characters.

See Also

IsDelimiter (see page 4262)

AnsiStrRScan (see page 4124)

39.3.167 SysUtils.LoadPackage Function

Loads a specified package.

Pascal

```
function LoadPackage(const Name: string): HMODULE; overload;
function LoadPackage(const Name: string; AValidatePackage: TValidatePackageProc): HMODULE;
overload;
```

C++

```
HMODULE LoadPackage(const UnicodeString Name);
HMODULE LoadPackage(const UnicodeString Name, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

LoadPackage loads the package specified by the Name parameter, checks for duplicate units, and calls the initialization blocks of all units contained in the package.

See Also

UnloadPackage (see page 4352)

39.3.168 SysUtils.LoadStr Function

Loads a string resource from the application's executable file.

Pascal

```
function LoadStr(Ident: Integer): string;
```

C++

```
UnicodeString LoadStr(int Ident);
```

File

SysUtils

Description

LoadStr loads the string resource given by Ident from the application's executable file or shared library. If the string resource does not exist, LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

Note: :LoadStr is provided primarily for backward compatibility. To isolate string resources in Delphi, use the resourcestring keyword in new applications.

39.3.169 SysUtils.LowerCase Function

Converts an ASCII string to lowercase.

Pascal

```
function LowerCase(const S: string): string; overload;
function LowerCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
UnicodeString LowerCase(const UnicodeString S);
UnicodeString LowerCase(const UnicodeString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase (see page 4117).

C++ Examples:

```
/*
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = LowerCase(Edit1->Text);
}
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.LowerCase(Edit1.Text);
end;
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

AnsiLowerCase (see page 4117)

UpperCase (see page 4352)

39.3.170 SysUtils.MSecsToTimeStamp Function

Converts a specified number of milliseconds into a TTimeStamp (see page 4375) value.

Pascal

```
function MSecsToTimeStamp(MSecs: Comp): TTimeStamp;
```

C++

```
TTimeStamp MSecsToTimeStamp(Comp MSecs);
```

File

SysUtils

Description

Call MSecsToTimeStamp to convert MSecs, a number of milliseconds, into the number of days represented, plus the number of milliseconds remaining. The days and milliseconds are expressed as a TTimeStamp (see page 4375) value, where the Date (see page 4151) field is the number of days and the Time (see page 4338) field is the remainder in milliseconds.

See Also

TimeStampToMSecs (see page 4340)

TTimeStamp (see page 4375)

39.3.171 SysUtils.NewStr Function

Allocates a string on the heap.

Pascal

```
function NewStr(const S: AnsiString): PAnsiString; deprecated;
```

C++

```
PAnsiString NewStr(const AnsiString S);
```

File

SysUtils

Description

NewStr allocates a new string on the heap. Do not use NewStr in new applications. This function is provided for backward compatibility only.

39.3.172 SysUtils.NextCharIndex Function

Returns the byte index of the next character.

Pascal

```
function NextCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
function NextCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int NextCharIndex(const UnicodeString S, int Index);
int NextCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

NextCharIndex returns the byte index of the first byte of the character following the character starting at S[Index].

39.3.173 SysUtils.Now Function

Returns the current date and time.

Pascal

```
function Now: TDateTime;
```

C++

```
TDateTime Now();
```

File

SysUtils

Description

Returns the current date and time, corresponding to the sum of the value returned by the global Date (see page 4151) and Time (see page 4338) functions.

Note: Although TDateTime (see page 3335) values can represent milliseconds, Now is accurate only to the nearest second.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

See Also

Date (see page 4151)

DateTimeToStr (see page 4153)

Time (see page 4338)

39.3.174 SysUtils.OutOfMemoryError Function

Raises an EOutOfMemory (see page 4008) exception.

Pascal

```
procedure OutOfMemoryError;
```

C++

```
OutOfMemoryError();
```

File

SysUtils

Description

Call OutOfMemoryError to raise the EOutOfMemory (see page 4008) exception. OutOfMemoryError runs without allocating additional memory.

See Also

EOutOfMemory (see page 4008)

39.3.175 SysUtils.PlatformBytesOf Function

Pascal

```
function PlatformBytesOf(const Value: string): TBytes;
```

C++

```
TBytes PlatformBytesOf(const UnicodeString Value);
```

File

SysUtils

Description

This is function SysUtils.PlatformBytesOf.

39.3.176 SysUtils.PlatformStringOf Function

Pascal

```
function PlatformStringOf(const Value: TBytes): UnicodeString;
```

C++

```
UnicodeString PlatformStringOf(const TBytes Value);
```

File

SysUtils

Description

This is function SysUtils.PlatformStringOf.

39.3.177 SysUtils.QuotedStr Function

Returns the quoted version of a string.

Pascal

```
function QuotedStr(const S: string): string; overload;
```

C++

```
UnicodeString QuotedStr(const UnicodeString S);
```

File

SysUtils

Description

Use QuotedStr to convert the string S to a quoted string. A single quote character (') is inserted at the beginning and end of S,

and each single quote character in the string is repeated.

Note: When working with multi-byte character systems (MBCS), use AnsiQuotedStr (see page 4118) instead.

39.3.178 SysUtils.RaiseLastOSError Function

Raises an exception for the last occurring OS or system library error.

Pascal

```
procedure RaiseLastOSError; overload;
procedure RaiseLastOSError(LastError: Integer); overload;
```

C++

```
RaiseLastOSError();
RaiseLastOSError(int LastError);
```

File

SysUtils

Description

Call RaiseLastOSError to raise an EOSError (see page 4005) exception for the last OS API call that failed. RaiseLastOSError retrieves the code for the last occurring API call error, if any. If this belongs to a previously occurring error, RaiseLastOSError raises an EOSError (see page 4005) exception with the error code and message associated with that error.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);

    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
    on Ex : EOSError do
       MessageDlg('Caught an OS error with code: ' +
          IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
  end;

  { Let the Delphi's Exception dialog pop-up }
  RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

  { Finally set the last error to none }
  System.SetLastError(ERROR_SUCCESS);

  if GetLastError() <> ERROR_SUCCESS then
     MessageDlg('Whoops, something went wrong in the ' +
                'mean time!', mtError, [mbOK], 0);

  { No exception should be thrown here }
  RaiseLastOSError();
end;
```

See Also

EOSError (see page 4005)

39.3.179 SysUtils.RaiseLastWin32Error Function

Raises an exception for the last occurring Win32 error.

Pascal

```
procedure RaiseLastWin32Error; deprecated;
```

C++

```
RaiseLastWin32Error();
```

File

SysUtils

Description

Call RaiseLastWin32Error to raise an EWin32Error (see page 4015) exception for the last Windows API call that failed. RaiseLastWin32Error retrieves the code for the last occurring Win32 error, if any. If Windows returns a previously occurring error code, RaiseLastWin32Error raises an EWin32Error (see page 4015) exception with the error code and message associated with that error.

Note: RaiseLastWin32Error is a deprecated Windows-only procedure. New (see page 3712) applications should use RaiseLastOSError (see page 4274) instead.

See Also

EWin32Error (see page 4015)

Win32Check (see page 4359)

RaiseLastOSError (see page 4274)

39.3.180 SysUtils.RCS Function

Check for Delphi Client/Server Licence.

Pascal

```
procedure RCS;
```

C++

```
RCS();
```

File

SysUtils

Description

RCS is used by VCL components to check whether there is a Client/Server licence, raising an exception of not.

See Also

GDAL (see page 4246)

RPR (see page 4281)

39.3.181 SysUtils.RemoveDir Function

Deletes an existing empty directory.

Pascal

```
function RemoveDir(const Dir: string): Boolean;
```

C++

```
Boolean RemoveDir(const UnicodeString Dir);
```

File

SysUtils

Description

Call RemoveDir to remove the directory specified by the Dir parameter. The return value is true if a new directory was successfully deleted, false if an error occurred. The directory must be empty before it can be successfully deleted.

See Also

ExtractFileDir (see page 4176)

GetCurrentDir (see page 4246)

SetCurrentDir (see page 4283)

39.3.182 SysUtils.RenameFile Function

Changes a file name.

Pascal

```
function RenameFile(const OldName: string; const NewName: string): Boolean;
```

C++

```
Boolean RenameFile(const UnicodeString OldName, const UnicodeString NewName);
```

File

SysUtils

Description

RenameFile attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns true. If RenameFile cannot rename the file (for example, if the application does not have permission to modify the file), it returns false.

Note: If NewFile represents the name of an existing file, RenameFile behaves differently under Windows and Linux. Under Windows, the rename operation fails. Under Linux, RenameFile silently removes the other file.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).t_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
```

```
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
```

```
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

DeleteFile (see page 4162)

Rename (see page 3733)

39.3.183 SysUtils.ReplaceDate Function

Replaces the date portion of a TDateTime (see page 3335) value with a specified date.

Pascal

```
procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime);
```

C++

```
ReplaceDate(TDateTime DateTime, const TDateTime NewDate);
```

File

SysUtils

Description

ReplaceDate changes the value of the DateTime parameter so that it has the same date as the NewDate parameter. The time

portion of DateTime remains the same. ReplaceDate makes any necessary adjustments for negative dates (before 1900).

See Also

DecodeDate (see page 4159)

EncodeDate (see page 4169)

ReplaceTime (see page 4280)

IncMonth (see page 4258)

RecodeDateDay

RecodeDateHour

RecodeDateMillisecond

RecodeDateMinute

RecodeDateMonth

RecodeDateSecond

RecodeDateYear

RecodeDate

39.3.184 SysUtils.ReplaceTime Function

Replaces the time portion of a TDateTime (see page 3335) value with a specified time.

Pascal

```
procedure ReplaceTime(var DateTime: TDateTime; const NewTime: TDateTime);
```

C++

```
ReplaceTime(TDateTime DateTime, const TDateTime NewTime);
```

File

SysUtils

Description

ReplaceTime changes the value of the DateTime parameter so that it has the same time as the NewTime parameter. The date portion of DateTime remains the same. ReplaceTime makes any necessary adjustments for negative dates. (before 1900)

See Also

DecodeTime (see page 4161)

EncodeTime (see page 4170)

ReplaceDate (see page 4279)

RecodeDateDay

RecodeDateHour

RecodeDateMillisecond

RecodeDateMinute

RecodeDateMonth

RecodeDateSecond

RecodeDateYear

StartOfADay

EndOfADay

RecodeTime

39.3.185 SysUtils.RPR Function

Check for Delphi Professional Licence.

Pascal

```
procedure RPR;
```

C++

```
RPR();
```

File

SysUtils

Description

RPR is used by VCL components to check whether there is a Professional licence, raising an exception of not.

See Also

GDAL (see page 4246)

RCS (see page 4276)

39.3.186 SysUtils.SafeLoadLibrary Function

Safe and simple library loading.

Pascal

```
function SafeLoadLibrary(const FileName: string; ErrorMode: UINT = SEM_NOOPENFILEERRORBOX):
HMODULE;
```

C++

```
HMODULE SafeLoadLibrary(const UnicodeString FileName, UINT ErrorMode = SEM_NOOPENFILEERRORBOX);
```

File

SysUtils

Description

SafeLoadLibrary loads a Windows DLL or Linux shared object file, as specified by Filename. SafeLoadLibrary preserves the current FPU control word, preventing library initialization code from permanently overwriting precision and exception masks.

Note: On Windows, SafeLoadLibrary temporarily sets the system error mode to ErrorMode. The default, SEM_NOOPENFILEERRORBOX, suppresses error dialogs. The previous error mode is restored before SafeLoadLibrary exits. For a list of error modes, refer to SetErrorMode in the Microsoft documentation.

Note: On Linux, the Dummy argument is ignored.

See Also

LoadPackage (see page 4268)

39.3.187 SysUtils.SameFileName Function

Compares file names based on the current locale.

Pascal

```
function SameFileName(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean SameFileName(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

SameFileName compares two file names in the same way as AnsiCompareFileName (see page 4113). SameFileName returns true if the file names match, false otherwise. Like AnsiCompareFileName (see page 4113), SameFileName supports locales and MBCS strings; see the AnsiCompareFileName (see page 4113) topic for more details.

See Also

AnsiCompareText (see page 4115)

39.3.188 SysUtils.SameStr Function

Compares two strings by ordinal value with case sensitivity.

Pascal

```
function SameStr(const S1: string; const S2: string): Boolean; overload;
function SameStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameStr(const UnicodeString S1, const UnicodeString S2);
Boolean SameStr(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameStr compares S1 and S2 and returns true if they are identical. SameStr is case sensitive and is not affected by the current locale.

See Also

SameText (see page 4283)

CompareStr (see page 4142)

CompareText (see page 4143)

AnsicompareStr

AnsicompareText

CompareMem (see page 4142)

39.3.189 SysUtils.SameText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function SameText(const S1: string; const S2: string): Boolean; overload;
function SameText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameText(const UnicodeString S1, const UnicodeString S2);
Boolean SameText(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameText compares S1 and S2 and returns true if they are equal. SameText is not case sensitive and is not affected by the current locale.

See Also

CompareStr (see page 4142)

AnsiCompareText (see page 4115)

CompareMem (see page 4142)

CompareText (see page 4143)

39.3.190 SysUtils.SetCurrentDir Function

Sets the current directory.

Pascal

```
function SetCurrentDir(const Dir: string): Boolean;
```

C++

```
Boolean SetCurrentDir(const UnicodeString Dir);
```

File

SysUtils

Description

The SetCurrentDir function sets the current directory. The return value is true if the current directory was successfully changed, or false if an error occurred.

See Also

ChDir (see page 3638)

CreateDir (see page 4147)

ExtractFileDir (see page 4176)

GetCurrentDir (see page 4246)

RemoveDir (see page 4276)

39.3.191 SysUtils.ShowException Function

Displays an exception message with its physical address

Pascal

```
procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer);
```

C++

```
ShowException(TObject * ExceptObject, void * ExceptAddr);
```

File

SysUtils

Description

ShowException displays the message associated with an exception, together with the exception's physical address. The exception's address can be used with Search | Find Error (see page 3662) menu command to find the statement that raised the exception.

See Also

ExceptionErrorMessage (see page 4171)

ExceptObject (see page 3664)

ExceptAddr (see page 3663)

39.3.192 SysUtils.Sleep Function

Delay program execution for a specified number of milliseconds.

Pascal

```
procedure Sleep(milliseconds: Cardinal); stdcall;
```

C++

```
__stdcall Sleep(unsigned milliseconds);
```

File

SysUtils

Description

Sleep pauses program executions as specified by the milliseconds parameter. Under Windows, Sleep is just a link to the Sleep function in the system API.

39.3.193 SysUtils.StrAlloc Function

Allocates a buffer for a null-terminated string and returns a pointer to its first character.

Pascal

```
function StrAlloc(Size: Cardinal): PChar;
```

C++

```
const char * StrAlloc(unsigned Size);
```

File

SysUtils

Description

StrAlloc allocates a buffer for a null-terminated string with a maximum length of Size - 1 (1 byte must be reserved for the termination character). The result points to the location where the first character of the string is to be stored. A 32-bit number giving the total amount of memory allocated is stored in the four bytes preceding the first character; it is equal to Size + 4. If space for a string is allocated with StrAlloc, it should be deallocated via StrDispose (see page 4291).

Because AnsiStrings (long strings) are implicitly null terminated and dynamically allocated, the use of StrAlloc is deprecated.

See Also

StrDispose (see page 4291)

StrNew (see page 4310)

StrBufSize (see page 4285)

39.3.194 SysUtils.StrBufSize Function

Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4285).

Pascal

```
function StrBufSize(const Str: PAnsiChar): Cardinal; overload;
function StrBufSize(const Str: PWideChar): Cardinal; overload;
```

C++

```
unsigned StrBufSize(const PAnsiChar Str);
unsigned StrBufSize(const PWideChar Str);
```

File

SysUtils

Description

StrBufSize returns the number of bytes in Str (see page 3764), including the byte for the termination character.

Note: NoteIf Str (see page 3764) does not point to a string buffer allocated by StrAlloc (see page 4285) or StrNew (see page 4310), no error message is returned, and the result is unpredictable.

See Also

StrAlloc (see page 4285)

StrNew (see page 4310)

39.3.195 SysUtils.StrByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function StrByteType(Str: PAnsiChar; Index: Cardinal): TMbcsByteType; overload;
function StrByteType(Str: PWideChar; Index: Cardinal): TMbcsByteType; overload;
```

C++

```
TMbcsByteType StrByteType(PAnsiChar Str, unsigned Index);
TMbcsByteType StrByteType(PWideChar Str, unsigned Index);
```

File

SysUtils

Description

Call StrByteType to determine the type of the byte specified by the Index parameter, where 0 specifies the first byte in the string Str (see page 3764), 1 specifies the second byte, and so on.

If the system is not using a multi-byte character system (MBCS), StrByteType always returns mbSingleByte. Otherwise, StrByteType returns mbSingleByte if the indicated byte represents a complete character in Str (see page 3764), mbLeadByte if it represents the first byte of a double byte character, and mbTrailByte if it represents the second byte of a double byte character.

Note: No checking is done to ensure that Index is less than the length of Str (see page 3764). It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

ByteType (see page 4135)

LeadBytes (see page 4394)

39.3.196 SysUtils.StrCat Function

Appends a copy of Source to the end of Dest and returns the concatenated string.

Pascal

```
function StrCat(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrCat(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrCat(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrCat(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

Use StrCat to concatenate Source to the end of Dest. StrCat does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4303)(Dest)+StrLen (see page 4303)(Source)+1 characters.

To check length, use the StrLCat (see page 4299) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer.get(), AnsiString(Label1->Caption).t_str());
  StrCat(szBuffer.get(), AnsiString(Edit1->Text).t_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
begin
  GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
  StrCopy(Buffer, PChar(Label1.Caption));
  SysUtils.StrCat(Buffer, PChar(Edit1.Text));
  Label1.Caption := Buffer;
  Edit1.Clear;
  FreeMem(Buffer);
end;
```

See Also

StrLCat (see page 4299)

StrLen (see page 4303)

39.3.197 SysUtils.StrCharLength Function

Returns the length, in bytes, of a character.

Pascal

```
function StrCharLength(const Str: PAnsiChar): Integer; overload;
function StrCharLength(const Str: PWideChar): Integer; overload;
```

C++

```
int StrCharLength(const PAnsiChar Str);
int StrCharLength(const PWideChar Str);
```

File

SysUtils

Description

StrCharLength returns the size, in bytes, of the first character in Str (see page 3764). This value can be 1 or 2 under Windows, 1 to 6 under Linux.

See Also

ByteType (see page 4135)

LeadBytes (see page 4394)

39.3.198 SysUtils.StrComp Function

Compares two strings with case sensitivity.

Pascal

```
function StrComp(const Str1: PAnsiChar; const Str2: PAnsiChar): Integer; overload;
function StrComp(const Str1: PWideChar; const Str2: PWideChar): Integer; overload;
```

C++

```
int StrComp(const PAnsiChar Str1, const PAnsiChar Str2);
int StrComp(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

Call StrComp to compare two null-terminated strings, with case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1is the same as Str2
>0	Str1 sorts after Str2

Note: When working with International characters, use AnsiStrComp (see page 4120) instead.

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 20]);
  int iResult = StrComp(Edit1->Text.t_str(), Edit2->Text.t_str());
  StrCopy(szResult.get(), Edit1->Text.c_str());
  if (iResult < 0)
    StrCat(szResult.get(), " is less than ");
  else if (iResult > 0)
    StrCat(szResult.get(), " is greater than ");
  else
    StrCat(szResult.get(), " is equal to ");

  StrCat(szResult.get(), AnsiString(Edit2->Text).t_str());
  ShowMessage(AnsiString(szResult.get()));
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Msg: string;
  CompResult: Integer;
begin
  Msg := Edit1.Text;
  CompResult := SysUtils.StrComp(PChar(Edit1.Text), PChar(Edit2.Text));
  if CompResult < 0 then
    Msg := Msg + ' is less than '
  else if CompResult > 0 then
    Msg := Msg + ' is greater than '
  else
    Msg := Msg + ' is equal to ';
  Msg := Msg + Edit2.Text;
  ShowMessage(Msg);
end;
```

See Also

AnsiStrComp (see page 4120)

StrIComp (see page 4295)

StrLComp (see page 4300)

StrLIComp (see page 4305)

39.3.199 SysUtils.StrCopy Function

Copies a null-terminated string.

Pascal

```
function StrCopy(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrCopy(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrCopy(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrCopy(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

Use StrCopy to copy Source to Dest. StrCopy returns Dest.

StrCpy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4303)(Source)+1 characters.

For length checking, use the StrLCopy (see page 4302) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer.get(), AnsiString(Label1->Caption).t_str());
  StrCat(szBuffer.get(), AnsiString(Edit1->Text).t_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
begin
  GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
  StrCopy(Buffer, PChar(Label1.Caption));
  SysUtils.StrCat(Buffer, PChar(Edit1.Text));
  Label1.Caption := Buffer;
  Edit1.Clear;
  FreeMem(Buffer);
end;
```

See Also

StrECopy (see page 4292)

StrLCopy (see page 4302)

39.3.200 SysUtils.StrDispose Function

Disposes of a string.

Pascal

```
procedure StrDispose(Str: PAnsiChar); overload;
procedure StrDispose(Str: PWideChar); overload;
```

C++

```
StrDispose(PAnsiChar Str);
StrDispose(PWideChar Str);
```

File

SysUtils

Description

StrDispose is provided for backward compatibility. StrDispose disposes of a string on a heap that was previously allocated with StrAlloc (see page 4285) or StrNew (see page 4310).

If Str (see page 3764) is nil (Delphi) or NULL (C++), StrDispose does nothing.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.c_str());
  Application->MessageBox(
    Edit1->Text.t_str(),
    L"StrNew, StrDispose example",
    MB_OK);
  // Deallocate memory.
  StrDispose(psz);
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

StrAlloc (see page 4285)

StrNew (see page 4310)

39.3.201 SysUtils.StrECopy Function

Copies null-terminated string.

Pascal

```
function StrECopy(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrECopy(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrECopy(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrECopy(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

StrECopy copies Source to Dest and returns StrEnd (see page 4293)(Dest).

StrECopy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4303)(Source)+1 characters.

Nested calls to StrECopy to concatenate a sequence of strings run more efficiently than multiple calls to StrCat (see page 4286).

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is appended to a static buffer. Then the
buffer is displayed in the label's caption.
*/
const static int MAX_LENGTH = 15;
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static char szBuffer[MAX_LENGTH];
  static char* pszBuffer = szBuffer;

  if (StrLen(pszBuffer) + StrLen(Edit1->Text.c_str()) < MAX_LENGTH)
    pszBuffer = StrECopy(pszBuffer, Edit1->Text.c_str());
  else
    ShowMessage("Can't fit new string in allocated memory space.");
  Label1->Caption = szBuffer;
}
```

Delphi Examples:

```
{
This example uses a button on a form.  Since StrECopy returns
the destination char array, nested calls are possible and
actually more efficient than repeated calls.  StrECopy does
no bounds checking. When the button is clicked, the inline
text is appended to a static buffer. Then the buffer is
displayed in the form's canvas.
```

```
}

procedure TForm1.Button1Click(Sender: TObject);
const
  Turbo: PChar = 'Object';
  Pascal: PChar = 'Pascal';
 var
  S: array[0..15] of Char;
begin
  SysUtils.StrECopy(SysUtils.StrECopy(SysUtils.StrECopy(S, Turbo), ' '), Pascal);
  Canvas.TextOut(10, 10, string(S));
end;
```

See Also

StrCat (see page 4286)

StrCopy (see page 4289)

StrEnd (see page 4293)

StrLen (see page 4303)

39.3.202 SysUtils.StrEnd Function

Returns a pointer to the end of a null terminated string.

Pascal

```
function StrEnd(const Str: PAnsiChar): PAnsiChar; overload;
function StrEnd(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrEnd(const PAnsiChar Str);
PWideChar StrEnd(const PWideChar Str);
```

File

SysUtils

Description

StrEnd returns a pointer to the null character at the end of Str (see page 3764).

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is copied into a buffer. Then the buffer is
displayed backwards in the label.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> pszEditText(new char[Edit1->Text.Length()+1]);
  char *psz;
  StrCopy(pszEditText.get(), Edit1->Text.c_str());
  psz = StrEnd(pszEditText.get());
  Label1->Caption = "";
  while (psz >= &pszEditText.get()[0])
  {
```

```
    psz--;
    Label1->Caption = Label1->Caption + *psz;
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is copied into a buffer. Then the buffer is
displayed backwards in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TextBuffer: PChar;
  Ptr: PChar;
begin
  GetMem(TextBuffer, Length(Edit1.Text)+1);
  StrCopy(TextBuffer, PChar(Edit1.Text));
  Ptr := StrEnd(TextBuffer);
  Label1.Caption := '';
  while Ptr > TextBuffer do
  begin
    Dec(Ptr);
    Label1.Caption := Label1.Caption + Ptr^;
  end;
  FreeMem(TextBuffer);
end;
```

See Also

AnsiStrLastChar (see page 4121)

StrLen (see page 4303)

39.3.203 SysUtils.StrFmt Function

Formats entries in an array.

Pascal

```
function StrFmt(Buffer: PAnsiChar; Format: PAnsiChar; const Args: array of const): PAnsiChar;
overload;
function StrFmt(Buffer: PAnsiChar; Format: PAnsiChar; const Args: array of const; const
FormatSettings: TFormatSettings): PAnsiChar; overload;
function StrFmt(Buffer: PWideChar; Format: PWideChar; const Args: array of const): PWideChar;
overload;
function StrFmt(Buffer: PWideChar; Format: PWideChar; const Args: array of const; const
FormatSettings: TFormatSettings): PWideChar; overload;
```

C++

```
PAnsiChar StrFmt(PAnsiChar Buffer, PAnsiChar Format, const array of const Args);
PAnsiChar StrFmt(PAnsiChar Buffer, PAnsiChar Format, const array of const Args, const
TFormatSettings FormatSettings);
PWideChar StrFmt(PWideChar Buffer, PWideChar Format, const array of const Args);
PWideChar StrFmt(PWideChar Buffer, PWideChar Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrFmt formats the series of arguments from an open array into a buffer.

Buffer is a buffer that receives the results.

Format (see page 4232) is a null-terminated format string that indicates how to generate the results. For more information about how the result is formatted, see Format (see page 4232) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 4232).

Note: Args_Size gives the index of the last argument (one less than the number of elements in Args).

StrFmt returns a pointer to the destination buffer.

The first form of StrFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

Format (see page 4232)

GetLocaleFormatSettings (see page 4250)

39.3.204 SysUtils.StrIComp Function

Compares two strings without case sensitivity.

Pascal

```
function StrIComp(const Str1: PAnsiChar; const Str2: PAnsiChar): Integer; overload;
function StrIComp(const Str1: PWideChar; const Str2: PWideChar): Integer; overload;
```

C++

```
int StrIComp(const PAnsiChar Str1, const PAnsiChar Str2);
int StrIComp(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

Call StrIComp to compare two strings without case sensitivity. StrIComp returns a value greater than 0 if Str1 > Str2, less than 0 if Str1 < Str2, and returns 0 if the strings are equal except for differences in case.

When working with international characters, use AnsiStrIComp (see page 4121) instead.

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared case-insensitively.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 20]);
```

```
  int iResult = StrIComp(Edit1->Text.c_str(), Edit2->Text.c_str());

  StrCopy(szResult.get(), Edit1->Text.c_str());
  if (iResult < 0)
    StrCat(szResult.get(), " is less than ");
  else if (iResult > 0)
    StrCat(szResult.get(), " is greater than ");
  else
    StrCat(szResult.get(), " is equal to ");
  StrCat(szResult.get(), Edit2->Text.c_str());
  ShowMessage(AnsiString(szResult.get()));
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared case-insensitively.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Msg: string;
  CompResult: Integer;
begin
  Msg := Edit1.Text;
  CompResult := StrIComp(PChar(Edit1.Text), PChar(Edit2.Text));
  if CompResult < 0 then
    Msg := Msg + ' is less than '
  else if CompResult > 0 then
    Msg := Msg + ' is greater than '
  else
    Msg := Msg + ' is equal to ';
  Msg := Msg + Edit2.Text;
  ShowMessage(Msg);
end;
```

See Also

AnsiStrIComp (see page 4121)

StrComp (see page 4288)

StrLComp (see page 4300)

StrLIComp (see page 4305)

39.3.205 SysUtils.StringOf Function

Pascal

```
function StringOf(const Bytes: TBytes): UnicodeString;
```

C++

```
UnicodeString StringOf(const TBytes Bytes);
```

File

SysUtils

Description

This is function SysUtils.StringOf.

39.3.206 SysUtils.StringReplace Function

Returns a string with occurrences of one substring replaced by another substring.

Pascal

```
function StringReplace(const S: string; const OldPattern: string; const NewPattern: string;
Flags: TReplaceFlags): string; overload;
```

C++

```
UnicodeString StringReplace(const UnicodeString S, const UnicodeString OldPattern, const
UnicodeString NewPattern, TReplaceFlags Flags);
```

File

SysUtils

Description

StringReplace replaces occurrences of the substring specified by OldPattern with the substring specified by NewPattern. StringReplace assumes that the source string may contain Multibyte characters.

S is the source string, whose substrings are changed.

OldPattern is the substring to locate and replace with NewPattern.

NewPattern is the substring to substitute for occurrences of OldPattern.

Flags is a set of flags that govern how StringReplace locates and replaces occurrences of OldPattern. If Flags does not include rfReplaceAll, StringReplace only replaces the first occurrence of OldPattern in S. Otherwise, StringReplace replaces all instances of OldPattern with NewPattern. If the Flags parameter includes rfIgnoreCase, the comparison operation is case insensitive.

39.3.207 SysUtils.StringToGUID Function

Converts a string to a GUID.

Pascal

```
function StringToGUID(const S: string): TGUID;
```

C++

```
TGUID StringToGUID(const UnicodeString S);
```

File

SysUtils

Description

StringToGUID converts the string representation of a GUID, S, into an equivalent TGUID (see page 3800) data structure.

If StringToGUID succeeds it returns the requested GUID. If StringToGUID fails an EConvertError (see page 3985) exception is raised.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
```

```
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
```

```
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
  if IsEqualGUID(MyGuid0, MyGuid1) then
     Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                     '2 randomly generated GUIDs cannot be equal!');
end;
```

See Also

GUIDToString (see page 4254)

39.3.208 SysUtils.StrLCat Function

Appends up to a specified maximum number of characters to string.

Pascal

```
function StrLCat(Dest: PAnsiChar; const Source: PAnsiChar; MaxLen: Cardinal): PAnsiChar;
overload;
function StrLCat(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrLCat(PAnsiChar Dest, const PAnsiChar Source, unsigned MaxLen);
PWideChar StrLCat(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCat appends at most MaxLen - StrLen (see page 4303)(Dest) characters from Source to the end of Dest and returns Dest. That is, MaxLen indicates the maximum length that is allowed in the result string.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString EditText = AnsiString(Edit1->Text);
  char* szFirstHalf = new char[StrLen(EditText.c_str())/2+2];
  char* szSecondHalf = new char[StrLen(EditText.c_str())/2+2];
  char* pszSecondHalf = &EditText[StrLen(EditText.c_str())/2+1];

  *szFirstHalf = 0;
  *szSecondHalf = 0;

  StrLCat(szFirstHalf, EditText.c_str(), StrLen(EditText.c_str())/2);
  StrCat(szSecondHalf, pszSecondHalf);

  Application->MessageBox(WideString(szFirstHalf).c_bstr(), L"First Half", MB_OKCANCEL);
  Application->MessageBox(WideString(szSecondHalf).c_bstr(), L"Second Half", MB_OKCANCEL);

  delete [] szFirstHalf;
  delete [] szSecondHalf;
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstHalf: PChar;
  SecondHalf: PChar;
  HalfLen: Integer;
begin
  HalfLen := StrLen(PChar(string(Edit1.Text))) div 2;
  GetMem(FirstHalf,HalfLen+2);
  GetMem(SecondHalf,HalfLen+2);
  FirstHalf^ := Chr(0);
  SecondHalf^ := Chr(0);
  StrLCat(FirstHalf, PChar(string(Edit1.Text)), HalfLen);
  StrCat(SecondHalf, PChar(string(Edit1.Text)) + HalfLen);
//  Application.MessageBox(FirstHalf, 'First Half', [smbOK], smsInformation, smbOK, smbCancel);
  Application.MessageBox(FirstHalf, 'First Half', MB_OKCANCEL);
  Application.MessageBox(SecondHalf, 'Second Half', MB_OKCANCEL);
  FreeMem(FirstHalf);
  FreeMem(SecondHalf);
end;
```

See Also

SizeOf (see page 3761)

StrCat (see page 4286)

StrLen (see page 4303)

39.3.209 SysUtils.StrLComp Function

Compares up to a specified maximum number of characters in two strings.

Pascal

```
function StrLComp(const Str1: PAnsiChar; const Str2: PAnsiChar; MaxLen: Cardinal): Integer;
overload;
function StrLComp(const Str1: PWideChar; const Str2: PWideChar; MaxLen: Cardinal): Integer;
overload;
```

C++

```
int StrLComp(const PAnsiChar Str1, const PAnsiChar Str2, unsigned MaxLen);
int StrLComp(const PWideChar Str1, const PWideChar Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLComp compares Str1 to Str2, up to a maximum length of MaxLen characters. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1 is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLComp (see page 4122) instead.

C++ Examples:

```
/*
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings.
*/
#include <algorithm>  // for the min function
#include <memory>       // for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 65]);
  int len = std::min(Edit1->Text.Length(), Edit2->Text.Length());
  int iResult = Sysutils::StrLComp(
    Edit1->Text.c_str(), Edit2->Text.c_str(), len + 1);
  StrCopy(szResult.get(), "The first ");
  StrCat(szResult.get(), IntToStr(len).c_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit1->Text.c_str());
  if (iResult < 0)
    StrCat(szResult.get(), " are less than the first ");
  else if (iResult > 0)
    StrCat(szResult.get(), " are greater than the first ");
  else
    StrCat(szResult.get(), " are equal to the first ");
  StrCat(szResult.get(), IntToStr(len).c_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit2->Text.c_str());

  ShowMessage(szResult.get());
}
```

Delphi Examples:

```
{
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S1, S2: String;
  Pos: Integer;
  Equal: Boolean;
begin
  Pos := 1;
  Equal := True;
  S1 := Edit1.Text;
  S2 := Edit2.Text;
  Form1.Repaint;
  while (Equal = True) do
  begin
    if StrLComp(PChar(S1), PChar(S2), Pos) = 0 then
```

```
      Inc(Pos)
    else
      Equal := False;
  end;
  Canvas.TextOut(10, 10, 'The first ' + IntToStr(Pos - 1) +
    ' characters are equal.');
end;
```

See Also

AnsiStrLComp (see page 4122)

StrComp (see page 4288)

StrIComp (see page 4295)

StrLIComp (see page 4305)

39.3.210 SysUtils.StrLCopy Function

Copies up to a specified maximum number of characters from Source to Dest.

Pascal

```
function StrLCopy(Dest: PAnsiChar; const Source: PAnsiChar; MaxLen: Cardinal): PAnsiChar;
overload;
function StrLCopy(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrLCopy(PAnsiChar Dest, const PAnsiChar Source, unsigned MaxLen);
PWideChar StrLCopy(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCopy copies at most MaxLen characters from Source to Dest, then adds a null terminator to Dest and returns Dest. The SizeOf (see page 3761) standard function (Delphi) or the sizeof operator (C++) can be used to determine the MaxLen parameter. Usually MaxLen equals SizeOf (see page 3761)(Dest)-1.

Note: StrLCopy copies a null terminator as well as MaxLen characters. Be sure that Dest is large enough to accommodate the null terminator at the end.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the first X bytes of the
edit control are copied into a buffer, where X is a
predefined number.
*/
const MAX_BUFFER = 10;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szBuffer[MAX_BUFFER+1];
  StrLCopy(szBuffer, Edit1->Text.c_str(), MAX_BUFFER);
  Application->MessageBox(WideString(szBuffer).c_bstr(), L"StrLCopy example", MB_OKCANCEL);
```

```
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form. When the button is clicked, the first X bytes of the
edit control are copied into a buffer, where X is a
predefined number.
}
const MAX_BUFFER = 10;
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: array [0..MAX_BUFFER] of char;
begin
  StrLCopy(Buffer, PChar(Edit1.Text), MAX_BUFFER);
  Application.MessageBox(Buffer, 'StrLCopy example', MB_OKCANCEL);
end;
```

See Also

SizeOf (see page 3761)

StrCopy (see page 4289)

StrLen (see page 4303)

39.3.211 SysUtils.StrLen Function

Returns number of characters in a string excluding the null terminator.

Pascal

```
function StrLen(const Str: PAnsiChar): Cardinal; overload;
function StrLen(const Str: PWideChar): Cardinal; overload;
```

C++

```
unsigned StrLen(const PAnsiChar Str);
unsigned StrLen(const PWideChar Str);
```

File

SysUtils

Description

StrLen returns the number of characters in Str (see page 3764), not counting the null terminator.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString EditText = AnsiString(Edit1->Text);
  char* szFirstHalf = new char[StrLen(EditText.c_str())/2+2];
  char* szSecondHalf = new char[StrLen(EditText.c_str())/2+2];
  char* pszSecondHalf = &EditText[StrLen(EditText.c_str())/2+1];
```

```
  *szFirstHalf = 0;
  *szSecondHalf = 0;

  StrLCat(szFirstHalf, EditText.c_str(), StrLen(EditText.c_str())/2);
  StrCat(szSecondHalf, pszSecondHalf);

  Application->MessageBox(WideString(szFirstHalf).c_bstr(), L"First Half", MB_OKCANCEL);
  Application->MessageBox(WideString(szSecondHalf).c_bstr(), L"Second Half", MB_OKCANCEL);

  delete [] szFirstHalf;
  delete [] szSecondHalf;
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstHalf: PChar;
  SecondHalf: PChar;
  HalfLen: Integer;
begin
  HalfLen := StrLen(PChar(string(Edit1.Text))) div 2;
  GetMem(FirstHalf,HalfLen+2);
  GetMem(SecondHalf,HalfLen+2);
  FirstHalf^ := Chr(0);
  SecondHalf^ := Chr(0);
  StrLCat(FirstHalf, PChar(string(Edit1.Text)), HalfLen);
  StrCat(SecondHalf, PChar(string(Edit1.Text)) + HalfLen);
//  Application.MessageBox(FirstHalf, 'First Half', [smbOK], smsInformation, smbOK, smbCancel);
  Application.MessageBox(FirstHalf, 'First Half', MB_OKCANCEL);
  Application.MessageBox(SecondHalf, 'Second Half', MB_OKCANCEL);
  FreeMem(FirstHalf);
  FreeMem(SecondHalf);
end;
```

See Also

StrEnd (see page 4293)

39.3.212 SysUtils.StrLFmt Function

Formats a series of arguments from a specified open array into a buffer.

Pascal

```
function StrLFmt(Buffer: PAnsiChar; MaxBufLen: Cardinal; Format: PAnsiChar; const Args: array
of const): PAnsiChar; overload;
function StrLFmt(Buffer: PAnsiChar; MaxBufLen: Cardinal; Format: PAnsiChar; const Args: array
of const; const FormatSettings: TFormatSettings): PAnsiChar; overload;
function StrLFmt(Buffer: PWideChar; MaxBufLen: Cardinal; Format: PWideChar; const Args: array
of const): PWideChar; overload;
function StrLFmt(Buffer: PWideChar; MaxBufLen: Cardinal; Format: PWideChar; const Args: array
of const; const FormatSettings: TFormatSettings): PWideChar; overload;
```

C++

```
PAnsiChar StrLFmt(PAnsiChar Buffer, unsigned MaxBufLen, PAnsiChar Format, const array of const
```

```
Args);
PAnsiChar StrLFmt(PAnsiChar Buffer, unsigned MaxBufLen, PAnsiChar Format, const array of const
Args, const TFormatSettings FormatSettings);
PWideChar StrLFmt(PWideChar Buffer, unsigned MaxBufLen, PWideChar Format, const array of const
Args);
PWideChar StrLFmt(PWideChar Buffer, unsigned MaxBufLen, PWideChar Format, const array of const
Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrLFmt formats the series of arguments in and open array and writes the results to a buffer.

Buffer is the buffer that receives the results.

MaxLen is the maximum number of bytes that can be written to Buffer. This value does not include the terminating null character.

Format (see page 4232) is the format string that governs the operation.

Args is an array of arguments that replace the format specifiers in Format (see page 4232).

Note: Args_Size is the index of the last argument in Args (one less than the number of arguments).

StrLFmt returns a pointer to the destination buffer.

The first form of StrLFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrLFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrLFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

FormatBuf (see page 4237)

StrFmt (see page 4294)

FmtStr (see page 4229)

GetLocaleFormatSettings (see page 4250)

39.3.213 SysUtils.StrLIComp Function

Compares strings up to a specified maximum number of characters, without case sensitivity.

Pascal

```
function StrLIComp(const Str1: PAnsiChar; const Str2: PAnsiChar; MaxLen: Cardinal): Integer;
overload;
function StrLIComp(const Str1: PWideChar; const Str2: PWideChar; MaxLen: Cardinal): Integer;
overload;
```

C++

```
int StrLIComp(const PAnsiChar Str1, const PAnsiChar Str2, unsigned MaxLen);
int StrLIComp(const PWideChar Str1, const PWideChar Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLIComp compares Str1 to Str2, up to a maximum length of MaxLen characters, without case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1 is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLIComp (see page 4122) instead.

C++ Examples:

```
/*
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings without case sensitivity.
*/
#include <algorithm>  // for the min function
#include <memory>       // for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 65]);
  int len = std::min(Edit1->Text.Length(), Edit2->Text.Length());
  int iResult = StrLIComp(
    Edit1->Text.c_str(), Edit2->Text.c_str(), len);

  StrCopy(szResult.get(), "The first ");
  StrCat(szResult.get(), IntToStr(len).c_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit1->Text.c_str());
  if (iResult < 0)
    StrCat(szResult.get(), " are less than the first ");
  else if (iResult > 0)
    StrCat(szResult.get(), " are greater than the first ");
  else
    StrCat(szResult.get(), " are equal to the first ");
  StrCat(szResult.get(), IntToStr(len).c_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit2->Text.c_str());

  ShowMessage(szResult.get());
}
```

Delphi Examples:

```
{
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings, without case sensitivity.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S1, S2: String;
  Pos: Integer;
  Equal: Boolean;
begin
  Pos := 1;
  Equal := True;
```

```
    S1 := Edit1.Text;
    S2 := Edit2.Text;
    Form1.Repaint;
    while (Equal = True) do
    begin
      if StrLIComp(PChar(S1), PChar(S2), Pos) = 0 then
        Inc(Pos)
      else
        Equal := False;
    end;
    Canvas.TextOut(10, 10, 'The first ' + IntToStr(Pos - 1) +
      ' characters are equal.');
end;
```

See Also

AnsiStrLIComp (see page 4122)

StrComp (see page 4288)

StrIComp (see page 4295)

StrLComp (see page 4300)

39.3.214 SysUtils.StrLower Function

Converts a string to lowercase.

Pascal

```
function StrLower(Str: PAnsiChar): PAnsiChar; overload;
function StrLower(Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrLower(PAnsiChar Str);
PWideChar StrLower(PWideChar Str);
```

File

SysUtils

Description

StrLower converts Str (see page 3764) to lowercase and returns Str (see page 3764).

Note: If the source string contains international characters, use AnsiStrLower (see page 4123) instead. If the source string is an AnsiString (long string), use LowerCase (see page 4269) (or AnsiLowerCase (see page 4117)) instead.

C++ Examples:

```
/*
The following example uses an edit box, two labels, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in label1's
caption and upper case in label2's caption.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = StrLower(Edit1->Text.c_str());
  Label2->Caption = StrUpper(Edit1->Text.c_str());
}
```

Delphi Examples:

```
{
The following example uses an edit box, a label, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in the label's
caption.
}
var
  S: array[0..20] of Char = 'A fUnNy StRiNg';

procedure TForm1.Button1Click(Sender: TObject);
var
  lower, upper : string;
begin
  lower := string(StrLower(S));
  upper := string(StrUpper(S));
  Canvas.TextOut(5, 10, lower + ' ' + upper);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.Caption := S;
end;
```

See Also

AnsiLowerCase (see page 4117)

AnsiStrLower (see page 4123)

LowerCase (see page 4269)

StrUpper (see page 4334)

39.3.215 SysUtils.StrMove Function

Copies specified number of characters to string.

Pascal

```
function StrMove(Dest: PAnsiChar; const Source: PAnsiChar; Count: Cardinal): PAnsiChar;
overload;
function StrMove(Dest: PWideChar; const Source: PWideChar; Count: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrMove(PAnsiChar Dest, const PAnsiChar Source, unsigned Count);
PWideChar StrMove(PWideChar Dest, const PWideChar Source, unsigned Count);
```

File

SysUtils

Description

StrMove copies exactly Count characters from Source to Dest and returns Dest. Source and Dest can overlap.

C++ Examples:

```
/*
This example shows two functions that do the same thing as
StrNew and StrDispose.
*/
```

```
char * AHeapaString(char *S)
{
// Allocate string on heap
  if ((S != NULL) && (S[0] != '\0'))
  {
    int L = StrLen(S) + 1;
    char *p = (char *)malloc(L);
    return StrMove(p, S, L);
  }
  return NULL;
}
void DisposeDaString(char *S)
{
// Dispose string on heap
  if (S != NULL)
    free(S);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char *S = "Now you see it.";
  char *otherS = AHeapaString(S);
  Canvas->TextOut(5, 10, otherS);
  Canvas->Refresh();
  Sleep(5000);
  DisposeDaString(otherS);
}
```

Delphi Examples:

```
{
This example shows two functions that do the same thing as
StrNew and StrDispose.
}
function AHeapaString(S: PChar): PChar;
{ Allocate string on heap }
 var
  L: Cardinal;
  P: PChar;
  StrNew: PChar;
begin
  StrNew := nil;
  if (S <> nil) and (S[0] <> #0) then
  begin
    L := StrLen(S) + 1;
    GetMem(P, L);
    StrNew := StrMove(P, S, L);
  end;
  Result := StrNew;
end;

procedure DisposeDaString(S: PChar);
{ Dispose string on heap }
begin
  if S <> nil then FreeMem(S, StrLen(S) + 1);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, otherS: PChar;
begin
  S := 'Now you see it';
  otherS := AHeapaString(S);
  Canvas.TextOut(5, 10, otherS);
  Canvas.Refresh;
  Sleep(5000);
  DisposeDaString(otherS);
end;
```

See Also

Move (see page 3711)

39.3.216 SysUtils.StrNew Function

Allocates space on and copies a string to the heap; returning a pointer to the string.

Pascal

```
function StrNew(const Str: PAnsiChar): PAnsiChar; overload;
function StrNew(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrNew(const PAnsiChar Str);
PWideChar StrNew(const PWideChar Str);
```

File

SysUtils

Description

StrNew allocates a copy of Str (see page 3764) on the heap.

If Str (see page 3764) is nil (Delphi) or NULL (C++) or points to an empty string, StrNew returns a pointer to a new empty string.

Otherwise, StrNew makes a duplicate of Str (see page 3764), obtaining space with a call to StrAlloc (see page 4285), and returns a pointer to the duplicated string.

The allocated space is the length of Str (see page 3764) + 5 bytes.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.c_str());
  Application->MessageBox(
    Edit1->Text.t_str(),
    L"StrNew, StrDispose example",
    MB_OK);
  // Deallocate memory.
  StrDispose(psz);
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

GetMem (see page 3683)

StrDispose (see page 4291)

StrAlloc (see page 4285)

39.3.217 SysUtils.StrNextChar Function

Returns pointer to following character.

Pascal

```
function StrNextChar(const Str: PAnsiChar): PAnsiChar; overload;
function StrNextChar(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrNextChar(const PAnsiChar Str);
PWideChar StrNextChar(const PWideChar Str);
```

File

SysUtils

Description

Note: StrNextChar returns a pointer to the second character in Str (see page 3764), allowing for multi-byte characters.

39.3.218 SysUtils.StrPas Function

Converts null-terminated string to an AnsiString (long string).

Pascal

```
function StrPas(const Str: PAnsiChar): AnsiString; overload;
function StrPas(const Str: PWideChar): UnicodeString; overload;
```

C++

```
AnsiString StrPas(const PAnsiChar Str);
UnicodeString StrPas(const PWideChar Str);
```

File

SysUtils

Description

This function is provided for backwards compatibility only. To convert a null terminated string to an AnsiString or native Delphi language string, use a typecast or an assignment.

C++ Examples:

```
/*
The following code locks the memory for text on the Clipboard, then
reads the text.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
AnsiString MyString;

int TextHandle;
PChar pText;

Clipboard()->Open();
try
{
  TextHandle = Clipboard()->GetAsHandle(CF_UNICODETEXT);
  pText = (PChar)GlobalLock((HGLOBAL)TextHandle);
  MyString = pText;
  ListBox1->Items->Add(StrPas(pText));
  GlobalUnlock((HGLOBAL)TextHandle);
}
catch (...)
{
  Clipboard()->Close();
  throw;
}
Clipboard()->Close();
}
```

Delphi Examples:

```
{
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyHandle: THandle;
  TextPtr: PChar;
begin
  ClipBoard.Open;
try
{$IFNDEF UNICODE}
  MyHandle := Clipboard.GetAsHandle(CF_TEXT);
{$ELSE}
  MyHandle := Clipboard.GetAsHandle(CF_UNICODETEXT);
{$ENDIF}

  TextPtr := GlobalLock(MyHandle);
  ListBox1.Items.Add(StrPas(TextPtr));
  GlobalUnlock(MyHandle);
finally
  Clipboard.Close;
end;
end;
```

See Also

StrPCopy (see page 4313)

39.3.219 SysUtils.StrPCopy Function

Copies an AnsiString (long string) to a null-terminated string.

Pascal

```
function StrPCopy(Dest: PAnsiChar; const Source: AnsiString): PAnsiChar; overload;
function StrPCopy(Dest: PWideChar; const Source: UnicodeString): PWideChar; overload;
```

C++

```
PAnsiChar StrPCopy(PAnsiChar Dest, const AnsiString Source);
PWideChar StrPCopy(PWideChar Dest, const UnicodeString Source);
```

File

SysUtils

Description

StrPCopy copies Source into a null-terminated string Dest. It returns a pointer to Dest.

StrPCopy does not perform any length checking.

The destination buffer must have room for at least Length (see page 3702)(Source)+1 characters.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the label's
caption and the edit control's text are combined in a buffer.
Then the buffer is displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  StrPCopy(szBuffer.get(), Label1->Caption);
  StrCat(szBuffer.get(), Edit1->Text.c_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
This example requires a button on a form and uses StrPCopy
to copy a string into an array and then display the character
array on the form canvas.  StrPCopy does not do length checking.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  A: array[0..79] of Char;
  S: String;
begin
  S := 'Honk if you know Blaise.';
  StrPCopy(A, S);
  Canvas.TextOut(10, 10, string(A));
end;
```

See Also

StrCopy (see page 4289)

39.3.220 SysUtils.StrPLCopy Function

Copies characters from an AnsiString (long string) into a null-terminated string.

Pascal

```
function StrPLCopy(Dest: PAnsiChar; const Source: AnsiString; MaxLen: Cardinal): PAnsiChar;
overload;
function StrPLCopy(Dest: PWideChar; const Source: UnicodeString; MaxLen: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrPLCopy(PAnsiChar Dest, const AnsiString Source, unsigned MaxLen);
PWideChar StrPLCopy(PWideChar Dest, const UnicodeString Source, unsigned MaxLen);
```

File

SysUtils

Description

StrPLCopy copies up to a maximum of MaxLen characters from Source into the null-terminated string Dest. StrPLCopy returns a pointer to Dest.

Note: Dest should point to a location with room for MaxLen + 1 characters, including the null terminator.

39.3.221 SysUtils.StrPos Function

Returns a pointer to the first occurrence of STR2 in STR1.

Pascal

```
function StrPos(const Str1: PAnsiChar; const Str2: PAnsiChar): PAnsiChar; overload;
function StrPos(const Str1: PWideChar; const Str2: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrPos(const PAnsiChar Str1, const PAnsiChar Str2);
PWideChar StrPos(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

StrPos returns a pointer to the first occurrence of Str2 in Str1. If Str2 does not occur in Str1, StrPos returns nil (Delphi) or NULL (C++).

Note: If the source string contains international characters, use AnsiStrPos (see page 4124) instead.

C++ Examples:

```
/*
```

```
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
edit control is searched for the text in the second edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (StrPos(Edit1->Text.c_str(), Edit2->Text.c_str()))
    ShowMessage("Substring found.");
  else
    ShowMessage("Substring not found.");
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
edit control is searched for the text in the second edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if StrPos(PChar(Edit1.Text), PChar(Edit2.Text)) <> nil then
    ShowMessage('Substring found')
  else
    ShowMessage('Substring not found');
end;
```

See Also

AnsiStrPos (see page 4124)

39.3.222 SysUtils.StrRScan Function

Returns a pointer to the last occurrence of a specified character in a string.

Pascal

```
function StrRScan(const Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function StrRScan(const Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrRScan(const PAnsiChar Str, AnsiChar Chr);
PWideChar StrRScan(const PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

StrRScan returns a pointer to the last occurrence of Chr (see page 3640) in Str (see page 3764).

If Chr (see page 3640) does not occur in Str (see page 3764), StrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrRScan (see page 4124) instead.

C++ Examples:

```
/*
The following example uses an Open dialog box and a button
on a form. When the button is clicked, the user is prompted
```

```
  for a filename. Then the filename, minus its path, is
  displayed in a message box.
  Note:   This technique will not work with multi-byte
  character sets (MBCS). To be more general, use ExtractFileName.
  */
  char *__fastcall GetFileName(char *pszFilePath)
  {
    char *pszFileName = StrRScan(pszFilePath, '\\');
    if (!pszFileName)
      pszFileName = pszFilePath;
    else
      pszFileName++;
    return pszFileName;
  }
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    if (OpenDialog1->Execute())
    {
      char *name = GetFileName(OpenDialog1->FileName.c_str());
      Application->MessageBox(
        WideString(name).c_bstr(),
        L"File Name",
        MB_OK);
    }
  }
```

Delphi Examples:

```
  {
  This function behaves much like ExtractFileName except that
  it uses null-terminated strings and it will not work with
  multi-byte character sets (MBCS).
  }
  function NamePart(FileName: PAnsiChar): PAnsiChar;
   var
    P: PAnsiChar;
    sep: Char;
  begin
    sep := '/';
    P := StrRScan(FileName, '/');
    if P = nil then
    begin
      P := StrRScan(FileName, ':');
      if P = nil then P := FileName;
    end;
    NamePart := P;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    S : string;
  begin
    S := string(NamePart('C:\Test.fil'));
    Canvas.TextOut(10, 10, S);
  end;
```

See Also

AnsiStrRScan (see page 4124)

StrScan (see page 4317)

39.3.223 SysUtils.StrScan Function

Returns a pointer to first occurrence of a specified character in a string.

Pascal

```
function StrScan(const Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function StrScan(const Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrScan(const PAnsiChar Str, AnsiChar Chr);
PWideChar StrScan(const PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

StrScan returns a pointer to the first occurrence of Chr (see page 3640) in Str (see page 3764). If Chr (see page 3640) does not occur in Str (see page 3764), StrScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrScan (see page 4125) instead.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the text in the edit
control is searched for a wildcard (asterisk character).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (StrScan(Edit1->Text.c_str(), '*'))
    ShowMessage("Wildcard found.");
  else
    ShowMessage("Wildcard not found.");
}
```

Delphi Examples:

```
{
The following example uses a button on a form. When the
button is clicked, the text is searched for a wildcard
(asterisk character).
}
function HasWildcards(FileName: PChar): Boolean;
{ Return true if file name has wildcards in it }
begin
  HasWildcards := (StrScan(FileName, '*') <> nil) or
    (StrScan(FileName, '?') <> nil);
end;

procedure TForm1.Button1Click(Sender: TObject);
const
  P: PChar = 'C:\Test.* ';
begin
  if HasWildcards(P) then
    ShowMessage('The string has wildcards')
  else
```

```
    ShowMessage('The string does not have wildcards');
end;
```

See Also

AnsiStrScan (see page 4125)

StrRScan (see page 4315)

39.3.224 SysUtils.StrToBool Function

Converts a string to a Boolean value.

Pascal

```
function StrToBool(const S: string): Boolean; overload;
```

C++

```
Boolean StrToBool(const UnicodeString S);
```

File

SysUtils

Description

StrToBool converts the string specified by S to a Boolean. If S can be converted to a number, StrToBool returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 4403) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4392) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4403) or FalseBoolStrs (see page 4392), StrToBool raises an EConvertError (see page 3985) exception.

Note: If you do not assign a value to TrueBoolStrs (see page 4403), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 4392), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4403) and FalseBoolStrs (see page 4392) variables are undefined when an application starts. Any use of StrToBool, BoolToStr (see page 4130) or TryStrToBool (see page 4346) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4130)

TrueBoolStrs (see page 4403)

FalseBoolStrs (see page 4392)

39.3.225 SysUtils.StrToBoolDef Function

Converts a string to a Boolean value, with error default.

Pascal

```
function StrToBoolDef(const S: string; const Default: Boolean): Boolean; overload;
```

C++

```
Boolean StrToBoolDef(const UnicodeString S, const Boolean Default);
```

File

SysUtils

Description

StrToBoolDef converts the string specified by S to a Boolean. If S can be converted to a number, StrToBoolDef returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBoolDef returns true if S is any of the strings listed in TrueBoolStrs (see page 4403) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4392) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4403) or FalseBoolStrs (see page 4392), StrToBoolDef returns Default.

Note: If you do not assign a value to TrueBoolStrs (see page 4403), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 4392), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4403) and FalseBoolStrs (see page 4392) variables are undefined when an application starts. Any use of StrToBool (see page 4318), BoolToStr (see page 4130) or TryStrToBool (see page 4346) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4130)

TrueBoolStrs (see page 4403)

FalseBoolStrs (see page 4392)

StrToBool (see page 4318)

39.3.226 SysUtils.StrToCurr Function

Converts a string to a Currency (see page 3242) value.

Pascal

```
function StrToCurr(const S: string): Currency; overload;
function StrToCurr(const S: string; const FormatSettings: TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurr(const UnicodeString S);
Currency StrToCurr(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToCurr to convert a string that represents a floating-point value to the corresponding Currency (see page 3242) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurr raises an EConvertError (see page 3985) exception.

The first form of StrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.227 SysUtils.StrToCurrDef Function

Converts a string to a Currency (see page 3242) value, with error default.

Pascal

```
function StrToCurrDef(const S: string; const Default: Currency): Currency; overload;
function StrToCurrDef(const S: string; const Default: Currency; const FormatSettings:
TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurrDef(const UnicodeString S, const Currency Default);
Currency StrToCurrDef(const UnicodeString S, const Currency Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToCurrDef to convert a string that represents a floating-point value to the corresponding Currency (see page 3242) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurrDef returns Default.

The first form of StrToCurrDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurrDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurrDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.228 SysUtils.StrToDate Function

Converts a string to a TDateTime (see page 3335) value.

Pascal

```
function StrToDate(const S: string): TDateTime; overload;
function StrToDate(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDate(const UnicodeString S);
TDateTime StrToDate(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDate to parse a string that specifies a date. If S does not contain a valid date, StrToDate raises an EConvertError (see page 3985) exception.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4389) global variable or its TFormatSettings (see page 4367) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4398) global variable or its TFormatSettings (see page 4367) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of StrToDate is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

39.3.229 SysUtils.StrToDateDef Function

Converts a string to a TDateTime (see page 3335) value, with error default.

Pascal

```
function StrToDateDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToDateDef(const UnicodeString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToDateDef to parse a string that specifies a date. If S does not contain a valid date, StrToDateDef returns Default.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4389) global variable or its TFormatSettings (see page 4367) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4398) global variable or its TFormatSettings (see page 4367) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of StrToDateDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.230 SysUtils.StrToDateTime Function

Converts a string to a TDateTime (see page 3335) value.

Pascal

```
function StrToDateTime(const S: string): TDateTime; overload;
function StrToDateTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDateTime(const UnicodeString S);
TDateTime StrToDateTime(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTime raises an EConvertError (see page 3985) exception.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of StrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.231 SysUtils.StrToDateTimeDef Function

Converts a string to a TDateTime (see page 3335) value with error default.

Pascal

```
function StrToDateTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateTimeDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToDateTimeDef(const UnicodeString S, const TDateTime Default, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDateTimeDef to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTimeDef returns Default.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of StrToDateTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

DateTimeToStr (see page 4153)

StrToDate (see page 4320)

StrToTime (see page 4332)

CurrencyAndDatetimeFormatting

StrToDateTime (see page 4322)

39.3.232 SysUtils.StrToFloat Function

Converts a given string to a floating-point value.

Pascal

```
function StrToFloat(const S: string): Extended; overload;
function StrToFloat(const S: string; const FormatSettings: TFormatSettings): Extended;
overload;
```

C++

```
Extended StrToFloat(const UnicodeString S);
Extended StrToFloat(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use StrToFloat to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4390) global variable or its TFormatSettings (see page 4367) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloat raises an EConvertError (see page 3985) exception.

The first form of StrToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
```

```
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;

end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
```

```
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
{
This example uses a form with a combo box, two list boxes,
an edit control, and a button.  The user selects a
conversion family from the combo box, which causes the list
boxes to fill with the conversion types registered with that
family. When the user clicks the button, a message box
appears showing the value in the edit control converted from
the units selected in the first list box to the units
selected in the second list box.
Note:   If you have not registered your own conversion families,
include the stdconvs unit in your uses clause to use the
standard conversion families.
}

{
The following OnCreate event handler for the form initializes
the combo box and list boxes:
}

procedure TForm1.FormCreate(Sender: TObject);
var
  FamilyList: TConvFamilyArray;
  i: Integer;
begin
  GetConvFamilies(FamilyList);
  for i := 0 to (Length(FamilyList) - 1) do
    ComboBox1.Items.Add(ConvFamilyToDescription(FamilyList[i]));
  ComboBox1.ItemIndex := 0; { select the first item }
  ComboBox1Select(ComboBox1); {trigger the event to initialize lists}
end;

{
The following OnClick event handler for the button reads the
value entered in the edit box and converts it between the
measurement units selected in the list boxes:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  newVal: Double;
  CurFamily: TConvFamily;
  FromType, ToType: TConvType;
begin
  DescriptionToConvFamily(ComboBox1.Items[ComboBox1.ItemIndex], CurFamily);
  DescriptionToConvType(CurFamily, ListBox1.Items[ListBox1.ItemIndex], FromType);
  DescriptionToConvType(CurFamily, ListBox2.Items[ListBox2.ItemIndex], ToType);
```

```
  newVal := Convert(StrToFloat(Edit1.Text), FromType, ToType);
  ShowMessage(Format('%g %s', [newVal, ConvTypeToDescription(ToType)]));
end;

{
The following OnSelect event handler for the combo box
initializes the list boxes with the registered conversion
types for the selected family:
}
procedure TForm1.ComboBox1Select(Sender: TObject);
var
  TypeList: TConvTypeArray;
  i: Integer;
  CurFamily: TConvFamily;Description:string;
beginDescription := ComboBox1.Items[ComboBox1.ItemIndex];
  if DescriptionToConvFamily(Description, CurFamily) then
  begin
    GetConvTypes(CurFamily, TypeList);
    ListBox1.Items.Clear;
    ListBox2.Items.Clear;
    for i := 0 to Length(TypeList) -1 do
      ListBox1.Items.Add(ConvTypeToDescription(TypeList[i]));
    ListBox2.Items := ListBox1.Items;
    ListBox1.ItemIndex := 0;
    ListBox2.ItemIndex := 0;
  end;
end;
```

39.3.233 SysUtils.StrToFloatDef Function

Converts a given string to a floating-point value with error default.

Pascal

```
function StrToFloatDef(const S: string; const Default: Extended): Extended; overload;
function StrToFloatDef(const S: string; const Default: Extended; const FormatSettings:
TFormatSettings): Extended; overload;
```

C++

```
Extended StrToFloatDef(const UnicodeString S, const Extended Default);
Extended StrToFloatDef(const UnicodeString S, const Extended Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use StrToFloatDef to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4390) global variable or its TFormatSettings (see page 4367) equivalent defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloatDef returns Default.

The first form of StrToFloatDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloatDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloatDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.234 SysUtils.StrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt(const S: string): Integer; overload;
```

C++

```
int StrToInt(const UnicodeString S);
```

File

SysUtils

Description

StrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt raises an EConvertError (see page 3985) exception.

C++ Examples:

```
/*
This example uses two edit boxes and a button on a form. When the user clicks
the button, the code converts the values in the edit boxes to numbers, adds them,
and displays a message indicating the sum:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int i1, i2;
  i1 = StrToInt(Edit1->Text);
  i2 = StrToInt(Edit2->Text);
  ShowMessage(IntToStr(i1 + i2));
}
```

Delphi Examples:

```
{
This example uses two edit boxes and a button on a form. When the user clicks
the button, the code converts the values in the edit boxes to numbers, adds them,
and displays a message indicating the sum:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  J: Integer;
begin
  I := StrToInt(Edit1.Text);
  J := StrToInt(Edit2.Text);
  ShowMessage(IntToStr(I + J));
end;
```

See Also

IntToHex (see page 4259)

IntToStr (see page 4260)

StrToIntDef (see page 4330)

TryStrToInt (see page 4349)

39.3.235 SysUtils.StrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64(const S: string): Int64; overload;
```

C++

```
Int64 StrToInt64(const UnicodeString S);
```

File

SysUtils

Description

StrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64 raises an EConvertError (see page 3985) exception.

See Also

IntToHex (see page 4259)

IntToStr (see page 4260)

StrToIntDef (see page 4330)

StrToInt64Def (see page 4329)

StrToInt (see page 4328)

TryStrToInt (see page 4349)

39.3.236 SysUtils.StrToInt64Def Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64Def(const S: string; const Default: Int64): Int64; overload;
```

C++

```
Int64 StrToInt64Def(const UnicodeString S, const Int64 Default);
```

File

SysUtils

Description

StrToInt64Def converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64Def returns the number passed in Default.

See Also

IntToStr (see page 4260)

StrToInt (see page 4328)

StrToInt64 (see page 4329)

StrToInt (see page 4328)

TryStrToInt64 (see page 4350)

39.3.237 SysUtils.StrToIntDef Function

Converts a string that represents an integer (decimal or hex notation) to a number with error default.

Pascal

```
function StrToIntDef(const S: string; Default: Integer): Integer; overload;
```

C++

```
int StrToIntDef(const UnicodeString S, int Default);
```

File

SysUtils

Description

StrToIntDef converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToIntDef returns Default.

C++ Examples:

```
/*
This example uses two edit boxes and a button on a form. The
user enters a number in the first edit box and clicks the
button. If the number entered was a valid integer, the same
value appears in the second edit box. If the number was not
a valid integer, the default value of 1000 appears in the
second edit box:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Number = StrToIntDef(Edit1->Text, 1000);
  Edit2->Text = IntToStr(Number);
}
```

Delphi Examples:

```
{
  This example uses the FetchParams method to sync the params
  with the server.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
```

```
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
```

```
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    SavedParams.Free;
  end;
  {
  This example uses two edit boxes and a button on a form. The
  user enters a number in the first edit box and clicks the
  button. If the number entered was a valid integer, the same
  value appears in the second edit box. If the number was not
  a valid integer, the default value of 1000 appears in the
  second edit box:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Number: Integer;
  begin
    Number := StrToIntDef(Edit1.Text, 1000);
    Edit2.Text := IntToStr(Number);
  end;
```

See Also

IntToHex (see page 4259)

IntToStr (see page 4260)

StrToInt (see page 4328)

TryStrToInt (see page 4349)

39.3.238 SysUtils.StrToTime Function

Converts a string to a TDateTime (see page 3335) value.

Pascal

```
function StrToTime(const S: string): TDateTime; overload;
function StrToTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToTime(const UnicodeString S);
TDateTime StrToTime(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToTime to parse a string that specifies a time value. If S does not contain a valid time, StrToTime raises an EConvertError (see page 3985) exception.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4403) global variable or its TFormatSettings (see page 4367) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a time in the edit box in the HH:MM:SS
format, the string entered is converted to a TDateTime value.
A message then appears that says "Good Morning" or "Good
Afternoon", depending on whether the time entered was in the
morning or afternoon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dtTime = StrToTime(Edit1->Text);
  if (dtTime < (TDateTime) 0.50 )
    ShowMessage("Good Morning");
  else
    ShowMessage("Good Afternoon");
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a time in the edit box in the HH:MM:SS
format, the string entered is converted to a TDateTime value.
A message then appears that says "Good Morning" or "Good
Afternoon", depending on whether the time entered was in the
morning or afternoon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ATime: TDateTime;
begin
  ATime := StrToTime(Edit1.Text);
  if ATime < 0.50 then
    ShowMessage('Good Morning')
  else
    ShowMessage('Good Afternoon');
end;
```

39.3.239 SysUtils.StrToTimeDef Function

Converts a string to a TDateTime (see page 3335) value with error default.

Pascal

```
function StrToTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToTimeDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToTimeDef(const UnicodeString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToTimeDef to parse a string that specifies a time value. If S does not contain a valid time, StrToTimeDef returns Default.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4403) global variable or its TFormatSettings (see page 4367) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.240 SysUtils.StrUpper Function

Returns a string in upper case.

Pascal

```
function StrUpper(Str: PAnsiChar): PAnsiChar; overload;
function StrUpper(Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrUpper(PAnsiChar Str);
PWideChar StrUpper(PWideChar Str);
```

File

SysUtils

Description

StrUpper converts Str (see page 3764) to uppercase and returns Str (see page 3764).

Note: If the source string contains international characters, use AnsiStrUpper (see page 4125) instead. If the source string is an AnsiString (long string), use UpperCase (see page 4352) (or AnsiUpperCase (see page 4126)) instead.

C++ Examples:

```
/*
The following example uses an edit box, two labels, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in label1's
caption and upper case in label2's caption.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = StrLower(Edit1->Text.c_str());
  Label2->Caption = StrUpper(Edit1->Text.c_str());
}
```

Delphi Examples:

```
{
The following example uses an edit box, a label, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in the label's
```

```
  caption.
  }
  var
    S: array[0..20] of Char = 'A fUnNy StRiNg';

  procedure TForm1.Button1Click(Sender: TObject);
  var
    lower, upper : string;
  begin
    lower := string(StrLower(S));
    upper := string(StrUpper(S));
    Canvas.TextOut(5, 10, lower + ' ' + upper);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Label1.Caption := S;
  end;
```

See Also

AnsiUpperCase (see page 4126)

UpperCase (see page 4352)

StrLower (see page 4307)

39.3.241 SysUtils.Supports Function

Indicates whether a given object or interface supports a specified interface.

Pascal

```
function Supports(const Instance: IInterface; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: IInterface; const IID: TGUID): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID): Boolean; overload;
function Supports(const AClass: TClass; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean Supports(const IInterface Instance, const TGUID IID,  Intf);
Boolean Supports(const TObject * Instance, const TGUID IID,  Intf);
Boolean Supports(const IInterface Instance, const TGUID IID);
Boolean Supports(const TObject * Instance, const TGUID IID);
Boolean Supports(const TClass AClass, const TGUID IID);
```

File

SysUtils

Description

Call Supports to determine whether the object or interface specified by Instance, or the class specified by AClass, supports the interface identified by the IID parameter. If Instance supports the interface, Supports returns the interface as the Intf parameter and returns true. If AClass supports the interface, Supports does not return an interface, but still returns true. If the interface specified by IID is not supported, Supports returns false.

On the Win32 platform, to determine whether the interface specified by IID is supported, Supports calls the QueryInterface method of the supplied interface (or the interface of the supplied object). However, unlike calling QueryInterface directly, Supports lets you pass a nil (Delphi) or NULL (C++) value as the instance.

On the .NET platform, Supports uses the typecasting facilities of the .NET framework.

See Also

TComponent

39.3.242 SysUtils.SysErrorMessage Function

Converts OS error codes into strings.

Pascal

```
function SysErrorMessage(ErrorCode: Cardinal): string;
```

C++

```
UnicodeString SysErrorMessage(unsigned ErrorCode);
```

File

SysUtils

Description

SysErrorMessage returns an error message string that corresponds to the specified OS error code.

See Also

RaiseLastOSError (see page 4274)

EWin32Error (see page 4015)

39.3.243 SysUtils.SystemTimeToDateTime Function

Converts a system time value into a TDateTime (see page 3335) value.

Pascal

```
function SystemTimeToDateTime(const SystemTime: TSystemTime): TDateTime;
```

C++

```
TDateTime SystemTimeToDateTime(const TSystemTime SystemTime);
```

File

SysUtils

Description

Call SystemTimeToDateTime to convert a value from Windows 32's TSystemTime (Delphi) or _SYSTEMTIME (C++) API format to a TDateTime (see page 3335) value. SystemTimeToDateTime is available on Windows only.

C++ Examples:

```
/*
The following code retrieves the current date and time using
the Windows GetLocalTime API call, and converts this value
to a TDateTime value.
*/
TDateTime __fastcall GetCurrentDateTime()
{
  SYSTEMTIME SystemTime;
  GetLocalTime(&SystemTime);
```

```
  return SystemTimeToDateTime(SystemTime);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(GetCurrentDateTime());
}
```

Delphi Examples:

```
{
The following code retrieves the current date and time using
the Windows GetLocalTime API call, and converts this value
to a TDateTime value.
}
function GetCurrentDateTime: TDateTime;
var
  SystemTime: TSystemTime;
begin
  GetLocalTime(SystemTime);
  Result := SysUtils.SystemTimeToDateTime(SystemTime);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(GetCurrentDateTime);
end;
```

See Also

DateTimeToSystemTime (see page 4155)

39.3.244 SysUtils.TextPos Function

Pascal

```
function TextPos(Str: PAnsiChar; SubStr: PAnsiChar): PAnsiChar; overload;
```

C++

```
PAnsiChar TextPos(PAnsiChar Str, PAnsiChar SubStr);
```

File

SysUtils

Description

This is function SysUtils.TextPos.

39.3.245 SysUtils.TextPos Function

Pascal

```
function TextPos(Str: PWideChar; SubStr: PWideChar): PWideChar; overload;
```

C++

```
PWideChar TextPos(PWideChar Str, PWideChar SubStr);
```

File

SysUtils

Description

This is function SysUtils.TextPos.

39.3.246 SysUtils.TextToFloat Function

Converts a null-terminated string to a floating-point value.

Pascal

```
function TextToFloat(Buffer: PAnsiChar; var Value; ValueType: TFloatValue): Boolean; overload;
function TextToFloat(Buffer: PAnsiChar; var Value; ValueType: TFloatValue; const
FormatSettings: TFormatSettings): Boolean; overload;
function TextToFloat(Buffer: PWideChar; var Value; ValueType: TFloatValue): Boolean; overload;
function TextToFloat(Buffer: PWideChar; var Value; ValueType: TFloatValue; const
FormatSettings: TFormatSettings): Boolean; overload;
```

C++

```
Boolean TextToFloat(PAnsiChar Buffer,  Value, TFloatValue ValueType);
Boolean TextToFloat(PAnsiChar Buffer,  Value, TFloatValue ValueType, const TFormatSettings
FormatSettings);
Boolean TextToFloat(PWideChar Buffer,  Value, TFloatValue ValueType);
Boolean TextToFloat(PWideChar Buffer,  Value, TFloatValue ValueType, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use TextToFloat to convert a null-terminated string, Buffer, to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The converted value is assigned to the Value parameter, which must be of type Extended or Currency (see page 3242), as indicated by the ValueType parameter.

TextToFloat returns true if the conversion is successful, or false if the Buffer does not contain a valid floating-point value.

The first form of TextToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of TextToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TextToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.247 SysUtils.Time Function

Returns the current time.

Pascal

```
function Time: TDateTime;
```

C++

```
TDateTime Time();
```

File

SysUtils

Description

Time and GetTime (see page 4253) return the current time as a TDateTime (see page 3335) value. The two functions are completely equivalent.

Note: Time clashes with an Xlib function of the same name. Use GetTime (see page 4253) in Linux and cross-platform code.

C++ Examples:

```
/*
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Caption = str;  // display the time on the form's caption
  // Note This could have been done with the following line of code:
  //  Caption = TimeToStr(Time());
}
```

Delphi Examples:

```
{
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
}
procedure TForm1.Timer1Timer(Sender: TObject);
var
  DateTime : TDateTime;
  str : string;
begin
  DateTime := Time;  // store the current date and time
  str := TimeToStr(DateTime); // convert the time into a string
  Caption := str;  // display the time on the form's caption
  { Note This could have been done with the following line of code:
    Caption := TimeToStr(Time); }
end;
```

See Also

Date (see page 4151)

DecodeTime (see page 4161)

Now (see page 4271)

TimeToStr (see page 4340)

39.3.248 SysUtils.TimeStampToDateTime Function

Converts a TTimeStamp (see page 4375) value into the corresponding TDateTime (see page 3335) value.

Pascal

```
function TimeStampToDateTime(const TimeStamp: TTimeStamp): TDateTime;
```

C++

```
TDateTime TimeStampToDateTime(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToDateTime to convert a TTimeStamp (see page 4375) value into a TDateTime (see page 3335) value. TTimeStamp (see page 4375) values represent time as separate date and time values, where the date is the number of calendar days since the start of the current calendar (that is, January 1, 0001 would have a value of 1), and time is the number of milliseconds since midnight. TDateTime (see page 3335) values represent time as the number of days (including fractional days) that have elapsed since 12:00 am on December 30, 1899.

See Also

DateTimeToTimeStamp (see page 4156)

39.3.249 SysUtils.TimeStampToMSecs Function

Converts a TTimeStamp (see page 4375) value into an absolute number of milliseconds.

Pascal

```
function TimeStampToMSecs(const TimeStamp: TTimeStamp): Comp;
```

C++

```
Comp TimeStampToMSecs(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToMSecs to convert a TTimeStamp (see page 4375) value, into the total number of milliseconds that comprise the Date (see page 4151) field plus the number of milliseconds given in the Time (see page 4338) field.

See Also

MSecsToTimeStamp (see page 4270)

TimeStampToDateTime (see page 4339)

39.3.250 SysUtils.TimeToStr Function

Returns a string that represents a TDateTime (see page 3335) value.

Pascal

```
function TimeToStr(const DateTime: TDateTime): string; overload;
function TimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
UnicodeString TimeToStr(const TDateTime DateTime);
UnicodeString TimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

TimeToStr converts the Time (see page 4338) parameter, a TDateTime (see page 3335) value, to a string. The conversion uses the format specified by the LongTimeFormat (see page 4396) global variable or its TFormatSettings (see page 4367) equivalent. Change the format of the string by changing the values of some of the date and time variables.

The first form of TimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of TimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
/*
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Caption = str;  // display the time on the form's caption
  // Note This could have been done with the following line of code:
  //  Caption = TimeToStr(Time());
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
{
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
}
procedure TForm1.Timer1Timer(Sender: TObject);
var
  DateTime : TDateTime;
  str : string;
begin
  DateTime := Time;  // store the current date and time
  str := TimeToStr(DateTime); // convert the time into a string
  Caption := str;  // display the time on the form's caption
  { Note This could have been done with the following line of code:
    Caption := TimeToStr(Time); }
end;
```

See Also

DateTimeToStr (see page 4153)

DateToStr (see page 4156)

StrToDateTime (see page 4322)

Time (see page 4338)

GetLocaleFormatSettings (see page 4250)

39.3.251 SysUtils.Trim Function

Trims leading and trailing spaces and control characters from a string.

Pascal

```
function Trim(const S: string): string; overload;
```

C++

```
UnicodeString Trim(const UnicodeString S);
```

File

SysUtils

Description

Trim removes leading and trailing spaces and control characters from the given string S.

In the 7-bit ASCII character set defined in ANSI X3.4-1977 (C0 and G0), "control codes" are defined as all characters whose code is between 0 and 31.

See Also

TrimLeft (see page 4343)

TrimRight (see page 4343)

39.3.252 SysUtils.TrimLeft Function

Trims leading spaces and control characters from a string.

Pascal

```
function TrimLeft(const S: string): string; overload;
```

C++

```
UnicodeString TrimLeft(const UnicodeString S);
```

File

SysUtils

Description

TrimLeft returns a copy of the string S with leading spaces and control characters removed.

See Also

Trim (see page 4342)

TrimRight (see page 4343)

39.3.253 SysUtils.TrimRight Function

Trims trailing spaces and control characters from a string.

Pascal

```
function TrimRight(const S: string): string; overload;
```

C++

```
UnicodeString TrimRight(const UnicodeString S);
```

File

SysUtils

Description

TrimRight returns a copy of the string S with trailing spaces and control characters removed.

See Also

TrimLeft (see page 4343)

Trim (see page 4342)

39.3.254 SysUtils.TryEncodeDate Function

Creates a TDateTime (see page 3335) value that represents a specified Year, Month, and Day.

Pascal

```
function TryEncodeDate(Year: Word; Month: Word; Day: Word; out Date: TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDate(Word Year, Word Month, Word Day, TDateTime Date);
```

File

SysUtils

Description

TryEncodeDate generates a TDateTime (see page 3335) value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

The Date (see page 4151) parameter returns the generated value if TryEncodeDate returns true.

If the specified values are not within range, TryEncodeDate returns false.

See Also

DateToStr (see page 4156)

DecodeDate (see page 4159)

TryEncodeTime (see page 4344)

TryEncodeDateTime

EncodeDate (see page 4169)

39.3.255 SysUtils.TryEncodeTime Function

Creates a TDateTime (see page 3335) value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function TryEncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word; out Time: TDateTime):
Boolean;
```

C++

```
Boolean TryEncodeTime(Word Hour, Word Min, Word Sec, Word MSec, TDateTime Time);
```

File

SysUtils

Description

EncodeTime (see page 4170) encodes the given hour, minute, second, and millisecond into a TDateTime (see page 3335) value.

Valid Hour values are 0 through 24. If Hour is 24, Min, Sec, and MSec must all be 0, and the resulting TDateTime (see page 3335) value represents midnight (12:00:00:000 AM) of the following day.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

The Time (see page 4338) paremter returns the generated TDateTime (see page 3335) value of the specifies values are in range. This value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

See Also

DecodeTime (see page 4161)

TryEncodeDate (see page 4343)

TryEncodeDateTime

EncodeTime (see page 4170)

39.3.256 SysUtils.TryFloatToCurr Function

Converts a floating-point value to a Currency (see page 3242) value.

Pascal

```
function TryFloatToCurr(const Value: Extended; out AResult: Currency): Boolean;
```

C++

```
Boolean TryFloatToCurr(const Extended Value, Currency AResult);
```

File

SysUtils

Description

TryFloatToCurr converts the specified floating-point value into a Currency (see page 3242) value.

Value is the floating-point value to convert.

AResult returns the equivalent Currency (see page 3242) value, if possible.

If TryFloatToCurr is successful, it returns true. If Value is not within the valid range for a Currency (see page 3242) value, TryFloatToCurr returns false.

See Also

FloatToCurr (see page 4223)

39.3.257 SysUtils.TryFloatToDateTime Function

Converts a floating-point value to a TDateTime (see page 3335) value.

Pascal

```
function TryFloatToDateTime(const Value: Extended; out AResult: TDateTime): Boolean;
```

C++

```
Boolean TryFloatToDateTime(const Extended Value, TDateTime AResult);
```

File

SysUtils

Description

TryFloatToDateTime converts a floating-point value into the corresponding TDateTime (see page 3335) value.

Value is the floating-point value to convert.

AResult returns the corresponding TDateTime (see page 3335) value if possible.

TryFloatToDateTime returns true if the conversion was successful, false otherwise.

See Also

MinDateTime (see page 4423)

FloatToDateTime (see page 4223)

39.3.258 SysUtils.TryStrToBool Function

Converts a string to a Boolean value, with a Boolean success code.

Pascal

```
function TryStrToBool(const S: string; out Value: Boolean): Boolean; overload;
```

C++

```
Boolean TryStrToBool(const UnicodeString S, Boolean Value);
```

File

SysUtils

Description

TryStrToBool converts the string specified by S to a Boolean and sets Value to the result. If S can be converted to a number, TryStrToBool sets Value to false if that number is 0, true otherwise. If S can't be converted to a number, TryStrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 4403) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4392) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4403) or FalseBoolStrs (see page 4392), TryStrToBool returns false to indicate that the conversion failed. If the conversion is successful, TryStrToBool returns true.

Note: If you do not assign a value to TrueBoolStrs (see page 4403), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 4392), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4403) and FalseBoolStrs (see page 4392) variables are undefined when an application starts. Any use of StrToBool (see page 4318), BoolToStr (see page 4130) or TryStrToBool will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4130)

TrueBoolStrs (see page 4403)

FalseBoolStrs (see page 4392)

StrToBool (see page 4318)

39.3.259 SysUtils.TryStrToCurr Function

Converts a string to a Currency (see page 3242) value, with a Boolean success code.

Pascal

```
function TryStrToCurr(const S: string; out Value: Currency): Boolean; overload;
function TryStrToCurr(const S: string; out Value: Currency; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToCurr(const UnicodeString S, Currency Value);
Boolean TryStrToCurr(const UnicodeString S, Currency Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToCurr to convert a string that represents a floating-point value to the corresponding Currency (see page 3242) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4390) global variable or its TFormatSettings (see page 4367) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, TryStrToCurr returns Default.

The first form of TryStrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.260 SysUtils.TryStrToDate Function

Converts a string to a TDateTime (see page 3335) value, with a Boolean success code.

Pascal

```
function TryStrToDate(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDate(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDate(const UnicodeString S, TDateTime Value);
Boolean TryStrToDate(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDate to parse a string that specifies a date. If S does not contain a valid date, TryStrToDate returns false.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4389) global

variable or its TFormatSettings (see page 4367) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4398) global variable or its TFormatSettings (see page 4367) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of TryStrToDate is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.261 SysUtils.TryStrToDateTime Function

Converts a string to a TDateTime (see page 3335) value with a Boolean success code.

Pascal

```
function TryStrToDateTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDateTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDateTime(const UnicodeString S, TDateTime Value);
Boolean TryStrToDateTime(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, TryStrToDateTime returns false.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4404). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3242) and Date (see page 4151)-Time (see page 4338) Formatting Variables" for more information.

The first form of TryStrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.262 SysUtils.TryStrToFloat Function

Converts a given string to a floating-point value with a boolean success code.

Pascal

```
function TryStrToFloat(const S: string; out Value: Extended): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Extended; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToFloat(const UnicodeString S, Extended Value);
Boolean TryStrToFloat(const UnicodeString S, Extended Value, const TFormatSettings
FormatSettings);
Boolean TryStrToFloat(const UnicodeString S, Double Value);
Boolean TryStrToFloat(const UnicodeString S, Double Value, const TFormatSettings
FormatSettings);
Boolean TryStrToFloat(const UnicodeString S, Single Value);
Boolean TryStrToFloat(const UnicodeString S, Single Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use TryStrToFloat to convert a astring, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4390) global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, TryStrToFloat returns Default.

TryStrToFloat places the result in Value, and returns true if the conversion was successful, false otherwise.

The first three forms of TryStrToFloat are not thread-safe, because they use localization information contained in global variables. The second three forms of TryStrToFloat, which are thread-safe, refer to localization information contained in the FormatSettings parameter. Before calling the thread-safe forms of TryStrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.263 SysUtils.TryStrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.

Pascal

```
function TryStrToInt(const S: string; out Value: Integer): Boolean; overload;
```

C++

```
Boolean TryStrToInt(const UnicodeString S, int Value);
```

File

SysUtils

Description

TryStrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt returns false; otherwise TryStrToInt returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToHex (see page 4259)

IntToStr (see page 4260)

StrToInt (see page 4328)

StrToIntDef (see page 4330)

39.3.264 SysUtils.TryStrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function TryStrToInt64(const S: string; out Value: Int64): Boolean; overload;
```

C++

```
Boolean TryStrToInt64(const UnicodeString S, Int64 Value);
```

File

SysUtils

Description

TryStrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt64 returns false; otherwise TryStrToInt64 returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToStr (see page 4260)

StrToInt (see page 4328)

StrToInt64 (see page 4329)

StrToInt (see page 4328)

39.3.265 SysUtils.TryStrToTime Function

Converts a string to a TDateTime (see page 3335) value with an error default.

Pascal

```
function TryStrToTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToTime(const UnicodeString S, TDateTime Value);
Boolean TryStrToTime(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToTime to parse a string that specifies a time value. If S does not contain a valid time, TryStrToTime returns false.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4403) global variable or its TFormatSettings (see page 4367) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of TryStrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.266 SysUtils.UIntToStr Function

Pascal

```
function UIntToStr(Value: Cardinal): string; overload;
```

C++

```
UnicodeString UIntToStr(unsigned Value);
```

File

SysUtils

Description

This is function SysUtils.UIntToStr.

39.3.267 SysUtils.UIntToStr Function

Pascal

```
function UIntToStr(Value: UInt64): string; overload;
```

C++

```
UnicodeString UIntToStr(UInt64 Value);
```

File

SysUtils

Description

This is function SysUtils.UIntToStr.

39.3.268 SysUtils.UnloadPackage Function

Unloads a package.

Pascal

```
procedure UnloadPackage(Module: HMODULE);
```

C++

```
UnloadPackage(HMODULE Module);
```

File

SysUtils

Description

Call UnloadPackage to unload a package that was loaded using the LoadPackage (see page 4268) function. UnloadPackage performs any finalization of units contained in the package and then unloads the package BPL.

See Also

LoadPackage (see page 4268)

39.3.269 SysUtils.UpperCase Function

Returns a copy of a string in uppercase.

Pascal

```
function UpperCase(const S: string): string; overload;
function UpperCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
UnicodeString UpperCase(const UnicodeString S);
UnicodeString UpperCase(const UnicodeString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

UpperCase returns a copy of the string S, with the same text but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase. To convert 8-bit international characters, use AnsiUpperCase (see page 4126) instead.

C++ Examples:

```
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

AnsiUpperCase (see page 4126)

LowerCase (see page 4269)

39.3.270 SysUtils.WideBytesOf Function

Pascal

```
function WideBytesOf(const Value: UnicodeString): TBytes;
```

C++

```
TBytes WideBytesOf(const UnicodeString Value);
```

File

SysUtils

Description

This is function SysUtils.WideBytesOf.

39.3.271 SysUtils.WideCompareStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideCompareStr(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. with S1 set to 'a' and S2 set to 'A', WideCompareStr returns a value lsess than zero, while CompareStr (see page 4142) returns a value greater than zero.

See Also

WideCompareText (see page 4355)

WideSameStr (see page 4357)

39.3.272 SysUtils.WideCompareText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideCompareText(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. WideCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

WideCompareStr (see page 4354)

WideSameText (see page 4358)

39.3.273 SysUtils.WideFmtStr Function

Assembles a formatted Unicode string using a format string and an array of arguments.

Pascal

```
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const); overload;
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const; const FormatSettings: TFormatSettings); overload;
```

C++

```
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args);
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args into a Unicode string. Formatting is controlled by the format string Format (see page 4232); the results are returned in the parameter Result.

For information on the format strings, see Format (see page 4232) Strings.

The first form of WideFmtStr is not thread-safe, because it uses localization information contained in global variables. The second form of WideFmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

GetLocaleFormatSettings (see page 4250)

39.3.274 SysUtils.WideFormat Function

Returns a formatted Unicode string assembled from a format string and an array of arguments.

Pascal

```
function WideFormat(const Format: WideString; const Args: array of const): WideString;
overload;
function WideFormat(const Format: WideString; const Args: array of const; const
FormatSettings: TFormatSettings): WideString; overload;
```

C++

```
BSTR WideFormat(const BSTR Format, const array of const Args);
BSTR WideFormat(const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the format string Format (see page 4232); the results are returned in the function result as a Unicode string.

For information on the format strings, see Format (see page 4232) Strings.

The first form of WideFormat is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

See Also

GetLocaleFormatSettings (see page 4250)

39.3.275 SysUtils.WideFormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const): Cardinal; overload;
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
```

C++

```
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args);
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Unicode format string Format (see page 4232) (whose length is given by FmtLen); the results are returned in Buffer (whose length is given by BufLen). The function returns the number of Unicode characters in the formatted string.

Note: Both buffers contain wide (Unicode) characters. BufLen and FmtLen must specify the number of Unicode characters in their respective buffers, not the number of bytes.

For information on the format strings, see Format (see page 4232) Strings.

The first form of WideFormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4250).

39.3.276 SysUtils.WideLowerCase Function

Returns Unicode string converted to lower case.

Pascal

```
function WideLowerCase(const S: WideString): WideString;
```

C++

```
BSTR WideLowerCase(const BSTR S);
```

File

SysUtils

Description

WideLowerCase returns S converted to lower case. The conversion uses the current locale.

See Also

WideUpperCase (see page 4359)

LowerCase (see page 4269)

39.3.277 SysUtils.WideSameStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideSameStr(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, Ffalse otherwise.

See Also

WideSameText (see page 4358)

WideCompareStr (see page 4354)

39.3.278 SysUtils.WideSameText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideSameText(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

WideSameStr (see page 4357)

WideCompareText (see page 4355)

39.3.279 SysUtils.WideStrAlloc Function

Pascal

```
function WideStrAlloc(Size: Cardinal): PWideChar;
```

C++

```
PWideChar WideStrAlloc(unsigned Size);
```

File

SysUtils

Description

This is function SysUtils.WideStrAlloc.

39.3.280 SysUtils.WideStringOf Function

Pascal

```
function WideStringOf(const Value: TBytes): UnicodeString;
```

C++

```
UnicodeString WideStringOf(const TBytes Value);
```

File

SysUtils

Description

This is function SysUtils.WideStringOf.

39.3.281 SysUtils.WideUpperCase Function

Returns Unicode string converted to upper case.

Pascal

```
function WideUpperCase(const S: WideString): WideString;
```

C++

```
BSTR WideUpperCase(const BSTR S);
```

File

SysUtils

Description

WideUpperCase returns S converted to upper case. The conversion uses the current locale.

See Also

WideLowerCase (see page 4357)

UpperCase (see page 4352)

39.3.282 SysUtils.Win32Check Function

Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.

Pascal

```
function Win32Check(RetVal: BOOL): BOOL; platform;
```

C++

```
BOOL Win32Check(BOOL RetVal);
```

File

SysUtils

Description

Call Win32Check with the return value of a Windows API call that returns a Boolean to indicate success or failure. If the Windows API returns false (indicating failure), Win32Check calls RaiseLastOSError (see page 4274) to raise an exception. If the Windows function returns true, Win32Check returns true.

Win32Check is available on Windows only.

See Also

RaiseLastOSError (see page 4274)

39.3.283 SysUtils.WrapText Function

Splits a string into multiple lines as its length approaches a specified size.

Pascal

```
function WrapText(const Line: string; const BreakStr: string; const BreakChars: TSysCharSet;
MaxCol: Integer): string; overload;
function WrapText(const Line: string; MaxCol: Integer = 45): string; overload;
```

C++

```
UnicodeString WrapText(const UnicodeString Line, const UnicodeString BreakStr, const
TSysCharSet BreakChars, int MaxCol);
UnicodeString WrapText(const UnicodeString Line, int MaxCol = 45);
```

File

SysUtils

Description

WrapText scans a string for occurrences of any of the characters specified by nBreakChars and inserts a line-break, specified by BreakStr, at the last occurrence of a character in nBreakChars before MaxCol. Line is the text WrapText scans. MaxCol is the maximum line length.

If the BreakStr and nBreakChars parameters are omitted, WrapText searches for space, hyphen, or tab characters on which to break the line and inserts a carriage return/line feed pair (Windows) or a linefeed (Linux) at the break points.

WrapText does not insert a break into an embedded quoted string (both single quotes and double quotes are supported).

For example, the following call wraps the text into two lines at the last space character:

WrapText('The rain in Spain falls mainly on the plain.', #13#10, ['.',' ',#9,'-'], 42);

TSysCharSet (see page 4384) bchars

bchars << '.' << '-' << ' ';

WrapText("The rain in Spain falls mainly on the plain.", "\r\n", bchars, 42);

The result:

The rain in Spain falls mainly on the

plain.

39.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFilenameCaseMatch (see page 4364)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 4364)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatValue (see page 4366)	This is record SysUtils.TFloatValue.
	TLocaleOptions (see page 4369)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 4369)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 4370)	This is record SysUtils.TNameType.

Records

	Name	Description
	Int64Rec (see page 4361)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 4362)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TCharSearch (see page 4362)	This is record SysUtils.TCharSearch.
	TExceptionRecord (see page 4363)	TExceptionRecord holds information about an exception in an application.
	TFloatRec (see page 4366)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFormatSettings (see page 4367)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 4368)	TLangRec declares a record used to describe a Windows locale.
	TSearchRec (see page 4371)	TSearchRec defines file information searched for by FindFirst (see page 4218) or FindNext (see page 4221).
	TSysLocale (see page 4374)	TSysLocale represents locale information.
	TThreadInfo (see page 4375)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 4375)	TTimeStamp represents time and date values.
	WordRec (see page 4376)	WordRec declares a utility record to store high and low order bytes of a variable.

39.4.1 SysUtils.Int64Rec Record

Int64Rec declares a utility record to provide access to the bytes of an Int64 value.

Pascal

```
Int64Rec = packed record
  case Integer of
    0: (Hi: Cardinal;
        Lo: Cardinal;);
    1: (Cardinals: array [0..1] of Cardinal);
        Words: array [0..3] of Word);
        Bytes: array [0..7] of Byte););
end;
```

C++

```
struct Int64Rec {
  unsigned Hi;
  unsigned Lo;
  array [0..1] of Cardinal) Cardinals;
```

```
    array [0..3] of Word) Words;
    array [0..7] of Byte) Bytes;
  };
```

File

SysUtils

Description

The Int64Rec type declares a utility record that stores 8 contiguous (packed) bytes of data. Int64Rec is used primarily for typecasting, where Int64 data needs to be dissected.

The 8 bytes may be accessed individually via the `Bytes` array, or as an array of 4 `Words`, or an array of 2 `Cardinals`. And finally, the value may be accessed via the `Hi` and `Lo` order Cardinals.

39.4.2 SysUtils.LongRec Record

LongRec declares a utility record to provide access to the bytes of a LongWord value.

Pascal

```
LongRec = packed record
  case Integer of
    0: (Hi: Word;
        Lo: Word;);
    1: (Words: array [0..1] of Word);
        Bytes: array [0..3] of Byte););
end;
```

C++

```
struct LongRec {
  Word Hi;
  Word Lo;
  array [0..1] of Word) Words;
  array [0..3] of Byte) Bytes;
};
```

File

SysUtils

Description

The LongRec type declares a utility record that stores 4 contiguous (packed) bytes of data. LongRec is used primarily for typecasting, where LongWord data needs to be dissected.

The 4 bytes may be accessed individually via the `Bytes` array, or as an array of 2 `Words`, or the value may be accessed via the `Hi` and `Lo` order Words.

39.4.3 SysUtils.TCharSearch Record

Pascal

```
TCharSearch = record
  ArrayPtr: PChar;
  MatchPtr: PChar;
end;
```

C++

```
struct TCharSearch {
  const char * ArrayPtr;
  const char * MatchPtr;
};
```

File

SysUtils

Description

This is record SysUtils.TCharSearch.

39.4.4 SysUtils.TExceptionRecord Record

TExceptionRecord holds information about an exception in an application.

Pascal

```
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
  ExceptionAddress: Pointer;
  NumberParameters: Cardinal;
  case Boolean of
    True: (ExceptionInformation: array [0..14] of Longint);
          False: (ExceptAddr: Pointer; ExceptObject: Pointer;);
end;
```

C++

```
struct TExceptionRecord {
  unsigned ExceptionCode;
  unsigned ExceptionFlags;
  PExceptionRecord ExceptionRecord;
  void * ExceptionAddress;
  unsigned NumberParameters;
  array [0..14] of Longint) ExceptionInformation;
  (ExceptAddr: Pointer; ExceptObject: Pointer False;
};
```

File

SysUtils

Description

TExceptionRecord holds information about an exception in an application. It is furnished by the Delphi exception handling process.

The record comprises the following elements :

Element	Meaning
ExceptionCode	The exception code, such as EXCEPTION_INT_OVERFLOW.
ExceptionFlags	Comprises one flag : EXCEPTION_NONCONTINUABLE_EXCEPTION. If 0, then the exception is allowed to continue.
ExceptionRecord	Pointer to subsequent exception records, if any.
ExceptionAddress	Address where the exception occured.

`NumberParameters`	Number of elements in the `ExceptionInformation` array.
`ExceptionInformation`	An array of numbers that describe the exception.

39.4.5 SysUtils.TFilenameCaseMatch Enumeration

TFilenameCaseMatch indicates how close a match was found to a specified file name.

Pascal

```
TFilenameCaseMatch = (
  mkNone,
  mkExactMatch,
  mkSingleMatch,
  mkAmbiguous
);
```

C++

```
enum TFilenameCaseMatch {
  mkNone,
  mkExactMatch,
  mkSingleMatch,
  mkAmbiguous
};
```

File

SysUtils

Description

These are the possible values of the TFilenameCaseMatch type:

Value	Meaning
mkNone	No actual file matches the file name.
mkExactMatch	A single file matches the file name.
mkSingleMatch	No file matches the file name, but a single match can be found by ignoring differences in case.
mkAmbiguous	No file matches the file name, but two or more matches can be found by ignoring differences in case.

39.4.6 SysUtils.TFloatFormat Enumeration

TFloatFormat is an enumerated list of formatting codes for float functions.

Pascal

```
TFloatFormat = (
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
);
```

C++

```
enum TFloatFormat {
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
};
```

File

SysUtils

Description

TFloatFormat defines an enumerated list of formatting codes for the float functions.

Format (see page 4232)	Defines
ffGeneral	General number format. The value is converted to the shortest possible decimal string using scientific format. Trailing zeros are removed from the resulting string.
ffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits before the exponent in the resulting string (including the one before the decimal point) is given by the Precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The Digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
ffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffFixed with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The ffNumber format corresponds to the ffFixed format, except that the resulting string contains thousandth separators. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffNumber with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffCurrency	Currency (see page 3242) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 4389), CurrencyFormat (see page 4388), NegCurrFormat (see page 4396), ThousandSeparator (see page 4402), and DecimalSeparator (see page 4390) global variables, all of which are initialized from the Currency (see page 3242) Format (see page 4232) in the International section of the Windows Control Panel. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffCurrency with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.

See Also

CurrencyAndDatetimeFormatting

FloatToDecimal

FloatToStr

FloatToStrF

FloatToText

FloatToTextFmt

39.4.7 SysUtils.TFloatRec Record

TFloatRec declares a utility record that provides access to the attributes of a floating point number.

Pascal

```
TFloatRec = packed record
  Exponent: Smallint;
  Negative: Boolean;
  Digits: array[0..20] of AnsiChar;
end;
```

C++

```
struct TFloatRec {
  Smallint Exponent;
  Boolean Negative;
  array[0..20] of AnsiChar Digits;
};
```

File

SysUtils

Description

The TFloatRec type declares a record that holds the attributes of a floating point number. The record makes the attributes accessible for easy formatting or specialised numerical processing.

`Exponent` holds the floating point number exponent value.

`Negative` is set true if the floating point number has a negative value.

`Digits` is a 21 element array of numeric characters representing the floating point number mantissa value : its significant digits.

TFloatRec is used by the FloatToDecimal (see page 4224) function to hold the results of the conversion of a floating point value to its decimal representation.

See Also

FloatToDecimal (see page 4224)

39.4.8 SysUtils.TFloatValue Enumeration

Pascal

```
TFloatValue = (
  fvExtended,
  fvCurrency
);
```

C++

```
enum TFloatValue {
  fvExtended,
  fvCurrency
};
```

File

SysUtils

Description

This is record SysUtils.TFloatValue.

39.4.9 SysUtils.TFormatSettings Record

TFormatSettings defines a thread-safe string formatting context.

Pascal

```
TFormatSettings = record
  CurrencyFormat: Byte;
  NegCurrFormat: Byte;
  ThousandSeparator: Char;
  DecimalSeparator: Char;
  CurrencyDecimals: Byte;
  DateSeparator: Char;
  TimeSeparator: Char;
  ListSeparator: Char;
  CurrencyString: string;
  ShortDateFormat: string;
  LongDateFormat: string;
  TimeAMString: string;
  TimePMString: string;
  ShortTimeFormat: string;
  LongTimeFormat: string;
  ShortMonthNames: array[1..12] of string;
  LongMonthNames: array[1..12] of string;
  ShortDayNames: array[1..7] of string;
  LongDayNames: array[1..7] of string;
  TwoDigitYearCenturyWindow: Word;
end;
```

C++

```
struct TFormatSettings {
  Byte CurrencyFormat;
  Byte NegCurrFormat;
  Char ThousandSeparator;
  Char DecimalSeparator;
  Byte CurrencyDecimals;
  Char DateSeparator;
  Char TimeSeparator;
  Char ListSeparator;
  UnicodeString CurrencyString;
  UnicodeString ShortDateFormat;
  UnicodeString LongDateFormat;
  UnicodeString TimeAMString;
  UnicodeString TimePMString;
  UnicodeString ShortTimeFormat;
  UnicodeString LongTimeFormat;
  array[1..12] of string ShortMonthNames;
  array[1..12] of string LongMonthNames;
  array[1..7] of string ShortDayNames;
  array[1..7] of string LongDayNames;
  Word TwoDigitYearCenturyWindow;
};
```

File

SysUtils

Description

TFormatSettings defines a data structure containing locale information used by string formatting routine. Each member of TFormatSettings is equivalent to the global variable with the same name. A variable of type TFormatSettings defines a thread-safe context that formatting functions can use in place of the default global context, which is not thread-safe.

To create and use the thread-safe environment defined by TFormatSettings, follow these steps:

Define a variable of type TFormatSettings

Call GetLocaleFormatSettings (see page 4250) to populate the TFormatSettings variable with locale information.

Pass the TFormatSettings variable as the last parameter of the string formatting routine.

Each routine that accepts a TFormatSettings parameter is thread-safe, and is an overload of an equivalent function that refers to the global formatting variables.

39.4.10 SysUtils.TLangRec Record

TLangRec declares a record used to describe a Windows locale.

Pascal

```
TLangRec = packed record
  FName: string;
  FLCID: LCID;
  FExt: string;
end;
```

C++

```
struct TLangRec {
  UnicodeString FName;
  LCID FLCID;
  UnicodeString FExt;
};
```

File

SysUtils

Description

The TLangRec type declares a record that holds the attributes of a Windows language. The record is principally used internally by TLanguages (see page 4051).

`FName` holds the Windows locale name.

`FLCID` holds the local id for the Windows locale.

`FExt` holds the recommended filename extension for the Windows locale.

See Also

TLanguages (see page 4051)

39.4.11 SysUtils.TLocaleOptions Enumeration

TLocaleOptions defines a choice of dependent and independent locale options.

Pascal

```
TLocaleOptions = (
  loInvariantLocale,
  loUserLocale
);
```

C++

```
enum TLocaleOptions {
  loInvariantLocale,
  loUserLocale
};
```

File

SysUtils

Description

The TLocaleOptions type is an enumeration of two choices for dealing with the operating locale of an application. This Microsoft Windows specific type is used in a number of string handling routines, where the locale language may have a bearing on the routine.

Format (see page 4232)	Defines
loInvariantLocale	Use this option where a consistent inter-application operation is desired, inedependent of the actual locale.
loUserLocale	Use this option where the user locale is taken into account.

See Also

CompareStr (see page 4142)

CompareText (see page 4143)

LowerCase (see page 4269)

SameStr (see page 4282)

SameText (see page 4283)

UpperCase (see page 4352)

39.4.12 SysUtils.TMbcsByteType Enumeration

TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).

Pascal

```
TMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
```

```
);
```

C++

```
enum TMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

SysUtils

Description

TMbcsByteType represents the possible return values of the ByteType (see page 4135) and StrByteType (see page 4286) functions. Possible values are as follows:

Values	Meaning
mbSingleByte	The char is used to represent an entire character in the string. The value of Char can not be included in the LeadBytes (see page 4394) set.
mbLeadByte	The char is used to represent the first byte of a multi-byte character. The value of Char must be included in the LeadBytes (see page 4394) set.
mbTrailByte	The char is used to represent one of the trailing bytes in a multi-byte character. There are no a priori restrictions on the value of char.

See Also

ByteType

LeadBytes

StrByteType

39.4.13 SysUtils.TNameType Enumeration

Pascal

```
TNameType = (
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
);
```

C++

```
enum TNameType {
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
};
```

File

SysUtils

Description

This is record SysUtils.TNameType.

39.4.14 SysUtils.TSearchRec Record

TSearchRec defines file information searched for by FindFirst (see page 4218) or FindNext (see page 4221).

Pascal

```
TSearchRec = record
  Time: Integer;
  Size: Int64;
  Attr: Integer;
  Name: TFileName;
  ExcludeAttr: Integer;
  FindHandle: THandle platform;
  FindData: TWin32FindData platform;
end;
```

C++

```
struct TSearchRec {
  int Time;
  Int64 Size;
  int Attr;
  TFileName Name;
  int ExcludeAttr;
  THandle platform FindHandle;
  TWin32FindData platform FindData;
};
```

File

SysUtils

Description

The TSearchRec type defines file information searched for by a FindFirst (see page 4218) or FindNext (see page 4221) function call. If a file is found, the fields of the TSearchRec type parameter are modified to specify the found file.

Attr represents the file attributes of the file. Test Attr against the following attribute constants or values to determine if a file attribute matches the file's properties.

Note: On Windows, attribute constants map directly to DOS file attributes.

Note: On Linux, attribute constants map to sets of file characteristics; use the Mode field to access the specific permission mode.

Constant	Value	Description	Linux Meaning
faReadOnly (see page 4414)	1	Read (see page 3727)-only files	Current user does not have write access.
faHidden (see page 4413)	2	Hidden files	File name begins with ".".
faSysFile (see page 4415)	4	System (see page 3165) files	File is socket, symbolic link, device file, or FIFO.
faVolumeID (see page 4416)	8	Volume ID files	Not used.
faDirectory (see page 4412)	16	Directory files	Directory.

faArchive (see page 4412)	32	Archive files	Not used.
faSymLink (see page 4415)	64	Symbolic link	File is a symbolic link.
faAnyFile (see page 4411)	71	Any file	Any file.

The faReadOnly (see page 4414) constant has the same name as the enumerated value that is defined by the TFieldAttribute type. If both the SysUtils (see page 3967) and the Db units are used in your source files, you must disambiguate by specifying the unit to qualify the use of faReadOnly (see page 4414). That is, write SysUtils.faReadOnly (see page 4414) (Delphi) or SysUtils::faReadOnly (see page 4414) (C++).

To test for an attribute, combine the value of the Attr field with the attribute constant with the and operator. If the file has that attribute, the result will be greater than 0. For example, if the found file is a hidden file, the following expression will evaluate to true:

(SearchRec.Attr and faHidden (see page 4413)) <> 0.

(SearchRec.Attr & faHidden (see page 4413)) != 0.

Time contains the time stamp of the file. It can be converted to a TDateTime (see page 3335) value using FileDateToDateTime (see page 4191).

Size contains the size of the file in bytes.

Name contains the base file name, including extension.

FindHandle is an internal handle used to track find state.

FindData (Windows only) contains additional information such as the file creation time, last access time, and both the long and short file names.

Mode (Linux only) is the file's permission mask.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
```

```
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
      FindClose(sr);
    end;
  end;
end;
```

39.4.15 SysUtils.TSysLocale Record

TSysLocale represents locale information.

Pascal

```
TSysLocale = packed record
  DefaultLCID: Integer;
  PriLangID: Integer;
  SubLangID: Integer;
  FarEast: Boolean;
  MiddleEast: Boolean;
end;
```

C++

```
struct TSysLocale {
  int DefaultLCID;
  int PriLangID;
  int SubLangID;
  Boolean FarEast;
  Boolean MiddleEast;
};
```

File

SysUtils

Description

The TSysLocale type represents a locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

See Also

TLocale

Languages

TLocaleCode

39.4.16 SysUtils.TThreadInfo Record

Pascal

```
TThreadInfo = record
  Next: PThreadInfo;
  ThreadID: Cardinal;
  Active: Integer;
  RecursionCount: Cardinal;
end;
```

C++

```
struct TThreadInfo {
  PThreadInfo Next;
  unsigned ThreadID;
  int Active;
  unsigned RecursionCount;
};
```

File

SysUtils

Description

This is record SysUtils.TThreadInfo.

39.4.17 SysUtils.TTimeStamp Record

TTimeStamp represents time and date values.

Pascal

```
TTimeStamp = record
  Time: Integer;
  Date: Integer;
end;
```

C++

```
struct TTimeStamp {
  int Time;
  int Date;
};
```

File

SysUtils

Description

Use TTimeStamp to represent date and time values when a great deal of accuracy is required for the time portion. When the time values do not need to be precise to the millisecond, use the more compact TDateTime (see page 3335) representation. If additional precision is needed, use TSQLTimeStamp (but do NOT assign it to a Variant (see page 3430)).

The Time field indicates the number of milliseconds that have elapsed since midnight.

The Date field indicates the number of calendar days since the start of the calendar (the number of days since 1/1/0001 plus one).

See Also

DateTimeToTimeStamp

TDateTime

TimeStampToDateTime

TSQLTimeStamp

39.4.18 SysUtils.WordRec Record

WordRec declares a utility record to store high and low order bytes of a variable.

Pascal

```
WordRec = packed record
  case Integer of
    0: (Hi: Byte;
        Lo: Byte;);
    1: (Bytes: array [0..1] of Byte););
end;
```

C++

```
struct WordRec {
  Byte Hi;
  Byte Lo;
  array [0..1] of Byte) Bytes;
};
```

File

SysUtils

Description

The WordRec type declares a utility record that stores the high and low order bytes of the specified variable as type Byte. WordRec is used primarily for typecasting.

39.5 Types

The following table lists types in this documentation.

Types

Name	Description
ExceptClass (see page 4377)	ExceptClass declares an Exception (see page 4015) class.
PByteArray (see page 4378)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 4378)	PDayTable is a pointer to a TDayTable (see page 4381) type.
PExceptionRecord (see page 4379)	PExceptionRecord is a pointer to a TExceptionRecord (see page 4363) type.
PThreadInfo (see page 4379)	This is type SysUtils.PThreadInfo.
PWordArray (see page 4379)	PWordArray is a pointer to a TWordArray (see page 4386) type.
TByteArray (see page 4380)	TByteArray declares an array of Bytes.
TBytes (see page 4380)	TBytes declares an array of Bytes.
TCharArray (see page 4381)	This is type SysUtils.TCharArray.
TDayTable (see page 4381)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 4381)	TFileName is a string that represents a file name.
TIntegerSet (see page 4382)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 4382)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 4061).
TPackageInfoProc (see page 4383)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4253).
TProc (see page 4383)	This is type SysUtils.TProc.
TProcedure (see page 4384)	TProcedure declares a procedure.
TReplaceFlags (see page 4384)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 4384)	TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.
TTerminateProc (see page 4385)	TValidatePackageProc (see page 4385) is a call-back function invoked by LoadPackage (see page 4268).
TValidatePackageProc (see page 4385)	This is type SysUtils.TValidatePackageProc.
TWordArray (see page 4386)	TWordArray declares an array of Words.

39.5.1 SysUtils.ExceptClass Type

ExceptClass declares an Exception (see page 4015) class.

Pascal

```
ExceptClass = class of Exception;
```

C++

```
Exception ExceptClass;
```

File

SysUtils

Description

The ExceptClass type declares an Exception (see page 4015) class. This class encapsulates the fundamental properties and methods for all exceptions. Use Exception (see page 4015) as the base class for creating custom exceptions.

See Also

Exception (see page 4015)

39.5.2 SysUtils.PByteArray Type

PByteArray declares a pointer to an array of Bytes.

Pascal

```
PByteArray = ^TByteArray;
```

C++

```
^TByteArray PByteArray;
```

File

SysUtils

Description

The PByteArray type declares a pointer to a TByteArray (see page 4380) : a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

See Also

TByteArray (see page 4380)

39.5.3 SysUtils.PDayTable Type

PDayTable is a pointer to a TDayTable (see page 4381) type.

Pascal

```
PDayTable = ^TDayTable;
```

C++

```
^TDayTable PDayTable;
```

File

SysUtils

Description

The TDayTable (see page 4381) type points to a TDayTable (see page 4381) variable. TDayTable (see page 4381) declares a static array of 12 Words. It is principally used by the MonthDays (see page 4424) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

TDayTable (see page 4381)

39.5.4 SysUtils.PExceptionRecord Type

PExceptionRecord is a pointer to a TExceptionRecord (see page 4363) type.

Pascal

```
PExceptionRecord = ^TExceptionRecord;
```

C++

```
^TExceptionRecord PExceptionRecord;
```

File

SysUtils

Description

The PExceptionRecord type points to a TExceptionRecord (see page 4363) variable. TExceptionRecord (see page 4363) holds information about an exception in an application.

See Also

TExceptionRecord (see page 4363)

39.5.5 SysUtils.PThreadInfo Type

Pascal

```
PThreadInfo = ^TThreadInfo;
```

C++

```
^TThreadInfo PThreadInfo;
```

File

SysUtils

Description

This is type SysUtils.PThreadInfo.

39.5.6 SysUtils.PWordArray Type

PWordArray is a pointer to a TWordArray (see page 4386) type.

Pascal

```
PWordArray = ^TWordArray;
```

C++

```
^TWordArray PWordArray;
```

File

SysUtils

Description

The PWordArray type points to a TWordArray (see page 4386) variable. The TWordArray (see page 4386) type declares a static array of Words. The array size is 16,384 (2**14) elements.

See Also

TWordArray (see page 4386)

39.5.7 SysUtils.TByteArray Type

TByteArray declares an array of Bytes.

Pascal

```
TByteArray = array[0..32767] of Byte;
```

C++

```
array[0..32767] of Byte TByteArray;
```

File

SysUtils

Description

The TByteArray type declares a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

39.5.8 SysUtils.TBytes Type

TBytes declares an array of Bytes.

Pascal

```
TBytes = array of Byte;
```

C++

```
array of Byte TBytes;
```

File

SysUtils

Description

The TBytes type declares a dynamic array of Bytes.

Note: Dynamic arrays have no intrinsic size. SetLength (see page 3748) is used to allocate storage for the defined array size desired. Dynamic arrays are always integer-indexed, starting at 0.

39.5.9 SysUtils.TCharArray Type

Pascal

```
TCharArray = array of Char;
```

C++

```
array of Char TCharArray;
```

File

SysUtils

Description

This is type SysUtils.TCharArray.

39.5.10 SysUtils.TDayTable Type

TDayTable is used to hold the number of days of each month of a year.

Pascal

```
TDayTable = array[1..12] of Word;
```

C++

```
array[1..12] of Word TDayTable;
```

File

SysUtils

Description

The TDayTable type declares a static array of 12 Words. It is principally used by the MonthDays (see page 4424) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

MonthDays (see page 4424)

39.5.11 SysUtils.TFileName Type

TFileName is a string that represents a file name.

Pascal

```
TFileName = type string;
```

C++

```
type string TFileName;
```

File

SysUtils

Description

Use TFileName to represent strings that are only used for file names.

39.5.12 SysUtils.TIntegerSet Type

TIntegerSet is used to treat an integer as a set of bits.

Pascal

```
TIntegerSet = set of 0..SizeOf(Integer) * 8 - 1;
```

C++

```
0..SizeOf(Integer) * 8 - 1 TIntegerSet;
```

File

SysUtils

Description

Type TIntegerSet = set of 0 .. Integer-bit-size-1;

Assuming that an integer is 4 bytes (=32 bits), the set will 0 .. 31. TIntegerSet is typically used to treat an integer as a set of bits. To do so, an integer variable value is assigned to a TIntegerSet variable using TIntegerSet casting.

39.5.13 SysUtils.TMREWSync Type

TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 4061).

Pascal

```
TMREWSync = TMultiReadExclusiveWriteSynchronizer;
```

C++

```
TMultiReadExclusiveWriteSynchronizer TMREWSync;
```

File

SysUtils

Description

The TMREWSync type is used instead of TMultiReadExclusiveWriteSynchronizer (see page 4061).

Note: TMultiReadExclusiveWriteSynchronizer (see page 4061) is a class that minimizes thread serialization to gain read access to a resource shared among threads while still providing complete exclusivity to callers needing write access to the shared resource.

39.5.14 SysUtils.TPackageInfoProc Type

TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4253).

Pascal

```
TPackageInfoProc = procedure (const Name: string; NameType: TNameType; Flags: Byte; Param:
Pointer);
```

C++

```
(const Name: string; NameType: TNameType; Flags: Byte; Param: Pointer) ( TPackageInfoProc)();
```

File

SysUtils

Description

The TPackageInfoProc type declares the procedure that is invoked by GetPackageInfo (see page 4253) to retrieve package information. It is called for every unit in the package, and every package required by that package.

The following table lists the information provided in these call back invocations of your TPackageInfoProc variable.

Parameter name	Parameter value
Name	Unit or required package name
NameType	Indicates if this call is for a contained unit, required package or DcpBpiName.
Flags	
Param	

See Also

GetPackageInfo (see page 4253)

NameType

39.5.15 SysUtils.TProc Type

Pascal

```
TProc = reference to procedure;
```

C++

```
reference to procedure TProc;
```

File

SysUtils

Description

This is type SysUtils.TProc.

39.5.16 SysUtils.TProcedure Type

TProcedure declares a procedure.

Pascal

```
TProcedure = procedure;
```

C++

```
( TProcedure)();
```

File

SysUtils

Description

Use the TProcedure type as a generic Procedure pointer.

39.5.17 SysUtils.TReplaceFlags Type

TReplaceFlags indicates how to perform a search-and-replace operation

Pascal

```
TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase);
```

C++

```
(rfReplaceAll, rfIgnoreCase) TReplaceFlags;
```

File

SysUtils

Description

TReplaceFlags is a set of flags that govern how to locate and replace a substring within a string. This set can contain the following values:

Value	Meaning
rfReplaceAll	Replace all occurrences. If this flag is not present, only the first occurrence of the target substring is replaced.
rfIgnoreCase	Match occurrences of the substring case-insensitively. If this flag is not present, only case-sensitive matches are considered.

See Also

StringReplace

39.5.18 SysUtils.TSysCharSet Type

TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.

Pascal

```
TSysCharSet = set of AnsiChar;
```

C++

```
AnsiChar TSysCharSet;
```

File

SysUtils

Description

type TSysCharSet = set of Char;

typedef Set<char, 0, 255> TSysCharSet;

TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.

39.5.19 SysUtils.TTerminateProc Type

TValidatePackageProc (see page 4385) is a call-back function invoked by LoadPackage (see page 4268).

Pascal

```
TTerminateProc = function : Boolean;
```

C++

```
: Boolean ( TTerminateProc)();
```

File

SysUtils

Description

TValidatePackageProc (see page 4385) is optionally invoked by LoadPackage (see page 4268) when a package is loaded. The package is not loaded if TValidatePackageProc (see page 4385) determines that it is not valid and hence returns false.

See Also

LoadPackage (see page 4268)

39.5.20 SysUtils.TValidatePackageProc Type

Pascal

```
TValidatePackageProc = function (Module: HMODULE): Boolean;
```

C++

```
(Module: HMODULE): Boolean ( TValidatePackageProc)();
```

File

SysUtils

Description

This is type SysUtils.TValidatePackageProc.

39.5.21 SysUtils.TWordArray Type

TWordArray declares an array of Words.

Pascal

```
TWordArray = array[0..16383] of Word;
```

C++

```
array[0..16383] of Word TWordArray;
```

File

SysUtils

Description

The TWordArray type declares a static array of Words. The array size is 16,384 (2**14) elements.

Note: Static arrays are declared with an immutable size.

39.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CurrencyDecimals (see page 4388)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 4388)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 4389)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 4389)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 4390)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyAnsiStr (see page 4391)	This is variable SysUtils.EmptyAnsiStr.
EmptyStr (see page 4391)	An empty string constant.
EmptyWideStr (see page 4391)	An empty string version of the WideString (see page 3476) constant.
EraNames (see page 4392)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 4392)	Controls the DateTimeToStr (see page 4153) function (for East Asian locales).
FalseBoolStrs (see page 4392)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 4393)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 4393)	HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.
LeadBytes (see page 4394)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 4394)	The character used to separate items in a list.
LongDateFormat (see page 4394)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 4395)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a long day name to be substituted.
LongMonthNames (see page 4395)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a long month name to be substituted.
LongTimeFormat (see page 4396)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 4396)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullAnsiStr (see page 4397)	This is variable SysUtils.NullAnsiStr.
NullStr (see page 4397)	A pointer to an empty string.
NullWideStr (see page 4398)	A pointer to an empty wide string.
ShortDateFormat (see page 4398)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 4399)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a short day name to be substituted.
ShortMonthNames (see page 4399)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a short month name to be substituted.
ShortTimeFormat (see page 4400)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 4400)	SysLocale holds locale information.
ThousandSeparator (see page 4402)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.

TimeAMString (see page 4402)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.
TimePMString (see page 4402)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.
TimeSeparator (see page 4403)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 4403)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 4404)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 4404)	Specifies which the build level of the Windows Operating System (see page 3165) operating.
Win32CSDVersion (see page 4405)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 4405)	Specifies which major Windows Operating System (see page 3165) version is operating.
Win32MinorVersion (see page 4406)	Specifies which minor Windows Operating System (see page 3165) version is operating.
Win32Platform (see page 4406)	Specifies the Windows Operating System (see page 3165) platform identifier.

39.6.1 SysUtils.CurrencyDecimals Variable

CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.

Pascal

```
CurrencyDecimals: Byte;
```

C++

```
Byte CurrencyDecimals;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRDIGITS.

39.6.2 SysUtils.CurrencyFormat Variable

CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 | 1 = 1$ | 2 = $ 1 | 3 = 1 $

Pascal

```
CurrencyFormat: Byte;
```

C++

```
Byte CurrencyFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRENCY.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

39.6.3 SysUtils.CurrencyString Variable

CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.

Pascal

```
CurrencyString: string;
```

C++

```
UnicodeString CurrencyString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SCURRENCY.

39.6.4 SysUtils.DateSeparator Variable

DateSeparator is the character used to separate the year, month, and day parts of a date value.

Pascal

```
DateSeparator: Char;
```

C++

```
Char DateSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDATE.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

39.6.5 SysUtils.DecimalSeparator Variable

DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.

Pascal

```
DecimalSeparator: Char;
```

C++

```
Char DecimalSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDECIMAL.

39.6.6 SysUtils.EmptyAnsiStr Variable

Pascal

```
EmptyAnsiStr: AnsiString = '';
```

C++

```
AnsiString EmptyAnsiStr = '';
```

File

SysUtils

Description

This is variable SysUtils.EmptyAnsiStr.

39.6.7 SysUtils.EmptyStr Variable

An empty string constant.

Pascal

```
EmptyStr: string = '';
```

C++

```
UnicodeString EmptyStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

39.6.8 SysUtils.EmptyWideStr Variable

An empty string version of the WideString (see page 3476) constant.

Pascal

```
EmptyWideStr: WideString = '';
```

C++

```
BSTR EmptyWideStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

39.6.9 SysUtils.EraNames Variable

Defines the names of calendar eras for East Asian locales.

Pascal

```
EraNames: array [1..MaxEraCount] of string;
```

C++

```
array [1..MaxEraCount] of string EraNames;
```

File

SysUtils

Description

EraNames defines an array of 7 era names for Japanese and Taiwanese locales. It is principally used for date formatting.

See Also

FormatDateTime (see page 4239)

DateTimeToString (see page 4153)

39.6.10 SysUtils.EraYearOffsets Variable

Controls the DateTimeToStr (see page 4153) function (for East Asian locales).

Pascal

```
EraYearOffsets: array [1..MaxEraCount] of Integer;
```

C++

```
array [1..MaxEraCount] of Integer EraYearOffsets;
```

File

SysUtils

39.6.11 SysUtils.FalseBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
FalseBoolStrs: array of String;
```

C++

```
array of String FalseBoolStrs;
```

File

SysUtils

Description

FalseBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 4130) function, false values are converted to the first string in the list.

When using the StrToBool (see page 4318) function, any string found in the list, regardless of case, is converted to false. If no string match is found, TrueBoolStrs (see page 4403) is checked, true being returned if a match is found. No matches result in the EConvertError (see page 3985) exception being thrown.

See Also

StrToBool (see page 4318)

TryStrToBool (see page 4346)

TrueBoolStrs (see page 4403)

BoolToStr (see page 4130)

39.6.12 SysUtils.GetDiskFreeSpaceEx Variable

Pascal

```
GetDiskFreeSpaceEx: function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger;
TotalFree: PLargeInteger): Bool stdcall = nil;
```

C++

```
function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger; TotalFree:
PLargeInteger): Bool stdcall GetDiskFreeSpaceEx = nil;
```

File

SysUtils

Description

This is variable SysUtils.GetDiskFreeSpaceEx.

39.6.13 SysUtils.HexDisplayPrefix Variable

HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.

Pascal

```
HexDisplayPrefix: string = '$';
```

C++

```
UnicodeString HexDisplayPrefix = '$';
```

File

SysUtils

Description

This is for display only - this does not affect the string-to-integer conversion routines.

39.6.14 SysUtils.LeadBytes Variable

Lists all values that can be the first byte of a double byte character on the local system.

Pascal

```
LeadBytes: set of AnsiChar = [];
```

C++

```
set of AnsiChar LeadBytes = [];
```

File

SysUtils

Description

Use LeadBytes to test whether a value can represent the lead byte in a double byte character. If an individual value is in the LeadBytes set, it is not necessarily used as the lead byte of a double byte character. It may also appear as the second byte of a double byte character. To determine whether the character is used as a lead byte, use the global ByteType (see page 4135) or StrByteType (see page 4286) function.

Note: LeadBytes is always empty when running on a system with a locale that does not use a multi byte character system (MBCS).

See Also

ByteType (see page 4135)

StrByteType (see page 4286)

39.6.15 SysUtils.ListSeparator Variable

The character used to separate items in a list.

Pascal

```
ListSeparator: Char;
```

C++

```
Char ListSeparator;
```

File

SysUtils

Description

On Windows, the default value is fetched from LOCALE_SLIST.

39.6.16 SysUtils.LongDateFormat Variable

LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.

Pascal

```
LongDateFormat: string;
```

C++

```
UnicodeString LongDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SLONGDATE.

39.6.17 SysUtils.LongDayNames Variable

LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a long day name to be substituted.

Pascal

```
LongDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string LongDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SDAYNAME system locale entries.

39.6.18 SysUtils.LongMonthNames Variable

LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a long month name to be substituted.

Pascal

```
LongMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string LongMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SMONTHNAME system locale entries.

39.6.19 SysUtils.LongTimeFormat Variable

LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.

Pascal

```
LongTimeFormat: string;
```

C++

```
UnicodeString LongTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

39.6.20 SysUtils.NegCurrFormat Variable

NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.

Pascal

```
NegCurrFormat: Byte;
```

C++

```
Byte NegCurrFormat;
```

File

SysUtils

Description

Possible values are:

Value	Format (see page 4232)
0	($1)
1	-$1
2	$-1
3	$1-
4	(1$)
5	-1$
6	1-$
7	1$-
8	-1 $
9	-$ 1
10	1 $-
11	$ 1-

12	$ -1
13	1- $
14	($ 1)
15	(1 $)

On Windows, the initial value is fetched from LOCALE_INEGCURR.

39.6.21 SysUtils.NullAnsiStr Variable

Pascal

```
NullAnsiStr: PAnsiString = @EmptyAnsiStr;
```

C++

```
PAnsiString NullAnsiStr = @EmptyAnsiStr;
```

File

SysUtils

Description

This is variable SysUtils.NullAnsiStr.

39.6.22 SysUtils.NullStr Variable

A pointer to an empty string.

Pascal

```
NullStr: PString = @EmptyStr;
```

C++

```
PString NullStr = @EmptyStr;
```

File

SysUtils

Description

NullStr defines a pointer to the empty string EmptyStr (see page 4391).

See Also

EmptyStr (see page 4391)

NullWideStr (see page 4398)

PString

39.6.23 SysUtils.NullWideStr Variable

A pointer to an empty wide string.

Pascal

```
NullWideStr: PWideString = @EmptyWideStr;
```

C++

```
PWideString NullWideStr = @EmptyWideStr;
```

File

SysUtils

Description

NullWideStr defines a pointer to the empty wide string EmptyWideStr (see page 4391).

See Also

EmptyWideStr (see page 4391)

NullStr (see page 4397)

PWideString (see page 3853)

39.6.24 SysUtils.ShortDateFormat Variable

ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.

Pascal

```
ShortDateFormat: string;
```

C++

```
UnicodeString ShortDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SSHORTDATE.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
```

```
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

39.6.25 SysUtils.ShortDayNames Variable

ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4239) causes a short day name to be substituted.

Pascal

```
ShortDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string ShortDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVDAYNAME system locale entries.

39.6.26 SysUtils.ShortMonthNames Variable

ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4239) causes a short month name to be substituted.

Pascal

```
ShortMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string ShortMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVMONTHNAME system locale entries.

39.6.27 SysUtils.ShortTimeFormat Variable

ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.

Pascal

```
ShortTimeFormat: string;
```

C++

```
UnicodeString ShortTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

39.6.28 SysUtils.SysLocale Variable

SysLocale holds locale information.

Pascal

```
SysLocale: TSysLocale;
```

C++

```
TSysLocale SysLocale;
```

File

SysUtils

Description

The SysLocale holds the attributes of the current locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
```

```
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.t_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.t_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
      Memo1->Lines->Add(buffer);
      buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
      Memo1->Lines->Add(buffer);
      Memo1->Lines->Add("");
    }
  }
  __finally
  {
    Memo1->Lines->EndUpdate();
  }
}
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

GetTSysLocale

Languages (see page 4267)

39.6.29 SysUtils.ThousandSeparator Variable

ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.

Pascal

```
ThousandSeparator: Char;
```

C++

```
Char ThousandSeparator;
```

File

SysUtils

Description

A value of #0 indicates that no thousand separator character should be output, even if the format string specifies it.

On Windows, the initial value is fetched from LOCALE_STHOUSAND.

39.6.30 SysUtils.TimeAMString Variable

TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.

Pascal

```
TimeAMString: string;
```

C++

```
UnicodeString TimeAMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S1159.

39.6.31 SysUtils.TimePMString Variable

TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.

Pascal

```
TimePMString: string;
```

C++

```
UnicodeString TimePMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S2359.

39.6.32 SysUtils.TimeSeparator Variable

TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.

Pascal

```
TimeSeparator: Char;
```

C++

```
Char TimeSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_STIME.

39.6.33 SysUtils.TrueBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
TrueBoolStrs: array of String;
```

C++

```
array of String TrueBoolStrs;
```

File

SysUtils

Description

TrueBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 4130) function, false values are converted to the first string in the list.

When using the StrToBool (see page 4318) function, any string found in the list, regardless of case, is converted to true. If no string match is found, FalseBoolStrs (see page 4392) is checked, false being returned if a match is found. No matches result in the EConvertError (see page 3985) exception being thrown.

See Also

StrToBool (see page 4318)

TryStrToBool (see page 4346)

FalseBoolStrs (see page 4392)

BoolToStr (see page 4130)

39.6.34 SysUtils.TwoDigitYearCenturyWindow Variable

Determines what century is added to two-digit years when string dates are converted to numeric dates.

Pascal

```
TwoDigitYearCenturyWindow: Word = 50;
```

C++

```
Word TwoDigitYearCenturyWindow = 50;
```

File

SysUtils

Description

The value of TwoDigitYearCenturyWindow, if it is nonzero, is subtracted from the current year to calculate the floating "pivot" prior to which two-digit years are interpreted as falling in the next century. The default value is 50. The table below shows some examples.

39.6.35 SysUtils.Win32BuildNumber Variable

Specifies which the build level of the Windows Operating System (see page 3165) operating.

Pascal

```
Win32BuildNumber: Integer = 0;
```

C++

```
int Win32BuildNumber = 0;
```

File

SysUtils

Description

Win32BuildNumber specifies the build number of the Operating System (see page 3165). The unique identifier for a Windows Operating System (see page 3165) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber

Note: Win32BuildNumber is available on Windows only.

See Also

Win32CSDVersion (see page 4405)

Win32MinorVersion (see page 4406)

Win32MajorVersion (see page 4405)

Win32Platform (see page 4406)

39.6.36 SysUtils.Win32CSDVersion Variable

Specifies the Win32 latest installed service pack.

Pascal

```
Win32CSDVersion: string = '';
```

C++

```
UnicodeString Win32CSDVersion = '';
```

File

SysUtils

Description

Win32CSDVersion specifies the name of the most recent service pack.

Note: Win32CSDVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4404)

Win32MajorVersion (see page 4405)

Win32MinorVersion (see page 4406)

Win32Platform (see page 4406)

39.6.37 SysUtils.Win32MajorVersion Variable

Specifies which major Windows Operating System (see page 3165) version is operating.

Pascal

```
Win32MajorVersion: Integer = 0;
```

C++

```
int Win32MajorVersion = 0;
```

File

SysUtils

Description

Win32MajorVersion specifies the Major version of the Operating System (see page 3165). The unique identifier for a Windows Operating System (see page 3165) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 4404)

Note: Win32MajorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4404)

Win32CSDVersion (see page 4405)

Win32MinorVersion (see page 4406)

Win32Platform (see page 4406)

39.6.38 SysUtils.Win32MinorVersion Variable

Specifies which minor Windows Operating System (see page 3165) version is operating.

Pascal

```
Win32MinorVersion: Integer = 0;
```

C++

```
int Win32MinorVersion = 0;
```

File

SysUtils

Description

Win32MinorVersion specifies the Minor version of the Operating System (see page 3165). The unique identifier for a Windows Operating System (see page 3165) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 4404)

Note: Win32MinorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4404)

Win32CSDVersion (see page 4405)

Win32MajorVersion (see page 4405)

Win32Platform (see page 4406)

39.6.39 SysUtils.Win32Platform Variable

Specifies the Windows Operating System (see page 3165) platform identifier.

Pascal

```
Win32Platform: Integer = 0;
```

C++

```
int Win32Platform = 0;
```

File

SysUtils

Description

Win32Platform is used to identify the type of Win32 platform the system is running. This will be one of the following values:

Value	Meaning
VER_PLATFORM_WIN32s = 0	System (see page 3165) is Win32s.
VER_PLATFORM_WIN32_WINDOWS = 1	System (see page 3165) is Windows 95.

VER_PLATFORM_WIN32_NT = 2	System (see page 3165) is Windows NT. This includes Windows XP and Vista.

Note: Win32Platform is available on Windows only.

See Also

Win32BuildNumber (see page 4404)

Win32CSDVersion (see page 4405)

Win32MajorVersion (see page 4405)

Win32MinorVersion (see page 4406)

39.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DateDelta (see page 4409)	Specifies the difference between TDateTime (see page 3335) and TTimeStamp (see page 4375) starting values.
DefaultFalseBoolStr (see page 4410)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 4410)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 4411)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 4411)	Specifies any file type.
faArchive (see page 4412)	Specifies only Windows archived file types.
faDirectory (see page 4412)	Specifies only directory 'file' types.
faHidden (see page 4413)	Specifies only hidden file types.
faNormal (see page 4414)	This is constant SysUtils.faNormal.
faReadOnly (see page 4414)	Specifies only read only file types.
faSymLink (see page 4415)	Specifies only symbolic link file types.
faSysFile (see page 4415)	Specifies only system file types.
faTemporary (see page 4416)	This is constant SysUtils.faTemporary.
faVolumeID (see page 4416)	Specifies Volume ID file types.
fmOpenRead (see page 4417)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 4417)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 4418)	Specifies a file should be opened for writing only.
fmShareCompat (see page 4419)	This is constant SysUtils.fmShareCompat.
fmShareDenyNone (see page 4419)	This is constant SysUtils.fmShareDenyNone.
fmShareDenyRead (see page 4419)	This is constant SysUtils.fmShareDenyRead.
fmShareDenyWrite (see page 4420)	This is constant SysUtils.fmShareDenyWrite.
fmShareExclusive (see page 4420)	This is constant SysUtils.fmShareExclusive.
HoursPerDay (see page 4420)	Specifies the number of hours in a day.
MaxCurrency (see page 4421)	Defines the largest value supported for Currency (see page 3242) data types.
MaxDateTime (see page 4421)	Specifies the highest supported TDateTime (see page 3335) value.
MaxEraCount (see page 4422)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 4422)	Defines the largest negative value supported for Currency (see page 3242) data types.
MinDateTime (see page 4423)	Specifies the earliest supported TDateTime (see page 3335) value.
MinsPerDay (see page 4423)	Specifies the number of minutes in a day.
MinsPerHour (see page 4424)	Specifies the number of minutes in an hour.
MonthDays (see page 4424)	An array specifying how many days are in a month.
MSecsPerDay (see page 4425)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 4425)	Specifies the number of milliseconds in a second.
PathDelim (see page 4426)	Specifies a path delimiter character.
PathSep (see page 4426)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 4427)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 4427)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 4428)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 4428)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 4428)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 4429)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 4429)	Indicates that a package was compiled with {$DESIGNONLY ON}.

pfExeModule (see page 4430)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 4430)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 4431)	Indicates that a module is a library.
pfModuleTypeMask (see page 4431)	Indicates that a package is a module.
pfNeverBuild (see page 4432)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 4432)	Indicates that a module is a package.
pfProducerMask (see page 4433)	Indicates the producer of a package.
pfProducerUndefined (see page 4433)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 4434)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 4434)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 4435)	Specifies the number of seconds in a day.
SecsPerHour (see page 4435)	This is constant SysUtils.SecsPerHour.
SecsPerMin (see page 4436)	Specifies the number of seconds in a minute.
SwitchChars (see page 4436)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 4437)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 4437)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 4438)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 4438)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 4439)	This is constant SysUtils.ufWeakPackageUnit.
ufWeakUnit (see page 4439)	Indicates that a unit is a weak unit.
UnixDateDelta (see page 4439)	Specifies the difference between TDateTime (see page 3335) and Lunix date/time origins.

39.7.1 SysUtils.DateDelta Constant

Specifies the difference between TDateTime (see page 3335) and TTimeStamp (see page 4375) starting values.

Pascal

```
DateDelta = 693594;
```

C++

```
DateDelta = 693594;
```

File

SysUtils

Description

DateDelta is the number of days (693594) between January 1, 0001 (starting TTimeStamp (see page 4375) value), and December 31, 1899 (starting TDateTime (see page 3335) value). This value is useful for converting date-time values between TTimeStamp (see page 4375) and TDateTime (see page 3335).

See Also

TDateTime (see page 3858)

TTimeStamp (see page 4375)

UnixDateDelta (see page 4439)

39.7.2 SysUtils.DefaultFalseBoolStr Constant

Specifies the default string representation of the boolean false value.

Pascal

```
DefaultFalseBoolStr = 'False';
```

C++

```
DefaultFalseBoolStr = 'False';
```

File

SysUtils

Description

DefaultFalseBoolStr is 'False'. It is used internally by the StrToBool (see page 4318) and BoolToStr (see page 4130) routines. They assign DefaultFalseBoolStr to the 1st element of the FalseBoolStrs (see page 4392) array, used by these routines as a default string representation of the boolean false value.

See Also

DefaultTrueBoolStr (see page 4410)

StrToBool (see page 4318)

FalseBoolStrs (see page 4392)

TrueBoolStrs (see page 4403)

BoolToStr (see page 4130)

39.7.3 SysUtils.DefaultTrueBoolStr Constant

Specifies the default string representation of the boolean true value.

Pascal

```
DefaultTrueBoolStr = 'True';
```

C++

```
DefaultTrueBoolStr = 'True';
```

File

SysUtils

Description

DefaultTrueBoolStr is 'True'. It is used internally by the StrToBool (see page 4318) and BoolToStr (see page 4130) routines. They assign DefaultTrueBoolStr to the 1st element of the TrueBoolStrs (see page 4403) array, used by these routines as a default string representation of the boolean true value.

See Also

DefaultFalseBoolStr (see page 4410)

StrToBool (see page 4318)

FalseBoolStrs (see page 4392)

TrueBoolStrs (see page 4403)

BoolToStr (see page 4130)

39.7.4 SysUtils.DriveDelim Constant

Specifies the character used to separate the Windows drive from the path in a file name.

Pascal

```
DriveDelim = ':';
```

C++

```
DriveDelim = ':';
```

File

SysUtils

Description

DriveDelim is the ':' character that follows the drive character ('A', 'B' and so on) in a full filename on a Windows Operating System (see page 3165). The value is a null character for Linux Operating Systems since they do not support drive designations.

See Also

PathDelim (see page 4426)

PathSep (see page 4426)

39.7.5 SysUtils.faAnyFile Constant

Specifies any file type.

Pascal

```
faAnyFile = $000001FF;
```

C++

```
faAnyFile = 0x000001FF;
```

File

SysUtils

Description

faAnyFile is principally used by FindFirst (see page 4218) to indicate that files of any type encountered sould be returned. Namely, all files.

See Also

faArchive (see page 4412)

faDirectory (see page 4412)

faHidden (see page 4413)

faReadOnly (see page 4414)

faSymLink (see page 4415)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.6 SysUtils.faArchive Constant

Specifies only Windows archived file types.

Pascal

```
faArchive = $00000020 platform;
```

C++

```
faArchive = 0x00000020 platform;
```

File

SysUtils

Description

faArchive is principally used by FindFirst (see page 4218) to indicate that only archived files sould be returned. It is Windows specific : Linux does not support archiving.

See Also

faAnyFile (see page 4411)

faDirectory (see page 4412)

faHidden (see page 4413)

faReadOnly (see page 4414)

faSymLink (see page 4415)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.7 SysUtils.faDirectory Constant

Specifies only directory 'file' types.

Pascal

```
faDirectory = $00000010;
```

C++

```
faDirectory = 0x00000010;
```

File

SysUtils

Description

faDirectory is principally used by FindFirst (see page 4218) to indicate that only directories sould be returned. Whilst FindFirst (see page 4218) is generally used to search for files, it can also return directories within the search directory.

See Also

faAnyFile (see page 4411)

faArchive (see page 4412)

faHidden (see page 4413)

faReadOnly (see page 4414)

faSymLink (see page 4415)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.8 SysUtils.faHidden Constant

Specifies only hidden file types.

Pascal

```
faHidden = $00000002 platform;
```

C++

```
faHidden = 0x00000002 platform;
```

File

SysUtils

Description

faHidden is principally used by FindFirst (see page 4218) to indicate that only hidden files sould be returned. On Linux systems, hidden files are those that start with a "." character.

See Also

faAnyFile (see page 4411)

faArchibe

faDirectory (see page 4412)

faReadOnly (see page 4414)

faSymLink (see page 4415)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.9 SysUtils.faNormal Constant

Pascal

```
faNormal = $00000080 platform;
```

C++

```
faNormal = 0x00000080 platform;
```

File

SysUtils

Description

This is constant SysUtils.faNormal.

39.7.10 SysUtils.faReadOnly Constant

Specifies only read only file types.

Pascal

```
faReadOnly = $00000001 platform;
```

C++

```
faReadOnly = 0x00000001 platform;
```

File

SysUtils

Description

faReadOnly is principally used by FindFirst (see page 4218) to indicate that only read only files sould be returned. On Linux systems, read only is not necessarily a system wide attribute - access rights are user and group dependent. Read (see page 3727) only here means that the current application does read but not write to a file.

See Also

faAnyFile (see page 4411)

faArchibe

faDirectory (see page 4412)

faHidden (see page 4413)

faSymLink (see page 4415)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.11 SysUtils.faSymLink Constant

Specifies only symbolic link file types.

Pascal

```
faSymLink = $00000040 platform;
```

C++

```
faSymLink = 0x00000040 platform;
```

File

SysUtils

Description

faSymLink is principally used by FindFirst (see page 4218) to indicate that only symbolic link files sould be returned. Symbolic links are a feature of Linux systems, with no Windows counterpart.

See Also

faAnyFile (see page 4411)

faArchibe

faDirectory (see page 4412)

faHidden (see page 4413)

faReadOnly (see page 4414)

faSysFile (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.12 SysUtils.faSysFile Constant

Specifies only system file types.

Pascal

```
faSysFile = $00000004 platform;
```

C++

```
faSysFile = 0x00000004 platform;
```

File

SysUtils

Description

faSysFile is principally used by FindFirst (see page 4218) to indicate that only system files sould be returned. On Linux systems, a system file is one that is treated as a socket, symbolic link, device file, or FIFO.

See Also

faAnyFile (see page 4411)

faArchibe

faDirectory (see page 4412)

faHidden (see page 4413)

faReadOnly (see page 4414)

faSymLink (see page 4415)

FindFirst (see page 4218)

TSearchRec (see page 4371)

39.7.13 SysUtils.faTemporary Constant

Pascal

```
faTemporary = $00000100 platform;
```

C++

```
faTemporary = 0x00000100 platform;
```

File

SysUtils

Description

This is constant SysUtils.faTemporary.

39.7.14 SysUtils.faVolumeID Constant

Specifies Volume ID file types.

Pascal

```
faVolumeID = $00000008 platform deprecated;
```

C++

```
faVolumeID = 0x00000008 platform deprecated;
```

File

SysUtils

Description

faVolumeID is deprecated.

39.7.15 SysUtils.fmOpenRead Constant

Specifies a file should be opened for reading only.

Pascal

```
fmOpenRead = $0000;
```

C++

```
fmOpenRead = 0x0000;
```

File

SysUtils

Description

fmOpenRead is one of the values you can assign to FileMode (see page 3892). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenRead, subsequent calls to Reset (see page 3734) open files for reading only.

Note: You must use this value when you plan to read a read only file - using fmOpenReadWrite (see page 4417) will cause the open to fail because it will attempt and fail to get write access.

See Also

fmOpenReadWrite (see page 4417)

fmOpenWrite (see page 4418)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 3734)

FileMode (see page 3892)

Create

39.7.16 SysUtils.fmOpenReadWrite Constant

Specifies a file should be opened for reading and writing.

Pascal

```
fmOpenReadWrite = $0002;
```

C++

```
fmOpenReadWrite = 0x0002;
```

File

SysUtils

Description

fmOpenReadWrite is one of the values you can assign to FileMode (see page 3892). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenReadWrite, subsequent calls to Reset (see page 3734) open files for reading and writing.

Note: Do not assign this value when you plan to read a read only file - the open will fail because write access will be attempted but be denied.

See Also

fmOpenRead (see page 4417)

fmOpenWrite (see page 4418)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 3734)

FileMode (see page 3892)

Create

39.7.17 SysUtils.fmOpenWrite Constant

Specifies a file should be opened for writing only.

Pascal

```
fmOpenWrite = $0001;
```

C++

```
fmOpenWrite = 0x0001;
```

File

SysUtils

Description

fmOpenWrite is one of the values you can assign to FileMode (see page 3892). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenWrite, subsequent calls to Reset (see page 3734) open files for writing only.

See Also

fmOpenRead (see page 4417)

fmOpenReadWrite (see page 4417)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 3734)

FileMode (see page 3892)

Create

39.7.18 SysUtils.fmShareCompat Constant

Pascal

```
fmShareCompat = $0000 platform;
```

C++

```
fmShareCompat = 0x0000 platform;
```

File

SysUtils

Description

This is constant SysUtils.fmShareCompat.

39.7.19 SysUtils.fmShareDenyNone Constant

Pascal

```
fmShareDenyNone = $0040;
```

C++

```
fmShareDenyNone = 0x0040;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyNone.

39.7.20 SysUtils.fmShareDenyRead Constant

Pascal

```
fmShareDenyRead = $0030 platform;
```

C++

```
fmShareDenyRead = 0x0030 platform;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyRead.

39.7.21 SysUtils.fmShareDenyWrite Constant

Pascal

```
fmShareDenyWrite = $0020;
```

C++

```
fmShareDenyWrite = 0x0020;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyWrite.

39.7.22 SysUtils.fmShareExclusive Constant

Pascal

```
fmShareExclusive = $0010;
```

C++

```
fmShareExclusive = 0x0010;
```

File

SysUtils

Description

This is constant SysUtils.fmShareExclusive.

39.7.23 SysUtils.HoursPerDay Constant

Specifies the number of hours in a day.

Pascal

```
HoursPerDay = 24;
```

C++

```
HoursPerDay = 24;
```

File

SysUtils

Description

HoursPerDay is the number of hours (24) in a day. Using this constant rather than hard coding 24 makes for more readable code.

See Also

DateDelta (see page 4409)

MinsPerDay (see page 4423)

MinsPerHour (see page 4424)

MSecsPerDay (see page 4425)

MSecsPerSec (see page 4425)

SecsPerDay (see page 4435)

SecsPerMin (see page 4436)

39.7.24 SysUtils.MaxCurrency Constant

Defines the largest value supported for Currency (see page 3242) data types.

Pascal

```
MaxCurrency: Currency = 922337203685477.5807;
```

C++

```
Currency MaxCurrency = 922337203685477.5807;
```

File

SysUtils

Description

MaxCurrency is the largest value that can be held by a Currency (see page 3242) data variable on the Windows Opersting System (see page 3165).

Note: Linux supports a value 1.0 less than this maximum (922337203685476.5807).

See Also

MinCurrency (see page 4422)

39.7.25 SysUtils.MaxDateTime Constant

Specifies the highest supported TDateTime (see page 3335) value.

Pascal

```
MaxDateTime: TDateTime = 2958465.99999;
```

C++

```
TDateTime MaxDateTime = 2958465.99999;
```

File

SysUtils

Description

MaxDateTime is the most future date : 12/31/9999 23:59:59.999 that the TDateTime (see page 3335) type supports. The actual shown value of 2958465.99999 reflects the fact that TDateTime (see page 3335) is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

See Also

TDateTime (see page 3858)

MinDateTime (see page 4423)

39.7.26 SysUtils.MaxEraCount Constant

Pascal

```
MaxEraCount = 7;
```

C++

```
MaxEraCount = 7;
```

File

SysUtils

Description

This is constant SysUtils.MaxEraCount.

39.7.27 SysUtils.MinCurrency Constant

Defines the largest negative value supported for Currency (see page 3242) data types.

Pascal

```
MinCurrency: Currency = -922337203685477.5807;
```

C++

```
Currency MinCurrency = -922337203685477.5807;
```

File

SysUtils

Description

MinCurrency is the largest negative value that can be held by a Currency (see page 3242) data variable on the Windows Opersting System (see page 3165).

Note: Linux supports a value 1.0 less negative than this minimum (-922337203685476.5807).

See Also

MaxCurrency (see page 4421)

39.7.28 SysUtils.MinDateTime Constant

Specifies the earliest supported TDateTime (see page 3335) value.

Pascal

```
MinDateTime: TDateTime = -657434.0;
```

C++

```
TDateTime MinDateTime = -657434.0;
```

File

SysUtils

Description

MinDateTime is the most historic date : 01/01/0100 00:00:00.000 that the TDateTime (see page 3335) type supports. The actual shown value of -657434.0 reflects the fact that TDateTime (see page 3335) is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

Note: The fractional part of a TDateTime (see page 3335) is always treated in a positive sense. It always means the elapsed part of a day - if the TDateTime (see page 3335) value is negative, this refers only to a negative offset of the integral day value from 12/30/1899.

See Also

TDateTime (see page 3858)

MaxDateTime (see page 4421)

39.7.29 SysUtils.MinsPerDay Constant

Specifies the number of minutes in a day.

Pascal

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

C++

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

File

SysUtils

Description

MinsPerDay is the number of minutes (1440) in a day. It is the product of HoursPerDay (see page 4420) and MinsPerHour (see page 4424).

Using this constant rather than hard coding 1440 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerHour (see page 4424)

MSecsPerDay (see page 4425)

MSecsPerSec (see page 4425)

SecsPerDay (see page 4435)

SecsPerMinute

39.7.30 SysUtils.MinsPerHour Constant

Specifies the number of minutes in an hour.

Pascal

```
MinsPerHour = 60;
```

C++

```
MinsPerHour = 60;
```

File

SysUtils

Description

MinsPerHour is the number of minutes (60) in an hour. Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerHour

MSecsPerDay (see page 4425)

MSecsPerSec (see page 4425)

SecsPerDay (see page 4435)

SecsPerMin (see page 4436)

39.7.31 SysUtils.MonthDays Constant

An array specifying how many days are in a month.

Pascal

```
MonthDays: array [Boolean] of TDayTable = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

C++

```
array [Boolean] of TDayTable MonthDays = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

File

SysUtils

Description

The MonthDays array can be used to quickly find the number of days in a month:

```
MonthDays[IsLeapYear(Y), M]
```

39.7.32 SysUtils.MSecsPerDay Constant

Specifies the number of milliseconds in a day.

Pascal

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

C++

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

File

SysUtils

Description

MSecsPerDay is the number of milliseconds (86,400,000) in a day. It is the product of SecsPerDay (see page 4435) and MSecsPerSec (see page 4425).

Using this constant rather than hard coding 86400000 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerDay (see page 4423)

MinsPerHour (see page 4424)

MSecsPerSec (see page 4425)

SecsPerDay (see page 4435)

SecsPerMin (see page 4436)

39.7.33 SysUtils.MSecsPerSec Constant

Specifies the number of milliseconds in a second.

Pascal

```
MSecsPerSec = 1000;
```

C++

```
MSecsPerSec = 1000;
```

File

SysUtils

Description

MSecsPerSec is the number of milliseconds (1000) in a second.

Using this constant rather than hard coding 1000 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerDay (see page 4423)

MinsPerHour (see page 4424)

MSecsPerDay (see page 4425)

SecsPerMin (see page 4436)

SecsPerDay (see page 4435)

39.7.34 SysUtils.PathDelim Constant

Specifies a path delimiter character.

Pascal

```
PathDelim = '\';
```

C++

```
PathDelim = '\';
```

File

SysUtils

39.7.35 SysUtils.PathSep Constant

Specifies the character used to separate a sequence of file or directory names.

Pascal

```
PathSep = ';';
```

C++

```
PathSep = ';';
```

File

SysUtils

Description

PathSep is the ';' character for Windows, or ':' for Linux that is used to separate filenames in a sequence of file or directory names. FileSearch (see page 4206), for example, expects PathSep to be used to separate the values in the string of directory

names.

See Also

DriveDelim (see page 4411)

PathSep

FileSearch (see page 4206)

39.7.36 SysUtils.pfBCB4Produced Constant

Indicates that a package was produced by via C++.

Pascal

```
pfBCB4Produced = $08000000;
```

C++

```
pfBCB4Produced = 0x08000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfBCB4Produced is one of these flag values, indicating that a package was produced by C++. BCB stands for Borland C++ Builder.

See Also

GetPackageInfo (see page 4253)

39.7.37 SysUtils.pfConsumerBCB Constant

Indicates that C++ can be a consumer of a package.

Pascal

```
pfConsumerBCB = $00200000;
```

C++

```
pfConsumerBCB = 0x00200000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfConsumerBCB is one of these flag values, indicating that C++ is a legitimate consumer, or user, of a package. The C++ consumer is the Borland C++ Builder compiler, linking in the package.

A package will be marked with the pfConsumerBCB when the -JL flag has been set during compilation. This ensures that the extra support files (.lib, .bpi, .hpp)required by a C++ consumer are generated and included in the package.

See Also

GetPackageInfo (see page 4253)

39.7.38 SysUtils.pfConsumerCompat Constant

Pascal

```
pfConsumerCompat = $00000000;
```

C++

```
pfConsumerCompat = 0x00000000;
```

File

SysUtils

Description

This is constant SysUtils.pfConsumerCompat.

39.7.39 SysUtils.pfConsumerDelphi Constant

Indicates that Delphi can be a consumer of a package.

Pascal

```
pfConsumerDelphi = $00100000;
```

C++

```
pfConsumerDelphi = 0x00100000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfConsumerDelphi is one of these flag values, indicating that Delphi is a legitimate consumer, or user, of a package. The Delphi consumer is the Delphi compiler, linking in the package.

See Also

GetPackageInfo (see page 4253)

39.7.40 SysUtils.pfConsumerMask Constant

Indicates the comsumer of a package.

Pascal

```
pfConsumerMask = $00F00000;
```

C++

```
pfConsumerMask = 0x00F00000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfConsumerMask is a mask that you can use to detect the consumers can use a package. This mask covers these flags : pfConsumerBCB (see page 4427), pfConsumerCompat (see page 4428) and pfConsumerDelphi (see page 4428) flags.

Note: pfConsumerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4253)

pfConsumerBCB (see page 4427)

pfConsumerCompat (see page 4428)

pfConsumerDelphi (see page 4428)

39.7.41 SysUtils.pfDelphi4Produced Constant

Indicates that a package was produced by Delphi 4 or later.

Pascal

```
pfDelphi4Produced = $0C000000;
```

C++

```
pfDelphi4Produced = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfDelphi4Produced is one of these flag values, indicating that the a package was produced by Delphi 4 or later. Packages were introduced in Delphi 3.

See Also

GetPackageInfo (see page 4253)

39.7.42 SysUtils.pfDesignOnly Constant

Indicates that a package was compiled with {$DESIGNONLY ON}.

Pascal

```
pfDesignOnly = $00000002;
```

C++

```
pfDesignOnly = 0x00000002;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfDesignOnly is one of these flag values, indicating that the package was compiled with the $DESIGNONLY compiler directive set ON. This means that the for package is intended for installation in the IDE.

See Also

GetPackageInfo (see page 4253)

39.7.43 SysUtils.pfExeModule Constant

Indicates that a module is an executable.

Pascal

```
pfExeModule = $00000000;
```

C++

```
pfExeModule = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfExeModule is one of these flag values, indicating that the module is an executable.

See Also

GetPackageInfo (see page 4253)

39.7.44 SysUtils.pfIgnoreDupUnits Constant

Indicates that a package was compiled with {$DUPLICATEENABLE ON}.

Pascal

```
pfIgnoreDupUnits = $00000008;
```

C++

```
pfIgnoreDupUnits = 0x00000008;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfIgnoreDupUnits is one of these flag values, indicating that the package was compiled with the $DUPLICATEENABLE compiler directive set ON.

See Also

GetPackageInfo (see page 4253)

39.7.45 SysUtils.pfLibraryModule Constant

Indicates that a module is a library.

Pascal

```
pfLibraryModule = $80000000;
```

C++

```
pfLibraryModule = 0x80000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfLibraryModule is one of these flag values, indicating that the module is a library (.dll file extension).

See Also

GetPackageInfo (see page 4253)

39.7.46 SysUtils.pfModuleTypeMask Constant

Indicates that a package is a module.

Pascal

```
pfModuleTypeMask = $C0000000;
```

C++

```
pfModuleTypeMask = 0xC0000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfModuleTypeMask is a mask that you can use to detect whether the package is a module. This mask is the binary AND of the pfExeModule (see page 4430), pfPackageModule (see page 4432) and pfLibraryModule (see page 4431) flags.

Note: pfModuleTypeMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4253)

39.7.47 SysUtils.pfNeverBuild Constant

Indicates that a package was compiled with the -z flag.

Pascal

```
pfNeverBuild = $00000001;
```

C++

```
pfNeverBuild = 0x00000001;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfNeverBuild is one of these flag values, indicating that the package was compiled with the -z flag.

See Also

GetPackageInfo (see page 4253)

39.7.48 SysUtils.pfPackageModule Constant

Indicates that a module is a package.

Pascal

```
pfPackageModule = $40000000;
```

C++

```
pfPackageModule = 0x40000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfPackageModule is one of these flag values, indicating that the module is a package (.bpl file extension).

See Also

GetPackageInfo (see page 4253)

39.7.49 SysUtils.pfProducerMask Constant

Indicates the producer of a package.

Pascal

```
pfProducerMask = $0C000000;
```

C++

```
pfProducerMask = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfProducerMask is a mask that you can use to detect the producer of a package. This mask is the binary AND of the pfBCB4Produced (see page 4427), pfDelphi4Produced (see page 4429), pfProducerUndefined (see page 4433) and pfV3Produced (see page 4434) flags.

Note: pfProducerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4253)

pfBCB4Produced (see page 4427)

pfDelphi4Produced (see page 4429)

pfProducerUndefined (see page 4433)

pfV3Produced (see page 4434)

39.7.50 SysUtils.pfProducerUndefined Constant

Indicates that a package was produced by an unknown developmemt environment.

Pascal

```
pfProducerUndefined = $04000000;
```

C++

```
pfProducerUndefined = 0x04000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfProducerUndefined is one of these flag values, indicating that the producer of a package was undefined.

See Also

GetPackageInfo (see page 4253)

39.7.51 SysUtils.pfRunOnly Constant

Indicates that a package was compiled with {$RUNONLY ON}.

Pascal

```
pfRunOnly = $00000004;
```

C++

```
pfRunOnly = 0x00000004;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfRunOnly is one of these flag values, indicating that the package was compiled with the $RUNONLY compiler directive set ON. The {$RUNONLY ON} directive causes the package where it occurs to be compiled as runtime only. Packages compiled with {$RUNONLY ON} cannot be installed as design-time packages in the IDE.

See Also

GetPackageInfo (see page 4253)

39.7.52 SysUtils.pfV3Produced Constant

Indicates that a package was produced by Delphi 3 or later.

Pascal

```
pfV3Produced = $00000000;
```

C++

```
pfV3Produced = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about a package. It returns a set of flags.

pfV3Produced is one of these flag values, indicating that the a package was produced by Delphi 3 or later. Packages were introduced in Delphi 3.

See Also

GetPackageInfo (see page 4253)

39.7.53 SysUtils.SecsPerDay Constant

Specifies the number of seconds in a day.

Pascal

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

C++

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

File

SysUtils

Description

SecsPerDay is the number of seconds (86,400) in a day. It is the product of MinsPerDay (see page 4423) and SecsPerMin (see page 4436).

Using this constant rather than hard coding 86400 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerDay (see page 4423)

MinsPerHour (see page 4424)

MSecsPerDay (see page 4425)

MSecsPerSec (see page 4425)

SecsPerMin (see page 4436)

39.7.54 SysUtils.SecsPerHour Constant

Pascal

```
SecsPerHour = SecsPerMin * MinsPerHour;
```

C++

```
SecsPerHour = SecsPerMin * MinsPerHour;
```

File

SysUtils

Description

This is constant SysUtils.SecsPerHour.

39.7.55 SysUtils.SecsPerMin Constant

Specifies the number of seconds in a minute.

Pascal

```
SecsPerMin = 60;
```

C++

```
SecsPerMin = 60;
```

File

SysUtils

Description

SecsPerMin is the number of seconds (60) in a minute.

Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 4409)

HoursPerDay (see page 4420)

MinsPerDay (see page 4423)

MinsPerHour (see page 4424)

MSecsPerDay (see page 4425)

MSecsPerSec (see page 4425)

SecsPerDay (see page 4435)

39.7.56 SysUtils.SwitchChars Constant

Specifies the prefix character(s) for program command line arguments.

Pascal

```
SwitchChars = ['/','-'];
```

C++

```
SwitchChars = ['/','-'];
```

File

SysUtils

Description

When a program is started, an optional 'command line' string may be supplied. Such a string may specify program operational arguments, or 'switches', prefixed by a separator character. SwitchChars defines these separator characters as either '-' or '/' on a Windows system, and as '-' on Linux systems.

See Also

FindCmdLineSwitch (see page 4218)

39.7.57 SysUtils.ufImplicitUnit Constant

Indicates that a unit was implicitly imported into a package.

Pascal

```
ufImplicitUnit = $10;
```

C++

```
ufImplicitUnit = 0x10;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about the units within a package. TPackageInfoProc (see page 4383) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufImplicitUnit is set, then the unit was implicitly imported into a package. This means that the unit is not used in another package required by this package, but is used by one of the units this package contains.

See Also

GetPackageInfo (see page 4253)

TPackageInfoProc (see page 4383)

39.7.58 SysUtils.ufMainUnit Constant

Indicates that a unit is the main one in a project.

Pascal

```
ufMainUnit = $01;
```

C++

```
ufMainUnit = 0x01;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about the units within a package. TPackageInfoProc (see page 4383) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufMainUnit is set, then the unit is a Main unit.

See Also

GetPackageInfo (see page 4253)

TPackageInfoProc (see page 4383)

39.7.59 SysUtils.ufOrgWeakUnit Constant

Pascal

```
ufOrgWeakUnit = $08;
```

C++

```
ufOrgWeakUnit = 0x08;
```

File

SysUtils

Description

This is constant SysUtils.ufOrgWeakUnit.

39.7.60 SysUtils.ufPackageUnit Constant

Indicates that a unit is a package.

Pascal

```
ufPackageUnit = $02;
```

C++

```
ufPackageUnit = 0x02;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about the units within a package. TPackageInfoProc (see page 4383) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufPackageUnit is set, then the unit is a package unit.

See Also

GetPackageInfo (see page 4253)

TPackageInfoProc (see page 4383)

39.7.61 SysUtils.ufWeakPackageUnit Constant

Pascal

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

C++

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

File

SysUtils

Description

This is constant SysUtils.ufWeakPackageUnit.

39.7.62 SysUtils.ufWeakUnit Constant

Indicates that a unit is a weak unit.

Pascal

```
ufWeakUnit = $04;
```

C++

```
ufWeakUnit = 0x04;
```

File

SysUtils

Description

The GetPackageInfo (see page 4253) function returns information about the units within a package. TPackageInfoProc (see page 4383) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufWeakUnit is set, then the unit was compiled with the $WEAKPACKAGEUNIT compiler directive set.

See Also

GetPackageInfo (see page 4253)

TPackageInfoProc (see page 4383)

39.7.63 SysUtils.UnixDateDelta Constant

Specifies the difference between TDateTime (see page 3335) and Lunix date/time origins.

Pascal

```
UnixDateDelta = 25569;
```

C++

```
UnixDateDelta = 25569;
```

File

SysUtils

Description

UnixDateDelta is the number of days (25569) between December 30, 1899 (starting TDateTime (see page 3335) value), and Jamuary 1, 1970 (starting value for Linux date time values). This value is useful for converting date-time values between TDateTime (see page 3335) and Linux file date time values.

See Also

TDateTime (see page 3858)

TTimeStamp (see page 4375)

DateDelta (see page 4409)

40 TabNotBk Namespace

This is namespace TabNotBk.

Topics

Name	Description
Classes (see page 4442)	The following table lists classes in this documentation.
Types (see page 4491)	The following table lists types in this documentation.
Constants (see page 4492)	The following table lists constants in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 4442)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 4480)	TTabPage implements an individual tab page of a tabbed notebook control.

Constants

Name	Description
CM_TABFONTCHANGED (see page 4492)	This is constant TabNotBk.CM_TABFONTCHANGED.

Types

Name	Description
TPageChangeEvent (see page 4491)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4442) control.

40.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 4442)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 4480)	TTabPage implements an individual tab page of a tabbed notebook control.

40.1.1 TTabbedNotebook Class

TTabbedNotebook contains multiple pages, each with its own set of controls.

Class Hierarchy

Pascal

```
TTabbedNotebook = class(TCustomTabControl);
```

C++

```
class TTabbedNotebook : public TCustomTabControl;
```

File

TabNotBk

Description

TTabbedNotebook is provided for backwards compatibility. New (see page 3712) applications should use TPageControl instead.

40.1.1.1 TTabbedNotebook Members

The following tables list the members exposed by TTabbedNotebook.

Protected Methods

	Name	Description
	AlignControls (see page 4445)	Aligns any controls for which the control is the parent within a specified area of the control.
	Change (see page 4445)	Pending tab change event dispatcher.
	Click (see page 4445)	Respond to user click.
	CreateHandle (see page 4446)	Creates underlying screen object.
	CreateParams (see page 4447)	Sets up the window-creation parameters for the tab control.
	GetChildOwner (see page 4447)	Returns the owner of a child component being read from a stream.
	KeyDown (see page 4448)	Respond to key press events.
	Loaded (see page 4449)	Form load event dispatcher.
	ReadState (see page 4449)	Prepares the control for having its properties assigned values from a stream.
	SetPageIndex (see page 4450)	This is SetPageIndex, a member of class TTabbedNotebook.
	ShowControl (see page 4450)	Ensures that a specified child control is visible.

Protected Properties

	Name	Description
	Tabs (see page 4469)	Contains the list of text strings that label the tabs of the tab control.

Public Methods

	Name	Description
	CMTabFontChanged (see page 4446)	This is CMTabFontChanged, a member of class TTabbedNotebook.
	Create (see page 4443)	Constructs a tabbed notebook component.
	Destroy (see page 4444)	Destroys a tabbed notebook component.
	GetChildren (see page 4447)	Calls a specified method for each child of the control.
	GetIndexForPage (see page 4448)	Returns the PageIndex (see page 4459) value of the specified page.
	TabFontChanged (see page 4450)	Invalidates the tabs when the TabFont (see page 4468) property changes.

Public Properties

	Name	Description
	TopFont (see page 4471)	Returns the current value of font used on the tabs of the tabbed notebook.

Published Events

	Name	Description
	OnChange (see page 4473)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 4473)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 4474)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 4474)	Occurs when a control receives the input focus.
	OnExit (see page 4477)	Occurs when the input focus shifts away from one control to another.

Published Properties

	Name	Description
	ActivePage (see page 4451)	Determines which page displays in the tabbed notebook control.
	Align (see page 4452)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4455)	Specifies how the control is anchored to its parent.
	Constraints (see page 4456)	Specifies the size constraints for the control.
	DoubleBuffered (see page 4456)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 4458)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4458)	Controls the attributes of text written on or in the control.
	PageIndex (see page 4459)	Determines which page displays in the tabbed notebook component.
	Pages (see page 4460)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentDoubleBuffered (see page 4460)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4461)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4461)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4466)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 4468)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 4468)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 4470)	This is TabsPerRow, a member of class TTabbedNotebook.
	TabStop (see page 4471)	Determines if the user can tab to a control.
	Visible (see page 4471)	Determines whether the component appears onscreen.

40.1.1.2 TTabbedNotebook.Create Constructor

Constructs a tabbed notebook component.

Pascal

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

C++

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

Description

Call Create TTabbedNotebook (see page 4442) indirectly, using the new keyword, to instantiate a tabbed notebook at runtime. Tabbed notebooks added to forms at design time are created automatically.

The AOwner parameter assigns a value to the notebook's Owner property.

See Also

Owner

40.1.1.3 TTabbedNotebook.Destroy Destructor

Destroys a tabbed notebook component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabbedNotebook();
```

Description

Do not call Destroy directly in an application, Instead, use Free method, which checks that the notebook is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

40.1.1.4 TTabbedNotebook Methods

The methods of the TTabbedNotebook class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 4445)	Aligns any controls for which the control is the parent within a specified area of the control.
	Change (see page 4445)	Pending tab change event dispatcher.
	Click (see page 4445)	Respond to user click.
	CreateHandle (see page 4446)	Creates underlying screen object.
	CreateParams (see page 4447)	Sets up the window-creation parameters for the tab control.
	GetChildOwner (see page 4447)	Returns the owner of a child component being read from a stream.
	KeyDown (see page 4448)	Respond to key press events.
	Loaded (see page 4449)	Form load event dispatcher.
	ReadState (see page 4449)	Prepares the control for having its properties assigned values from a stream.
	SetPageIndex (see page 4450)	This is SetPageIndex, a member of class TTabbedNotebook.
	ShowControl (see page 4450)	Ensures that a specified child control is visible.

Public Methods

	Name	Description
	CMTabFontChanged (see page 4446)	This is CMTabFontChanged, a member of class TTabbedNotebook.
V	GetChildren (see page 4447)	Calls a specified method for each child of the control.
	GetIndexForPage (see page 4448)	Returns the PageIndex (see page 4459) value of the specified page.
	TabFontChanged (see page 4450)	Invalidates the tabs when the TabFont (see page 4468) property changes.

40.1.1.4.1 TTabbedNotebook.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

40.1.1.4.2 TTabbedNotebook.Change Method

Pending tab change event dispatcher.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically when the selected tab changes. Override Change to specify additional processing before the OnChange event handler executes, or to block the OnChange event.

See Also

OnChange

40.1.1.4.3 TTabbedNotebook.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

40.1.1.4.4 TTabbedNotebook.CMTabFontChanged Method

Pascal

```
procedure CMTabFontChanged(var Message: TMessage); message CM_TABFONTCHANGED;
```

C++

```
__fastcall CMTabFontChanged(TMessage Message);
```

Description

This is CMTabFontChanged, a member of class TTabbedNotebook.

40.1.1.4.5 TTabbedNotebook.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle

CreateWnd

Handle

Parent

40.1.1.4.6 TTabbedNotebook.CreateParams Method

Sets up the window-creation parameters for the tab control.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

Tab controls call CreateParams internally when the window for the tab control must be generated. CreateParams overrides the inherited method to specify a tab control window with the current property settings.

See Also

CreateWnd

40.1.1.4.7 TTabbedNotebook.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent, GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner of a component is responsible for destroying it.

See Also

Create

Owner

40.1.1.4.8 TTabbedNotebook.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

40.1.1.4.9 TTabbedNotebook.GetIndexForPage Method

Returns the PageIndex (see page 4459) value of the specified page.

Pascal

```
function GetIndexForPage(const PageName: string): Integer;
```

C++

```
__fastcall int GetIndexForPage(const UnicodeString PageName);
```

Description

The PageIndex (see page 4459) property value is determined by the page's position in the Pages (see page 4460) property array. Specify the name of the page as the value of the PageName parameter. The name you specify must be one of the strings in the Pages (see page 4460) property.

See Also

PageIndex (see page 4459)

Pages (see page 4460)

40.1.1.4.10 TTabbedNotebook.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

40.1.1.4.11 TTabbedNotebook.Loaded Method

Form load event dispatcher.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

The protected Loaded method is called automatically when the form that contains the tab control is loaded into memory. It provides an opportunity for the tab control to perform any initializations after the control and all other components in the form are loaded into memory, but before the first paint operation.

In TCustomTabControl, Loaded ensures that the layout of the tabs accommodate the images specified by the Images property.

See Also

Images

40.1.1.4.12 TTabbedNotebook.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

40.1.1.4.13 TTabbedNotebook.SetPageIndex Method

Pascal

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

C++

```
__fastcall SetPageIndex(int Value);
```

Description

This is SetPageIndex, a member of class TTabbedNotebook.

40.1.1.4.14 TTabbedNotebook.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl, ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show

Parent

40.1.1.4.15 TTabbedNotebook.TabFontChanged Method

Invalidates the tabs when the TabFont (see page 4468) property changes.

Pascal

```
procedure TabFontChanged(Sender: TObject);
```

C++

```
__fastcall TabFontChanged(TObject * Sender);
```

Description

TabFontChanged is called automatically when the TabFont (see page 4468) property changes. It invalidates the notebook so that the tabs are repainted in the new font.

See Also

TabFont (see page 4468)

Invalidate

40.1.1.5 TTabbedNotebook Properties

The properties of the TTabbedNotebook class are listed here.

Protected Properties

	Name	Description
	Tabs (see page 4469)	Contains the list of text strings that label the tabs of the tab control.

Public Properties

	Name	Description
R	TopFont (see page 4471)	Returns the current value of font used on the tabs of the tabbed notebook.

Published Properties

	Name	Description
	ActivePage (see page 4451)	Determines which page displays in the tabbed notebook control.
	Align (see page 4452)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4455)	Specifies how the control is anchored to its parent.
	Constraints (see page 4456)	Specifies the size constraints for the control.
	DoubleBuffered (see page 4456)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 4458)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4458)	Controls the attributes of text written on or in the control.
	PageIndex (see page 4459)	Determines which page displays in the tabbed notebook component.
	Pages (see page 4460)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentDoubleBuffered (see page 4460)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4461)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4461)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4466)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 4468)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 4468)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 4470)	This is TabsPerRow, a member of class TTabbedNotebook.
	TabStop (see page 4471)	Determines if the user can tab to a control.
	Visible (see page 4471)	Determines whether the component appears onscreen.

40.1.1.5.1 TTabbedNotebook.ActivePage Property

Determines which page displays in the tabbed notebook control.

Pascal

```
property ActivePage: string;
```

C++

```
__property UnicodeString ActivePage;
```

Description

The value of ActivePage must be one of the strings contained in the Pages (see page 4460) property.

Note: To indicate the page that appears in the tabbed notebook by index rather than text, use the PageIndex (see page 4459) property instead.

See Also

PageIndex (see page 4459)

Pages (see page 4460)

40.1.1.5.2 TTabbedNotebook.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
```

```
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
```

```
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

40.1.1.5.3 TTabbedNotebook.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

40.1.1.5.4 TTabbedNotebook.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

40.1.1.5.5 TTabbedNotebook.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

40.1.1.5.6 TTabbedNotebook.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

40.1.1.5.7 TTabbedNotebook.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

40.1.1.5.8 TTabbedNotebook.PageIndex Property

Determines which page displays in the tabbed notebook component.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

Changing the PageIndex value changes the page in the control.

Each string in the Pages (see page 4460) property is automatically assigned a PageIndex value when the page is created. The first page receives a value of 0, the second has a value of 1, and so on. If you delete a string from the Pages (see page 4460) property, the PageIndex values are reassigned so that the values always begin with 0 and continue to increase without any gaps between values.

See Also

ActivePage (see page 4451)

Pages (see page 4460)

40.1.1.5.9 TTabbedNotebook.Pages Property

Contains the strings that identify the individual pages of the tabbed notebook control.

Pascal

```
property Pages: TStrings;
```

C++

```
__property TStrings Pages;
```

Description

Tabbed notebook controls create a separate page for each string in the Pages property. For example, if Pages contains three strings, First, Second, and Third, the control has three separate pages. Each string in Pages corresponds to a TTabPage (see page 4480) object whose Caption property is the same as the string.

You can change the active page in a tabbed notebook control with either the ActivePage (see page 4451) or PageIndex (see page 4459) property.

See Also

ActivePage (see page 4451)

PageIndex (see page 4459)

40.1.1.5.10 TTabbedNotebook.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

40.1.1.5.11 TTabbedNotebook.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

40.1.1.5.12 TTabbedNotebook.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
```

```
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

40.1.1.5.13 TTabbedNotebook.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

40.1.1.5.14 TTabbedNotebook.TabFont Property

Determines the font used on the tabs of the tabbed notebook control.

Pascal

```
property TabFont: TFont;
```

C++

```
__property TFont TabFont;
```

Description

Use TabFont to specify the font characteristics used to display the label on the tabs of the notebook.

Note: The text on the selected tab is boldfaced if the selected font for the TabFont property is not also boldfaced.

See Also

TopFont (see page 4471)

40.1.1.5.15 TTabbedNotebook.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

40.1.1.5.16 TTabbedNotebook.Tabs Property

Contains the list of text strings that label the tabs of the tab control.

Pascal

```
property Tabs;
```

C++

```
__property Tabs;
```

Description

Tabs contains a list of strings that label the tabs in the tab control. Tabs can also contain objects associated with the labels, using the TStrings Objects property. Add new strings to the Tabs property to add new tabs to the tab control. Rearrange the list to rearrange the tabs.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
    TabControl1Change(Sender);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
```

```
  several files. To run the example, place a tab control on a
  form and add a memo control that fits into its client area.
  Be sure to leave enough room for the tabs when they appear.
  Then add an OpenDialog and a button to the form.  The
  "Add a file" button adds a single file to the tabcontrol.
  The "Assign files" button removes previous files and can be
  used to assign multiple files.  To assign multiple files,
  use CNTL Select or SHIFT Select to select files in the
  OpenDialog.
  }
  procedure TForm1.Add_a_fileClick(Sender: TObject);
  var index : integer;
  begin
    OpenDialog1.Options :=
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
      Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
      TabControl1Change(Sender);
    end;
  end;

  procedure TForm1.Assign_filesClick(Sender: TObject);
  var index : integer;
  begin
    OpenDialog1.Options :=
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      TabControl1.Tabs.Assign(OpenDialog1.Files);
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

TabIndex

Objects

TTabControl

40.1.1.5.17 TTabbedNotebook.TabsPerRow Property

Pascal

```
property TabsPerRow: Integer;
```

C++

```
__property int TabsPerRow;
```

Description

This is TabsPerRow, a member of class TTabbedNotebook.

40.1.1.5.18 TTabbedNotebook.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

40.1.1.5.19 TTabbedNotebook.TopFont Property

Returns the current value of font used on the tabs of the tabbed notebook.

Pascal

```
property TopFont: TFont;
```

C++

```
__property TFont TopFont;
```

Description

TopFont is a read-only property that reports the TabFont (see page 4468) value.

See Also

TabFont (see page 4468)

40.1.1.5.20 TTabbedNotebook.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
```

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

40.1.1.6 TTabbedNotebook Events

The events of the TTabbedNotebook class are listed here.

Published Events

	Name	Description
	OnChange (see page 4473)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 4473)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 4474)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 4474)	Occurs when a control receives the input focus.
	OnExit (see page 4477)	Occurs when the input focus shifts away from one control to another.

40.1.1.6.1 TTabbedNotebook.OnChange Event

Occurs when the page of a tabbed notebook changes.

Pascal

```
property OnChange: TPageChangeEvent;
```

C++

```
__property TPageChangeEvent OnChange;
```

Description

Write (see page 3789) an OnPage event handler to respond to a page change in a tabbed notebook.

See Also

OnClick (see page 4473)

40.1.1.6.2 TTabbedNotebook.OnClick Event

Occurs when a tab of a tabbed notebook is selected.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write (see page 3789) an OnClick event handler to respond when the user selects a tab in the tabbed notebook. Immediately after the OnClick event occurs, the OnChange (see page 4473) event occurs.

See Also

OnChange (see page 4473)

40.1.1.6.3 TTabbedNotebook.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

40.1.1.6.4 TTabbedNotebook.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

40.1.1.6.5 TTabbedNotebook.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));
```

```
    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

40.1.2 TTabPage Class

TTabPage implements an individual tab page of a tabbed notebook control.

Class Hierarchy

Pascal

```
TTabPage = class(TWinControl);
```

C++

```
class TTabPage : public TWinControl;
```

File

TabNotBk

Description

Each instance of this class holds the controls to be displayed when it is the active page of a tabbed notebook. TTabPage objects are associated with the strings in the corresponding tabbed notebook's Pages property by setting their Caption (see page 4482) property.

See Also

TTabbedNotebook (see page 4442)

TTabbedNotebook (see page 4442)

40.1.2.1 TTabPage Members

The following tables list the members exposed by TTabPage.

Protected Methods

	Name	Description
	ReadState (see page 4481)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	Create (see page 4481)	Creates an instance of TWinControl.

Published Properties

	Name	Description
	Caption (see page 4482)	Specifies a text string that identifies the control to the user.
	Enabled (see page 4485)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 4485)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 4486)	Indicates the position of the control in its parent's tab order.
	Visible (see page 4487)	Determines whether the component appears onscreen.
	Width (see page 4488)	Specifies the horizontal size of the control or form in pixels.

40.1.2.2 TTabPage.Create Constructor

Creates an instance of TWinControl.

Pascal

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

C++

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

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy method to free those resources.

See Also

Brush

TabOrder

CreateParented

InsertComponent

Create

Owner

40.1.2.3 TTabPage Methods

The methods of the TTabPage class are listed here.

Protected Methods

	Name	Description
	ReadState (see page 4481)	Prepares the control for having its properties assigned values from a stream.

40.1.2.3.1 TTabPage.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from

the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

40.1.2.4 TTabPage Properties

The properties of the TTabPage class are listed here.

Published Properties

	Name	Description
	Caption (see page 4482)	Specifies a text string that identifies the control to the user.
	Enabled (see page 4485)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 4485)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 4486)	Indicates the position of the control in its parent's tab order.
	Visible (see page 4487)	Determines whether the component appears onscreen.
	Width (see page 4488)	Specifies the horizontal size of the control or form in pixels.

40.1.2.4.1 TTabPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
```

```
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

40.1.2.4.2 TTabPage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

40.1.2.4.3 TTabPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
```

```
        pRich->HideSelection = false;
    }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

40.1.2.4.4 TTabPage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of

the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

40.1.2.4.5 TTabPage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

40.1.2.4.6 TTabPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
```

```
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

ClientWidth

Height

Left

40.2 Types

The following table lists types in this documentation.

Types

Name	Description
TPageChangeEvent (see page 4491)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4442) control.

40.2.1 TabNotBk.TPageChangeEvent Type

TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4442) control.

Pascal

```
TPageChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TPageChangeEvent)();
```

File

TabNotBk

Description

Sender is the TTabbedNotebook (see page 4442) control.

NewTab is the tab that is about to become the selected or current tab.

AllowChange specifies whether the change to a new page is allowed. If the event handler sets AllowChange to false, a new page change doesn't occur.

40.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_TABFONTCHANGED (see page 4492)	This is constant TabNotBk.CM_TABFONTCHANGED.

40.3.1 TabNotBk.CM_TABFONTCHANGED Constant

Pascal

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

C++

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

File

TabNotBk

Description

This is constant TabNotBk.CM_TABFONTCHANGED.

41 Tabs Namespace

This is namespace Tabs.

Topics

Name	Description
Classes (see page 4494)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4573)	The following table lists structs, records, enums in this documentation.
Types (see page 4576)	The following table lists types in this documentation.

Classes

	Name	Description
	TScroller (see page 4494)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 4499)	This is class Tabs.TTabList.
	TTabSet (see page 4506)	The TTabSet component presents horizontal tabs users can click to initiate actions.

Structs, Records, Enums

	Name	Description
	TEdgePart (see page 4573)	This is record Tabs.TEdgePart.
	TEdgeType (see page 4573)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 4574)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 4574)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 4575)	This is record Tabs.TTabSetTabStyle.

Types

Name	Description
TDrawTabEvent (see page 4576)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.
TMeasureTabEvent (see page 4576)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4506) control.
TTabChangeEvent (see page 4577)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4506) control changes.
TTabSetDrawTabEvent (see page 4577)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.
TTabStyle (see page 4578)	TTabStyle indicates whether a TTabSet (see page 4506) control is owner-drawn.

41.1 Classes

The following table lists classes in this documentation.

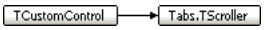
Classes

	Name	Description
	TScroller (see page 4494)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 4499)	This is class Tabs.TTabList.
	TTabSet (see page 4506)	The TTabSet component presents horizontal tabs users can click to initiate actions.

41.1.1 TScroller Class

TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.

Class Hierarchy



Pascal

```
TScroller = class(TCustomControl);
```

C++

```
class TScroller : public TCustomControl;
```

File

Tabs

Description

The TTabSet (see page 4506) object uses TScroller to scroll its visible tabs. TScroller is provided for backward compatibility only. TScroller is similar to an up-down control (TUpDown) with a horizontal orientation, except that it does not work with a companion control.

See Also

TTabSet (see page 4506)

TUpDown

41.1.1.1 TScroller Members

The following tables list the members exposed by TScroller.

Protected Methods

	Name	Description
	Paint (see page 4496)	Paints the image of the TScroller (see page 4494) object into its canvas.

Public Methods

	Name	Description
	Create (see page 4495)	Creates an instance of TScroller (see page 4494).
	Destroy (see page 4496)	Frees the memory associated with the TScroller (see page 4494) object.

Published Events

	Name	Description
	OnClick (see page 4499)	OnClick occurs after the Position (see page 4498) property has been updated in response to the user clicking one of the arrow buttons.

Published Properties

	Name	Description
	Change (see page 4497)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Max (see page 4497)	Specifies the maximum value the Position (see page 4498) property can take.
	Min (see page 4498)	Specifies the minimum value the Position (see page 4498) property can take.
	Position (see page 4498)	Position is the current position in the scroll range.
	ScrollOrientation (see page 4498)	This is ScrollOrientation, a member of class TScroller.

41.1.1.2 TScroller.Create Constructor

Creates an instance of TScroller (see page 4494).

Pascal

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

C++

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

Description

Call TScroller (see page 4494) to create an instance of TScroller (see page 4494) at runtime. TScroller (see page 4494) allocates the memory for the TScroller (see page 4494) object and initializes

Width to 24 and Height to 13

Min (see page 4498) and Max (see page 4497) to 0

Position (see page 4498) to 0

Change (see page 4497) to 1

The AOwner parameter specifies the component (usually the form) that is responsible for freeing the scroller.

See Also

Change (see page 4497)

Height

Max (see page 4497)

Min (see page 4498)

Position (see page 4498)

Width

41.1.1.3 TScroller.Destroy Destructor

Frees the memory associated with the TScroller (see page 4494) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScroller();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the TScroller (see page 4494) reference is nil before calling Destroy.

See Also

Create (see page 4495)

System::TObject::Free (see page 3382)

41.1.1.4 TScroller Methods

The methods of the TScroller class are listed here.

Protected Methods

	Name	Description
	Paint (see page 4496)	Paints the image of the TScroller (see page 4494) object into its canvas.

41.1.1.4.1 TScroller.Paint Method

Paints the image of the TScroller (see page 4494) object into its canvas.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically in response to WM_PAINT messages. Paint paints the arrow buttons, graying them out when Position (see page 4498) cannot be moved in the associated direction.

See Also

Position (see page 4498)

Canvas

41.1.1.5 TScroller Properties

The properties of the TScroller class are listed here.

Published Properties

	Name	Description
	Change (see page 4497)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Max (see page 4497)	Specifies the maximum value the Position (see page 4498) property can take.
	Min (see page 4498)	Specifies the minimum value the Position (see page 4498) property can take.
	Position (see page 4498)	Position is the current position in the scroll range.
	ScrollOrientation (see page 4498)	This is ScrollOrientation, a member of class TScroller.

41.1.1.5.1 TScroller.Change Property

Specifies the number of units to scroll when the user clicks the arrow buttons.

Pascal

```
property Change: Integer;
```

C++

```
__property int Change;
```

Description

Set Change to indicate the number of units between Min (see page 4498) and Max (see page 4497) the Position (see page 4498) moves whenever the user clicks on one of the arrow buttons. Change represents the mapping between the units used to specify Min (see page 4498), Max (see page 4497), and Position (see page 4498), and the logical units represented by the scroller.

For example, if the values of Min (see page 4498), Max (see page 4497), and Position (see page 4498) are given in screen pixels, Change would be the number of pixels to scroll whenever the user clicks the scroll arrow.

See Also

Max (see page 4497)

Min (see page 4498)

Position (see page 4498)

41.1.1.5.2 TScroller.Max Property

Specifies the maximum value the Position (see page 4498) property can take.

Pascal

```
property Max: Longint;
```

C++

```
__property Longint Max;
```

Description

Use Max to set the upper limit of the scrolling range. The Max and Min (see page 4498) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 4497)

Min (see page 4498)

Position (see page 4498)

41.1.1.5.3 TScroller.Min Property

Specifies the minimum value the Position (see page 4498) property can take.

Pascal

```
property Min: Longint;
```

C++

```
__property Longint Min;
```

Description

Use Min to set the lower limit of the scrolling range. The Min and Max (see page 4497) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 4497)

Max (see page 4497)

Position (see page 4498)

41.1.1.5.4 TScroller.Position Property

Position is the current position in the scroll range.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

Read (see page 3727) Position to determine the current position the scroller represents. Position is a value between Min (see page 4498) and Max (see page 4497). Use an OnClick (see page 4499) event handler to notify the object being scrolled when the Position changes.

See Also

Max (see page 4497)

Min (see page 4498)

OnClick (see page 4499)

41.1.1.5.5 TScroller.ScrollOrientation Property

Pascal

```
property ScrollOrientation: TScrollOrientation;
```

C++

```
__property TScrollOrientation ScrollOrientation;
```

Description

This is ScrollOrientation, a member of class TScroller.

41.1.1.6 TScroller Events

The events of the TScroller class are listed here.

Published Events

	Name	Description
	OnClick (see page 4499)	OnClick occurs after the Position (see page 4498) property has been updated in response to the user clicking one of the arrow buttons.

41.1.1.6.1 TScroller.OnClick Event

OnClick occurs after the Position (see page 4498) property has been updated in response to the user clicking one of the arrow buttons.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write (see page 3789) an OnClick event handler to implement the scrolling action of the object that uses the TScroller (see page 4494) to control scrolling. When the user clicks one of the arrow buttons, the Position (see page 4498) gets updated by Change (see page 4497) (unless the Max (see page 4497) or Min (see page 4498) value is hit), and then an OnClick event occurs.

See Also

Change (see page 4497)

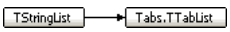
Max (see page 4497)

Min (see page 4498)

Position (see page 4498)

41.1.2 TTabList Class

Class Hierarchy



Pascal

```
TTabList = class(TStringList);
```

C++

```
class TTabList : public TStringList;
```

File

Tabs

Description

This is class Tabs.TTabList.

41.1.2.1 TTabList Members

The following tables list the members exposed by TTabList.

Protected Methods

	Name	Description
	Put (see page 4506)	Changes the value of the string with a specified index.

Public Methods

	Name	Description
	Add (see page 4500)	Adds a new string to the list.
	AddStrings (see page 4503)	Adds a group of strings to the list.
	Clear (see page 4504)	Deletes all the strings from the list.
	Create (see page 4500)	This is Create, a member of class TTabList.
	Delete (see page 4505)	Removes the string specified by the Index parameter.
	Insert (see page 4505)	Inserts a string to the list at the position specified by Index.

41.1.2.2 TTabList.Create Constructor

Pascal

```
constructor Create(const ATabSet: TTabSet);
```

C++

```
__fastcall TTabList(const TTabSet ATabSet);
```

Description

This is Create, a member of class TTabList.

41.1.2.3 TTabList Methods

The methods of the TTabList class are listed here.

Protected Methods

	Name	Description
	Put (see page 4506)	Changes the value of the string with a specified index.

Public Methods

	Name	Description
	Add (see page 4500)	Adds a new string to the list.
	AddStrings (see page 4503)	Adds a group of strings to the list.
	Clear (see page 4504)	Deletes all the strings from the list.
	Delete (see page 4505)	Removes the string specified by the Index parameter.
	Insert (see page 4505)	Inserts a string to the list at the position specified by Index.

41.1.2.3.1 TTabList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const UnicodeString S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates is set to dupError. If Duplicates is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  std::auto_ptr<TStringList> MyList(new TStringList());
  MyList->Add("Animals");
  MyList->Add("Flowers");
  MyList->Add("Ocean");
  MyList->Sort(); // Find will only work on sorted lists
  if (MyList->Find("Flowers", Index))
  {
    ListBox1->Items->AddStrings(MyList.get());
    Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
  }
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TStrings> StringList(new TStringList());
  StringList->Add("This example uses A string List. ");
  StringList->Add("It is the easiest way to add strings");
  StringList->Add("to a combobox's list of strings. ");
  StringList->Add("Always remember TStrings is abstract,");
  StringList->Add("So create a TStringList instead.");
  ComboBox1->Width = 210;
  ComboBox1->Items->Assign(StringList.get());
  ComboBox1->ItemIndex = 0;
}
```

Delphi Examples:

```
{
```

```
  This example uses a list box and a label on a form. When the
  application runs, a string list object is created and three
  strings are added to it. The Find method searches the
  strings to look for a match with the string 'Flowers'. If
  the string is found, all the strings in the string list are
  added to the list box, and the index value of the 'Flowers'
  string appears in the caption of the label control.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    MyList: TStringList;
    Index: Integer;
  begin
    MyList := TStringList.Create;
    try
      MyList.Add('Animals');
      MyList.Add('Flowers');
      MyList.Add('Cars');
      MyList.Sort;   { Find will only work on sorted lists! }
      if MyList.Find('Flowers', Index) then
      begin
        ListBox1.Items.AddStrings(MyList);
        Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
      end;
    finally
      MyList.Free;
    end;
  end;
  {
  This example creates a stringlist, adds elements to it and
  assigns the strings to items of a combo box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    StringList: TStringList;
  begin
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('This example uses A string List.');
        Add('It is the easiest way to add strings');
        Add('to a combobox''s list of strings.');
        Add('Always remember TStrings.Create method');
        Add('is abstract; So use TStringList.Create');
        Add('method instead.');
      end;
      with ComboBox1 do begin
        Width := 210;
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

AddStrings

Append

Delete

Duplicates

Insert

Sorted

41.1.2.3.2 TTabList.AddStrings Method

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); override;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
```

Description

Call AddStrings to add the strings from another TStrings object to the list. If both the source and destination TStrings objects support objects associated with their strings, references to the associated objects will be added as well.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  std::auto_ptr<TStringList> MyList(new TStringList());
  MyList->Add("Animals");
  MyList->Add("Flowers");
  MyList->Add("Ocean");
  MyList->Sort(); // Find will only work on sorted lists
  if (MyList->Find("Flowers", Index))
  {
    ListBox1->Items->AddStrings(MyList.get());
    Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
  }
}
/*
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  std::auto_ptr<TStringList> list(new TStringList());
  list->Add("Plants");
  list->Add("Animals");
  list->Add("Minerals");
  list->Sorted = true;
  ListBox1->Items->AddStrings(list.get());
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
  finally
    MyList.Free;
  end;
end;
{
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
begin
  MyList := TStringList.Create;
  MyList.Add('Plants');
  MyList.Add('Animals');
  MyList.Add('Minerals');
  MyList.Sorted := True;
  ListBox1.Items.AddStrings(MyList);
  MyList.Free;
end;
```

See Also

Add

AddObject

Assign

Count

Objects

41.1.2.3.3 TTabList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete

Objects

Strings

41.1.2.3.4 TTabList.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear

IndexOf

InsertObject

Objects

Strings

41.1.2.3.5 TTabList.Insert Method

Inserts a string to the list at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject method instead.

Note: If the list is sorted, calling Insert or InsertObject will raise an EListError exception. Use Add or AddObject with sorted lists.

See Also

Add

AddObject

EListError

InsertObject

Sorted

Strings

41.1.2.3.6 TTabList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

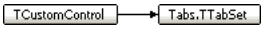
Strings

Objects

41.1.3 TTabSet Class

The TTabSet component presents horizontal tabs users can click to initiate actions.

Class Hierarchy



Pascal

```
TTabSet = class(TCustomControl);
```

C++

```
class TTabSet : public TCustomControl;
```

File

Tabs

Description

Tab set controls are commonly used to display tabbed pages within a dialog box. TTabSet is provided for backward compatibility. Use TTabControl component in 32-bit Windows applications.

You create a set of tabs for the tab set control when you specify a list of strings as the value of the Tabs (see page 4542) property. One tab is created for each string.

To determine which tab is currently selected or to use code to select a tab, use the TabIndex (see page 4542) property. To find out which tab is the first visible tab in the tab set control or to make a tab the first visible tab, use the FirstIndex (see page 4530) property.

41.1.3.1 TTabSet Members

The following tables list the members exposed by TTabSet.

Protected Methods

	Name	Description
	CanChange (see page 4510)	This is CanChange, a member of class TTabSet.
	CMHintShow (see page 4510)	This is CMHintShow, a member of class TTabSet.
V	CreateParams (see page 4511)	Initializes a window-creation parameter data structure.
V	DrawTab (see page 4511)	This is DrawTab, a member of class TTabSet.
V	GetImageIndex (see page 4512)	This is GetImageIndex, a member of class TTabSet.
V	MeasureTab (see page 4513)	This is MeasureTab, a member of class TTabSet.
V	MouseDown (see page 4514)	OnMouseDown event dispatcher.
V	Notification (see page 4515)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 4515)	Renders the image of a custom control.
V	SetTabIndex (see page 4516)	This is SetTabIndex, a member of class TTabSet.

Protected Properties

	Name	Description
R	Scroller (see page 4538)	This is Scroller, a member of class TTabSet.

Public Methods

	Name	Description
V	Create (see page 4509)	Constructs a tab set control.
V	Destroy (see page 4509)	Frees the memory associated with the TTabSet (see page 4506) object.
V	GetChildren (see page 4511)	Calls a specified method for each child of the control.
	ItemAtPos (see page 4512)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 4512)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 4513)	Returns the width, in pixels, of a specified tab in the set.
	MinClientRect (see page 4513)	This is the overview for the MinClientRect method overload.
	SelectNext (see page 4516)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

Public Properties

	Name	Description
	Canvas (see page 4522)	Specifies the TCanvas object that presents a drawing surface for the control.
	FirstIndex (see page 4530)	Specifies the tab that appears in the leftmost visible position in the tab set control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 4545)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 4546)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 4547)	Occurs just before a new tab is selected.
	OnClick (see page 4547)	Occurs when the user clicks the control.
	OnDragDrop (see page 4549)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4552)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4555)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 4555)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4556)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4557)	Occurs when a control receives the input focus.
	OnExit (see page 4559)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4562)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 4562)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 4563)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4564)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4566)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4566)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4566)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4569)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4571)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4572)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4517)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4521)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4521)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 4522)	Determines the background color of the tab set control.
	Constraints (see page 4524)	Specifies the size constraints for the control.
	DitherBackground (see page 4525)	Determines whether the background color set with the BackgroundColor (see page 4522) property is dithered.
	DoubleBuffered (see page 4525)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4527)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4527)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4528)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4529)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 4530)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 4530)	Controls the attributes of text written on or in the control.
	Images (see page 4532)	This is Images, a member of class TTabSet.
	ParentBackground (see page 4532)	Determines whether control uses parent's theme background.
	ParentDoubleBuffered (see page 4532)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4532)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4533)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 4538)	Determines the color of the selected tab in the tab set control.

	ShowHint (see page 4538)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 4540)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 4540)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 4540)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 4541)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 4541)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 4542)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 4542)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 4542)	Contains the list of text strings that appear on the tabs.
	UnselectedColor (see page 4543)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 4543)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 4544)	Contains the number of tabs currently visible in the tab set control.

41.1.3.2 TTabSet.Create Constructor

Constructs a tab set control.

Pascal

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

C++

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

Description

Create sets the initial values of the control, including creating a TTabList (see page 4499) object that holds the list of tabs for the control. The AOwner parameter specifies the component (usually a form) that is responsible for freeing the tab set.

See Also

Destroy (see page 4509)

41.1.3.3 TTabSet.Destroy Destructor

Frees the memory associated with the TTabSet (see page 4506) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabSet();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the tab set is not nil before calling Destroy. Destroy frees the TTabList (see page 4499) object that holds the list of tabs.

See Also

Create (see page 4509)

System::TObject::Free (see page 3382)

41.1.3.4 TTabSet Methods

The methods of the TTabSet class are listed here.

Protected Methods

	Name	Description
	CanChange (see page 4510)	This is CanChange, a member of class TTabSet.
	CMHintShow (see page 4510)	This is CMHintShow, a member of class TTabSet.
V	CreateParams (see page 4511)	Initializes a window-creation parameter data structure.
V	DrawTab (see page 4511)	This is DrawTab, a member of class TTabSet.
V	GetImageIndex (see page 4512)	This is GetImageIndex, a member of class TTabSet.
V	MeasureTab (see page 4513)	This is MeasureTab, a member of class TTabSet.
V	MouseDown (see page 4514)	OnMouseDown event dispatcher.
V	Notification (see page 4515)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 4515)	Renders the image of a custom control.
V	SetTabIndex (see page 4516)	This is SetTabIndex, a member of class TTabSet.

Public Methods

	Name	Description
V	GetChildren (see page 4511)	Calls a specified method for each child of the control.
	ItemAtPos (see page 4512)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 4512)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 4513)	Returns the width, in pixels, of a specified tab in the set.
	MinClientRect (see page 4513)	This is the overview for the MinClientRect method overload.
	SelectNext (see page 4516)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

41.1.3.4.1 TTabSet.CanChange Method

Pascal

```
function CanChange(NewIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CanChange(int NewIndex);
```

Description

This is CanChange, a member of class TTabSet.

41.1.3.4.2 TTabSet.CMHintShow Method

Pascal

```
procedure CMHintShow(var Message: TCMHintShow); message CM_HINTSHOW;
```

C++

```
__fastcall CMHintShow(TCMHintShow Message);
```

Description

This is CMHintShow, a member of class TTabSet.

41.1.3.4.3 TTabSet.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

41.1.3.4.4 TTabSet.DrawTab Method

Pascal

```
procedure DrawTab(TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean); virtual;
```

C++

```
virtual __fastcall DrawTab(TCanvas TabCanvas, TRect R, int Index, Boolean Selected);
```

Description

This is DrawTab, a member of class TTabSet.

41.1.3.4.5 TTabSet.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

41.1.3.4.6 TTabSet.GetImageIndex Method

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

This is GetImageIndex, a member of class TTabSet.

41.1.3.4.7 TTabSet.ItemAtPos Method

Returns the index of the tab indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; IgnoreTabHeight: Boolean = False): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean IgnoreTabHeight = False);
```

Description

Call ItemAtPos to identify the tab under the point specified by the Pos (see page 3720) parameter. Pos (see page 3720) is the point in the control in client coordinates. If the returned index is 0, the tab is the first tab in the tab set, if the index is 1, the tab is the second tab, and so on. ItemAtPos returns –1 if there is no tab at the specified point.

See Also

ItemRect (see page 4512)

41.1.3.4.8 TTabSet.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to obtain the rectangle that surrounds the tab when you are drawing on an owner-draw tab set.

See Also

ItemAtPos (see page 4512)

Style (see page 4541)

OnDrawTab (see page 4555)

OnMeasureTab (see page 4562)

41.1.3.4.9 TTabSet.ItemWidth Method

Returns the width, in pixels, of a specified tab in the set.

Pascal

```
function ItemWidth(Index: Integer): Integer;
```

C++

```
__fastcall int ItemWidth(int Index);
```

Description

Use ItemWidth to obtain the width, in pixels of a specified tab.

Index is the index of the tab, where 0 indicates the first tab, 1 indicates the second tab, and so on.

See Also

ItemAtPos (see page 4512)

ItemRect (see page 4512)

41.1.3.4.10 TTabSet.MeasureTab Method

Pascal

```
procedure MeasureTab(Index: Integer; var TabWidth: Integer); virtual;
```

C++

```
virtual __fastcall MeasureTab(int Index, int TabWidth);
```

Description

This is MeasureTab, a member of class TTabSet.

41.1.3.4.11 MinClientRect Method

This is the overview for the MinClientRect method overload.

Overload List

	Name	Description
	TTabSet.MinClientRect () (see page 4513)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Boolean) (see page 4513)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Integer, Boolean) (see page 4513)	Returns the boundaries that are required to display the tabs in the tab set.

41.1.3.4.11.1 TTabSet.MinClientRect Method ()

Returns the boundaries that are required to display the tabs in the tab set.

Pascal

```
function MinClientRect: TRect; overload;
function MinClientRect(IncludeScroller: Boolean): TRect; overload;
function MinClientRect(TabCount: Integer; IncludeScroller: Boolean = False): TRect; overload;
```

C++

```
__fastcall TRect MinClientRect();
__fastcall TRect MinClientRect(Boolean IncludeScroller);
__fastcall TRect MinClientRect(int TabCount, Boolean IncludeScroller = False);
```

Description

Use MinClientRect to obtain the minimum size that the tab set's client area must be to display its tabs and (if specified) a scroller for scrolling hidden tabs. MinClientRect calculates the width based on the widths of all the tabs up to a specified count, adding in the values of StartMargin (see page 4540) and EndMargin (see page 4530) and the space that appears between tabs. If requested, it also adds the width of a scroller for the tabs. It calculates the height based on the value of the TabHeight (see page 4541) property.

TabCount specifies the number of tabs to display. MinClientRect uses the widths of these tabs to calculate the width of the client rectangle. If you use a syntax that does not include a TabCount parameter, MinClientRect assumes that all tabs must be visible.

IncludeScroller indicates whether the client rectangle must include space for the scroller control that allows users to scroll the tabs, displaying any tabs that are not included in TabCount. If IncludeScroller is false, or if this parameter is omitted, MinClientRect does not include the width for the scroller in the resulting rectange.

MinClientRect returns the dimensions, in pixels, needed to display the specified number of tabs and, if specified, the scroller. The returned rectangle has an origin of 0,0.

See Also

TabHeight (see page 4541)

ItemWidth (see page 4513)

StartMargin (see page 4540)

EndMargin (see page 4530)

41.1.3.4.12 TTabSet.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which

it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

41.1.3.4.13 TTabSet.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

41.1.3.4.14 TTabSet.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

41.1.3.4.15 TTabSet.SelectNext Method

Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

Pascal

```
procedure SelectNext(Direction: Boolean);
```

C++

```
__fastcall SelectNext(Boolean Direction);
```

Description

The value of the Direction parameters determines if the tab to the left or right is selected. If Direction is true, the tab to the right is selected. If Direction is false, the tab to the left is selected. When the last tab in either direction is selected, calling SelectNext using the same direction wraps around to the beginning of the tab order. For example, if your application has three tabs, First, Second, and Third, and Third is the current tab, calling SelectNext(true) selects First. Likewise, if First is the current tab, SelectNext(false) selects Third.

When SelectNext is called, the OnClick (see page 4547) event of the tab set occurs, followed by the OnChange (see page 4547) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 4542)

OnChange (see page 4547)

41.1.3.4.16 TTabSet.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TTabSet.

41.1.3.5 TTabSet Properties

The properties of the TTabSet class are listed here.

Protected Properties

	Name	Description
	Scroller (see page 4538)	This is Scroller, a member of class TTabSet.

Public Properties

	Name	Description
	Canvas (see page 4522)	Specifies the TCanvas object that presents a drawing surface for the control.
	FirstIndex (see page 4530)	Specifies the tab that appears in the leftmost visible position in the tab set control.

Published Properties

	Name	Description
	Align (see page 4517)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4521)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4521)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 4522)	Determines the background color of the tab set control.
	Constraints (see page 4524)	Specifies the size constraints for the control.
	DitherBackground (see page 4525)	Determines whether the background color set with the BackgroundColor (see page 4522) property is dithered.
	DoubleBuffered (see page 4525)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4527)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4527)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4528)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4529)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 4530)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 4530)	Controls the attributes of text written on or in the control.
	Images (see page 4532)	This is Images, a member of class TTabSet.
	ParentBackground (see page 4532)	Determines whether control uses parent's theme background.
	ParentDoubleBuffered (see page 4532)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4532)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4533)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 4538)	Determines the color of the selected tab in the tab set control.
	ShowHint (see page 4538)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 4540)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 4540)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 4540)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 4541)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 4541)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 4542)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 4542)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 4542)	Contains the list of text strings that appear on the tabs.
	UnselectedColor (see page 4543)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 4543)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 4544)	Contains the number of tabs currently visible in the tab set control.

41.1.3.5.1 TTabSet.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
```

```
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
```

```
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
```

```
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

41.1.3.5.2 TTabSet.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

41.1.3.5.3 TTabSet.AutoScroll Property

Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is false, the application can still access tabs that aren't visible by using the FirstIndex (see page 4530) or TabIndex (see page 4542) properties at design time or runtime, but the user can't click on the tabs with the mouse at runtime.

See Also

FirstIndex (see page 4530)

TabIndex (see page 4542)

41.1.3.5.4 TTabSet.BackgroundColor Property

Determines the background color of the tab set control.

Pascal

```
property BackgroundColor: TColor;
```

C++

```
__property TColor BackgroundColor;
```

Description

Use BackgroundColor to indicate the color that the tab set control uses to paint the area between the tabs and the border of the control.

See Also

DitherBackground (see page 4525)

SelectedColor (see page 4538)

Color

41.1.3.5.5 TTabSet.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
```

```
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  */
  void __fastcall TForm1::DrawGrid1DrawCell(
    TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
  {
    long index = ARow * DrawGrid1->ColCount + ACol;
    DrawGrid1->Canvas->Brush->Color = clBackground;
    DrawGrid1->Canvas->FillRect(Rect);
    if (State.Contains(gdFocused))
    {
      DrawGrid1->Canvas->DrawFocusRect(Rect);
      Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
    }
    ImageList1->Draw(
      DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  }

  void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
        bool &CanSelect)
  {
    CanSelect= True;
  }
  /*
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  */
  {
    TPoint APoint = Point(X, Y);
    int Index = ListBox1->ItemAtPos(APoint, True);
    TStringGrid *myStringGrid;
    TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
    if (String(ClassRef->ClassName()) == "TStringGrid")
    {
      myStringGrid = dynamic_cast<TStringGrid *>(ListBox1->Items->Objects[Index]);
      // make sure we are using the TControls canvas and not the from's!
      myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
      myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
  myStringGrid->ClientRect.Bottom);
      myStringGrid = NULL;
    }
  }
```

Delphi Examples:

```
  {
  The following code uses the bitmaps in an image list
  component to draw the contents of each cell in a draw grid.
  It draws a focus rectangle around the cell that has focus.
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
```

```
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint

41.1.3.5.6 TTabSet.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

41.1.3.5.7 TTabSet.DitherBackground Property

Determines whether the background color set with the BackgroundColor (see page 4522) property is dithered.

Pascal

```
property DitherBackground: Boolean;
```

C++

```
__property Boolean DitherBackground;
```

Description

Dithering means the background is lightened by 50%, which is intended to make the tabs easier to see. If DitherBackground is true, the tab set control background is dithered. If it is false, there is no dithering.

See Also

BackgroundColor (see page 4522)

SelectedColor (see page 4538)

41.1.3.5.8 TTabSet.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

41.1.3.5.9 TTabSet.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

41.1.3.5.10 TTabSet.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

41.1.3.5.11 TTabSet.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

41.1.3.5.12 TTabSet.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

41.1.3.5.13 TTabSet.EndMargin Property

Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.

Pascal

```
property EndMargin: Integer;
```

C++

```
__property int EndMargin;
```

Description

EndMargin determines how far in pixels the rightmost tab appears from the right edge of the tab set control. Together with the StartMargin (see page 4540) property, EndMargin can play a role in determining how many tabs can fit within the tab set control.

If AutoScroll (see page 4521) is true and scroll buttons appear in the tab set control, EndMargin determines how far in pixels the rightmost tab appears from the left edge of the scroll buttons, rather than the edge of the tab set control.

See Also

AutoScroll (see page 4521)

EndMargin

StartMargin (see page 4540)

41.1.3.5.14 TTabSet.FirstIndex Property

Specifies the tab that appears in the leftmost visible position in the tab set control.

Pascal

```
property FirstIndex: Integer;
```

C++

```
__property int FirstIndex;
```

Description

Any tabs with a lower value in the FirstIndex property scroll to the left in the tab set control and don't appear until the user scrolls the tabs.

The default value of FirstIndex is 0 indicating that the tab with an index of 0 is in the leftmost position. For example, if you have three tabs labeled First, Second, and Third with TabIndex (see page 4542) values of 0, 1, and 2, respectively, First appears first, by default, because it has an index value of 0. If you want to shift the tabs so the Second or Third tab appears leftmost in the tab set control, change the FirstIndex value to 1 or 2.

See Also

Tabs (see page 4542)

41.1.3.5.15 TTabSet.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

41.1.3.5.16 TTabSet.Images Property

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

This is Images, a member of class TTabSet.

41.1.3.5.17 TTabSet.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

41.1.3.5.18 TTabSet.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

41.1.3.5.19 TTabSet.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

41.1.3.5.20 TTabSet.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

41.1.3.5.21 TTabSet.Scroller Property

Pascal

```
property Scroller: TScroller;
```

C++

```
__property TScroller Scroller;
```

Description

This is Scroller, a member of class TTabSet.

41.1.3.5.22 TTabSet.SelectedColor Property

Determines the color of the selected tab in the tab set control.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

SelectedColor determines the color of the selected tab in the tab set control.

See Also

BackgroundColor (see page 4522)

DitherBackground (see page 4525)

Color

41.1.3.5.23 TTabSet.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

41.1.3.5.24 TTabSet.ShrinkToFit Property

Pascal

```
property ShrinkToFit: Boolean;
```

C++

```
__property Boolean ShrinkToFit;
```

Description

This is ShrinkToFit, a member of class TTabSet.

41.1.3.5.25 TTabSet.SoftTop Property

Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.

Pascal

```
property SoftTop: Boolean;
```

C++

```
__property Boolean SoftTop;
```

Description

When SoftTop is false (the default), a line appears at the edge of the non-current tabs. When SoftTop is true, this line is not drawn, softening the border of the tab set.

41.1.3.5.26 TTabSet.StartMargin Property

Determines how far, in pixels, the first tab appears from the left edge of the tab set control.

Pascal

```
property StartMargin: Integer;
```

C++

```
__property int StartMargin;
```

Description

Together with the EndMargin (see page 4530) property, StartMargin determines how many tabs fit in the tab set control.

See Also

EndMargin (see page 4530)

41.1.3.5.27 TTabSet.Style Property

Determines whether the tab set is owner-drawn.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Owner-draw tabs can display objects other than strings, such as graphical images. Owner-draw tabs require more programming, however, because the application needs information on how to render the image for each tab in the tab set control.

Each time an item is displayed in an owner-drawn tab, two events occur. The first is the OnMeasureTab (see page 4562) event. In the OnMeasureTab (see page 4562) event handler, you write the code that calculates the width of the tab needed to hold the text or graphical image. After the OnMeasureTab (see page 4562) event, the OnDrawTab (see page 4555) event occurs. The code you write for the OnDrawTab (see page 4555) event draws the tab and its contents using the width found with the OnMeasureTab (see page 4562) event and the height specified as the value of the TabHeight (see page 4541) property.

Note: The TTabStyle (see page 4578) type defined in the tabs unit for the Style property differs from the TTabStyle (see page 4578) type defined in the comctrls unit. Do not to use or include the comctrls unit in source files where you are working with the tab control. Doing so can lead to compiler errors because of type mismatches.

See Also

TabHeight (see page 4541)

OnDrawTab (see page 4555)

OnMeasureTab (see page 4562)

41.1.3.5.28 TTabSet.TabHeight Property

Determines the height of the tabs in an owner-drawn tab set control.

Pascal

```
property TabHeight: Integer;
```

C++

```
__property int TabHeight;
```

Description

If the Style (see page 4541) property value is tsStandard, the TabHeight property value is ignored. If the Style (see page 4541) property value is tsOwnerDraw, TabHeight is the height of the tabs.

See Also

Style (see page 4541)

41.1.3.5.29 TTabSet.TabIndex Property

Determines which tab of a tab set control is currently selected.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

The value of TabIndex can be from –1 to one less than the number of strings in the Tabs (see page 4542) property array (Count – 1). TabIndex is an index to the Tabs (see page 4542) array property: A value of 0 indicates the first tab in the tab set control, 1 is the second tab, and so on. If no tabs are selected, TabIndex has a value of –1.

When a value is assigned to TabIndex, the OnClick (see page 4547) event for the tab set occurs, followed by the OnChange (see page 4547) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 4542)

OnChange (see page 4547)

OnClick

41.1.3.5.30 TTabSet.TabPosition Property

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

This is TabPosition, a member of class TTabSet.

41.1.3.5.31 TTabSet.Tabs Property

Contains the list of text strings that appear on the tabs.

Pascal

```
property Tabs: TStrings;
```

C++

```
__property TStrings Tabs;
```

Description

For each string in the Tabs property, a new tab is created to display the text string. At design time, you enter the text strings you want to appear on the tabs using the String List editor. Double-click the value column of the Tabs property to display the String List editor. You can also change strings in the Tabs property at runtime.

See Also

VisibleTabs (see page 4544)

41.1.3.5.32 TTabSet.UnselectedColor Property

Determines the color of the tabs that aren't currently selected in the tab set control.

Pascal

```
property UnselectedColor: TColor;
```

C++

```
__property TColor UnselectedColor;
```

Description

UnselectedColor determines the color of the tabs that aren't currently selected in the tab set control.

See Also

BackgroundColor (see page 4522)

SelectedColor (see page 4538)

41.1.3.5.33 TTabSet.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

41.1.3.5.34 TTabSet.VisibleTabs Property

Contains the number of tabs currently visible in the tab set control.

Pascal

```
property VisibleTabs: Integer;
```

C++

```
__property int VisibleTabs;
```

Description

Use VisibleTabs to determine how many tabs are visible in the tab set.

See Also

Tabs (see page 4542)

41.1.3.6 TTabSet Events

The events of the TTabSet class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 4545)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 4546)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 4547)	Occurs just before a new tab is selected.
	OnClick (see page 4547)	Occurs when the user clicks the control.
	OnDragDrop (see page 4549)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4552)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4555)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 4555)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4556)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4557)	Occurs when a control receives the input focus.
	OnExit (see page 4559)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4562)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 4562)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 4563)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4564)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4566)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4566)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4566)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4569)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4571)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4572)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

41.1.3.6.1 TTabSet.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

41.1.3.6.2 TTabSet.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

41.1.3.6.3 TTabSet.OnChange Event

Occurs just before a new tab is selected.

Pascal

```
property OnChange: TTabChangeEvent;
```

C++

```
__property TTabChangeEvent OnChange;
```

Description

OnChange occurs just as the TabIndex (see page 4542) value is about to change. To prevent the TabIndex (see page 4542) value from changing, set AllowChange to false in the OnChange event handler.

41.1.3.6.4 TTabSet.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

41.1.3.6.5 TTabSet.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

41.1.3.6.6 TTabSet.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
```

```
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

41.1.3.6.7 TTabSet.OnDrawTab Event

Occurs when a tab in an owner-drawn tab set control needs to redisplay.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

OnDrawTab occurs when a tab redisplays if the Style (see page 4541) property value is tsOwnerDraw. For example, it happens when the user selects a tab or scrolls the tabs.

You must write the code in the OnDrawTab event handler to draw the tab.

OnDrawTab occurs just after the OnMeasureTab (see page 4562) event, which contains the code to calculate the width of the tab needed. The height of the tab is determined by the value of the TabHeight (see page 4541) property of the tab set control. The code you write in the OnDrawTab event handler, therefore, must use the width determined with the OnMeasureTab (see page 4562) event to draw the tab.

See Also

TabHeight (see page 4541)

OnMeasureTab (see page 4562)

Style (see page 4541)

41.1.3.6.8 TTabSet.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

41.1.3.6.9 TTabSet.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

41.1.3.6.10 TTabSet.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}
```

```
void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
```

```
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

41.1.3.6.11 TTabSet.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit

event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
```

```
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

41.1.3.6.12 TTabSet.OnGetImageIndex Event

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

This is OnGetImageIndex, a member of class TTabSet.

41.1.3.6.13 TTabSet.OnMeasureTab Event

Occurs when an application needs to redisplay a tab in an owner-draw tab set.

Pascal

```
property OnMeasureTab: TMeasureTabEvent;
```

C++

```
__property TMeasureTabEvent OnMeasureTab;
```

Description

OnMeasureTab occurs only if the Style (see page 4541) property is tsOwnerDraw. In the OnMeasureTab event handler, write the code to calculate the width needed to draw the tab. Your code is responsible for calculating and returning the tab width, depending on what you have drawn in the tab (if the tab is of Style (see page 4541) tsOwnerDraw). After the OnMeasureTab

event occurs, the OnDrawTab (see page 4555) event occurs, when the OnDrawTab (see page 4555) event handler draws the tab using the width calculated in OnMeasureTab.

See Also

TabHeight (see page 4541)

OnDrawTab (see page 4555)

41.1.3.6.14 TTabSet.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

41.1.3.6.15 TTabSet.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
```

```
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

41.1.3.6.16 TTabSet.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

41.1.3.6.17 TTabSet.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

41.1.3.6.18 TTabSet.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

41.1.3.6.19 TTabSet.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

41.1.3.6.20 TTabSet.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

41.1.3.6.21 TTabSet.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

41.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgePart (see page 4573)	This is record Tabs.TEdgePart.
	TEdgeType (see page 4573)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 4574)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 4574)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 4575)	This is record Tabs.TTabSetTabStyle.

41.2.1 Tabs.TEdgePart Enumeration

Pascal

```
TEdgePart = (
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
);
```

C++

```
enum TEdgePart {
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
};
```

File

Tabs

Description

This is record Tabs.TEdgePart.

41.2.2 Tabs.TEdgeType Enumeration

Pascal

```
TEdgeType = (
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
```

```
);
```

C++

```
enum TEdgeType {
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
};
```

File

Tabs

Description

This is record Tabs.TEdgeType.

41.2.3 Tabs.TScrollBtn Enumeration

Pascal

```
TScrollBtn = (
  sbLeft,
  sbRight
);
```

C++

```
enum TScrollBtn {
  sbLeft,
  sbRight
};
```

File

Tabs

Description

This is record Tabs.TScrollBtn.

41.2.4 Tabs.TScrollOrientation Enumeration

Pascal

```
TScrollOrientation = (
  soLeftRight,
  soUpDown
);
```

C++

```
enum TScrollOrientation {
  soLeftRight,
  soUpDown
};
```

File

Tabs

Description

This is record Tabs.TScrollOrientation.

41.2.5 Tabs.TTabSetTabStyle Enumeration

Pascal

```
TTabSetTabStyle = (
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
);
```

C++

```
enum TTabSetTabStyle {
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
};
```

File

Tabs

Description

This is record Tabs.TTabSetTabStyle.

41.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDrawTabEvent (see page 4576)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.
TMeasureTabEvent (see page 4576)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4506) control.
TTabChangeEvent (see page 4577)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4506) control changes.
TTabSetDrawTabEvent (see page 4577)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.
TTabStyle (see page 4578)	TTabStyle indicates whether a TTabSet (see page 4506) control is owner-drawn.

41.3.1 Tabs.TDrawTabEvent Type

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.

Pascal

```
TDrawTabEvent = TTabSetDrawTabEvent;
```

C++

```
TTabSetDrawTabEvent TDrawTabEvent;
```

File

Tabs

Description

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.

Note: The TDrawTabEvent type defined in the tabs unit for the OnDrawTab event handler differs from the TDrawTabEvent type defined in the comctrls unit. Do not use the comctrls unit in source files where you are working with the tab control.

41.3.2 Tabs.TMeasureTabEvent Type

TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4506) control.

Pascal

```
TMeasureTabEvent = procedure (Sender: TObject; Index: Integer; var TabWidth: Integer) of
object;
```

C++

```
(Sender: TObject; Index: Integer; var TabWidth: Integer) ( TMeasureTabEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 4506) control whose tab is about to be drawn.

Index identifies the position of the tab in the tab set control.

TabWidth is the width of the tab.

41.3.3 Tabs.TTabChangeEvent Type

TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4506) control changes.

Pascal

```
TTabChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of
object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TTabChangeEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 4506) control whose tab is about to change.

NewTab is the tab that is about to become the selected tab.

AllowChange determines whether the change is permitted. If the event handler sets AllowChange to false, the user won't be able to select the new tab, in effect disabling it.

41.3.4 Tabs.TTabSetDrawTabEvent Type

TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4506) control.

Pascal

```
TTabSetDrawTabEvent = procedure (Sender: TObject; TabCanvas: TCanvas; R: TRect; Index:
Integer; Selected: Boolean) of object;
```

C++

```
(Sender: TObject; TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean) (
TTabSetDrawTabEvent)();
```

File

Tabs

Description

TabCanvas is the canvas on which the item is drawn.

Index is the position of the tab in the tab set control.

R is the area in the tab where the item is to be drawn.

Selected indicates whether the tab is currently selected.

41.3.5 Tabs.TTabStyle Type

TTabStyle indicates whether a TTabSet (see page 4506) control is owner-drawn.

Pascal

```
TTabStyle = TTabSetTabStyle;
```

C++

```
TTabSetTabStyle TTabStyle;
```

File

Tabs

Description

These are the possible values:

Value	Meaning
tsStandard	Each tab has the standard size and look.

tsOwnerDrawEach tab has the height specified with the TabHeight property and width needed to hold the text or glyph.

42 TConnect Namespace

This is namespace TConnect.

Topics

Name	Description
Classes (see page 4580)	The following table lists classes in this documentation.

Classes

	Name	Description
	TLocalConnection (see page 4580)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

42.1 Classes

The following table lists classes in this documentation.

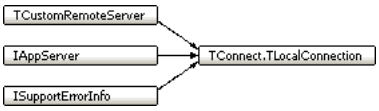
Classes

	Name	Description
	TLocalConnection (see page 4580)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

42.1.1 TLocalConnection Class

TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

Class Hierarchy



Pascal

```
TLocalConnection = class(TCustomRemoteServer, IAppServer, ISupportErrorInfo);
```

C++

```
class TLocalConnection : public TCustomRemoteServer, public IAppServer, public
ISupportErrorInfo;
```

File

TConnect

Description

Use TLocalConnection to manage the connection between one or more client datasets or XML brokers and the providers that reside in the same application. When working with local providers, it is not necessary to set the RemoteServer property of a client dataset or XML broker to a TLocalConnection component. However, using TLocalConnection lets your application communicate directly with local providers through the AppServer (see page 4591) property. This provides access to methods that would otherwise be unavailable, and makes it easier to scale up to a multi-tiered application at a later time.

TLocalConnection implements the IAppServer interface.

Warning: TLocalConnection only links clients to providers that are instances of TDataSetProvider.

See Also

TCustomClientDataSet

TSocketConnection (see page 176)

TWebConnection (see page 209)

TDCOMConnection

TCustomClientDataSet

IAppServer

TXMLBroker (see page 7255)

TDataSetProvider

TSoapConnection (see page 1128)

Specifying a Provider

Communicating with the Client Dataset

42.1.1.1 TLocalConnection Members

The following tables list the members exposed by TLocalConnection.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 4583)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 4583)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 4584)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 4585)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 4586)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 4586)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 4587)	Returns information from a specified record of the provider's dataset.
V	GetConnected (see page 4588)	Returns the value of the Connected property.
V	GetProvider (see page 4588)	This is GetProvider, a member of class TLocalConnection.
	InterfaceSupportsErrorInfo (see page 4589)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
V	Notification (see page 4589)	Forwards notification messages to all owned components.
	RegisterProvider (see page 4590)	This is RegisterProvider, a member of class TLocalConnection.
	UnRegisterProvider (see page 4591)	This is UnRegisterProvider, a member of class TLocalConnection.

Public Methods

	Name	Description
V	Create (see page 4581)	Creates and initializes a local connection component.
V	Destroy (see page 4582)	Frees an instance of TLocalConnection (see page 4580).
V	GetProviderNames (see page 4589)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 4589)	Returns the IAppServer interface for communicating with local providers.
V	SafeCallException (see page 4590)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AppServer (see page 4591)	Provides access to the IAppServer interface that handles communication with providers.
R	ProviderCount (see page 4591)	Indicates the number of providers in the Providers (see page 4592) property list.
R	Providers (see page 4592)	Lists the local providers with which this local connection component can communicate.

42.1.1.2 TLocalConnection.Create Constructor

Creates and initializes a local connection component.

Pascal

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

C++

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

Description

Call Create to instantiate TLocalConnection (see page 4580) at runtime. TLocalConnection (see page 4580) components placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually the form or data module) that is responsible for freeing the TLocalConnection (see page 4580) instance.

See Also

Owner

42.1.1.3 TLocalConnection.Destroy Destructor

Frees an instance of TLocalConnection (see page 4580).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocalConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TLocalConnection (see page 4580) reference is not nil before calling the destructor.

See Also

System::TObject::Free (see page 3382)

42.1.1.4 TLocalConnection Methods

The methods of the TLocalConnection class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 4583)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 4583)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 4584)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 4585)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 4586)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 4586)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 4587)	Returns information from a specified record of the provider's dataset.
	GetConnected (see page 4588)	Returns the value of the Connected property.
	GetProvider (see page 4588)	This is GetProvider, a member of class TLocalConnection.
	InterfaceSupportsErrorInfo (see page 4589)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
	Notification (see page 4589)	Forwards notification messages to all owned components.
	RegisterProvider (see page 4590)	This is RegisterProvider, a member of class TLocalConnection.
	UnRegisterProvider (see page 4591)	This is UnRegisterProvider, a member of class TLocalConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 4589)	Executes a callback for all provider objects on the application server.
	GetServer (see page 4589)	Returns the IAppServer interface for communicating with local providers.
	SafeCallException (see page 4590)	Handles exceptions in methods declared using the safecall calling convention.

42.1.1.4.1 TLocalConnection.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates

applyupdates

BeforeApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

AS_ApplyUpdates

as_applyupdates (see page 1151)

Communicating with the Client Dataset

42.1.1.4.2 TLocalConnection.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest

DataRequest

Communicating with the Client Dataset

42.1.1.4.3 TLocalConnection.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet

beforeexecute

AfterExecute

Params

Execute

Execute

BeforeExecute

AfterExecute

Options

PackageParams

UnpackParams

Communicating with the Client Dataset

42.1.1.4.4 TLocalConnection.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

aftergetparams

AS_Execute

UnpackParams

Communicating with the Client Dataset

42.1.1.4.5 TLocalConnection.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that uses the IAppServer interface of an application server. When calling any other IAppServer method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

providers

ProviderName

Providers (see page 1168)

Communicating with the Client Dataset

42.1.1.4.6 TLocalConnection.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data

Params

GetNextPacket

GetRecords

BeforeGetRecords

BeforeGetRecords

AfterGetRecords

Options

PackageParams

UnpackParams

Communicating with the Client Dataset

42.1.1.4.7 TLocalConnection.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchDetails

RefreshRecord

Communicating with the Client Dataset

42.1.1.4.8 TLocalConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

42.1.1.4.9 TLocalConnection.GetProvider Method

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const UnicodeString ProviderName);
```

Description

This is GetProvider, a member of class TLocalConnection.

42.1.1.4.10 TLocalConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames

AppServer

ProviderName

42.1.1.4.11 TLocalConnection.GetServer Method

Returns the IAppServer interface for communicating with local providers.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

GetServer returns the value of the AppServer (see page 4591) property.

See Also

AppServer (see page 4591)

42.1.1.4.12 TLocalConnection.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TIID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.

42.1.1.4.13 TLocalConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

42.1.1.4.14 TLocalConnection.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Prov);
```

Description

This is RegisterProvider, a member of class TLocalConnection.

42.1.1.4.15 TLocalConnection.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur. TComponent calls the implementation of this method for the interface supported by the component, if it exists. If the component does not support interfaces, this method calls the SafeCallException method inherited from TObject, which returns E_UNEXPECTED. This is a default return value that is appropriate for classes that do not support any interfaces.

See Also

System::TObject::SafeCallException (see page 3391)

42.1.1.4.16 TLocalConnection.UnRegisterProvider Method

Pascal

```
procedure UnRegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Prov);
```

Description

This is UnRegisterProvider, a member of class TLocalConnection.

42.1.1.5 TLocalConnection Properties

The properties of the TLocalConnection class are listed here.

Public Properties

	Name	Description
R	AppServer (see page 4591)	Provides access to the IAppServer interface that handles communication with providers.
R	ProviderCount (see page 4591)	Indicates the number of providers in the Providers (see page 4592) property list.
R	Providers (see page 4592)	Lists the local providers with which this local connection component can communicate.

42.1.1.5.1 TLocalConnection.AppServer Property

Provides access to the IAppServer interface that handles communication with providers.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to make explicit IAppServer calls to communicate with the local providers. The IAppServer interface lets you call provider methods that are otherwise unavailable because they are protected.

See Also

Providers (see page 4592)

AppServer

42.1.1.5.2 TLocalConnection.ProviderCount Property

Indicates the number of providers in the Providers (see page 4592) property list.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

Read (see page 3727) ProviderCount to determine the number of local providers. This is the number of entries in the Providers (see page 4592) property list.

See Also

Providers (see page 4592)

42.1.1.5.3 TLocalConnection.Providers Property

Lists the local providers with which this local connection component can communicate.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[UnicodeString const ProviderName];
```

Description

Providers provides access to the local providers that are available through the IAppServer interface. Providers only lists the providers whose Exported property is set to true.

ProviderName is the name of a provider. If you specify a name of a provider that is not listed, TLocalConnection (see page 4580) raises an exception.

See Also

ProviderCount (see page 4591)

AppServer (see page 4591)

Exported

43 ThemedActnCtrls Namespace

This is namespace ThemedActnCtrls.

Topics

Name	Description
Classes (see page 4594)	The following table lists classes in this documentation.

Classes

	Name	Description
	TThemedAddRemoveItem (see page 4594)	This is class ThemedActnCtrls.TThemedAddRemoveItem.
	TThemedButtonControl (see page 4597)	This is class ThemedActnCtrls.TThemedButtonControl.
	TThemedCustomizePopup (see page 4599)	This is class ThemedActnCtrls.TThemedCustomizePopup.
	TThemedDropDownButton (see page 4601)	This is class ThemedActnCtrls.TThemedDropDownButton.
	TThemedExpandBtn (see page 4604)	This is class ThemedActnCtrls.TThemedExpandBtn.
	TThemedMenuButton (see page 4605)	This is class ThemedActnCtrls.TThemedMenuButton.
	TThemedMenuItem (see page 4607)	This is class ThemedActnCtrls.TThemedMenuItem.
	TThemedPopupMenu (see page 4611)	This is class ThemedActnCtrls.TThemedPopupMenu.
	TThemedToolScrollBtn (see page 4613)	This is class ThemedActnCtrls.TThemedToolScrollBtn.

43.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TThemedAddRemoveItem (see page 4594)	This is class ThemedActnCtrls.TThemedAddRemoveItem.
	TThemedButtonControl (see page 4597)	This is class ThemedActnCtrls.TThemedButtonControl.
	TThemedCustomizePopup (see page 4599)	This is class ThemedActnCtrls.TThemedCustomizePopup.
	TThemedDropDownButton (see page 4601)	This is class ThemedActnCtrls.TThemedDropDownButton.
	TThemedExpandBtn (see page 4604)	This is class ThemedActnCtrls.TThemedExpandBtn.
	TThemedMenuButton (see page 4605)	This is class ThemedActnCtrls.TThemedMenuButton.
	TThemedMenuItem (see page 4607)	This is class ThemedActnCtrls.TThemedMenuItem.
	TThemedPopupMenu (see page 4611)	This is class ThemedActnCtrls.TThemedPopupMenu.
	TThemedToolScrollBtn (see page 4613)	This is class ThemedActnCtrls.TThemedToolScrollBtn.

43.1.1 TThemedAddRemoveItem Class

Class Hierarchy

Pascal

```
TThemedAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TThemedAddRemoveItem : public TCustomAddRemoveItem;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedAddRemoveItem.

43.1.1.1 TThemedAddRemoveItem Members

The following tables list the members exposed by TThemedAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 4595)	Draws a background image on the control.
	DrawGlyph (see page 4596)	Draws a symbol on the control.
	DrawSeparator (see page 4596)	This is DrawSeparator, a member of class TThemedAddRemoveItem.
	DrawText (see page 4596)	Draws text on the control.
	DrawUnusedEdges (see page 4597)	This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.
	GetImageSize (see page 4597)	This is GetImageSize, a member of class TThemedAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 4595)	Calculates the size of the add/remove menu item.

43.1.1.2 TThemedAddRemoveItem Methods

The methods of the TThemedAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4595)	Draws a background image on the control.
	DrawGlyph (see page 4596)	Draws a symbol on the control.
	DrawSeparator (see page 4596)	This is DrawSeparator, a member of class TThemedAddRemoveItem.
	DrawText (see page 4596)	Draws text on the control.
	DrawUnusedEdges (see page 4597)	This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.
	GetImageSize (see page 4597)	This is GetImageSize, a member of class TThemedAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 4595)	Calculates the size of the add/remove menu item.

43.1.1.2.1 TThemedAddRemoveItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height

Width

TextBounds

43.1.1.2.2 TThemedAddRemoveItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.1.2.3 TThemedAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

43.1.1.2.4 TThemedAddRemoveItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TThemedAddRemoveItem.

43.1.1.2.5 TThemedAddRemoveItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

43.1.1.2.6 TThemedAddRemoveItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.

43.1.1.2.7 TThemedAddRemoveItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedAddRemoveItem.

43.1.2 TThemedButtonControl Class

Class Hierarchy

Pascal

```
TThemedButtonControl = class(TCustomButtonControl);
```

C++

```
class TThemedButtonControl : public TCustomButtonControl;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedButtonControl.

43.1.2.1 TThemedButtonControl Members

The following tables list the members exposed by TThemedButtonControl.

Protected Methods

	Name	Description
	DrawBackground (see page 4598)	Draws a background image on the control.
	DrawFrame (see page 4598)	This is DrawFrame, a member of class TThemedButtonControl.
	DrawGlyph (see page 4598)	Draws a symbol on the control.

	DrawLargeGlyph (see page 4599)	This is DrawLargeGlyph, a member of class TThemedButtonControl.
	DrawText (see page 4599)	Draws text on the control.

43.1.2.2 TThemedButtonControl Methods

The methods of the TThemedButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4598)	Draws a background image on the control.
	DrawFrame (see page 4598)	This is DrawFrame, a member of class TThemedButtonControl.
	DrawGlyph (see page 4598)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4599)	This is DrawLargeGlyph, a member of class TThemedButtonControl.
	DrawText (see page 4599)	Draws text on the control.

43.1.2.2.1 TThemedButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.2.2.2 TThemedButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedButtonControl.

43.1.2.2.3 TThemedButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

43.1.2.2.4 TThemedButtonControl.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedButtonControl.

43.1.2.2.5 TThemedButtonControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

43.1.3 TThemedCustomizePopup Class

Class Hierarchy

Pascal

```
TThemedCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TThemedCustomizePopup : public TCustomizeActionToolBar;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedCustomizePopup.

43.1.3.1 TThemedCustomizePopup Members

The following tables list the members exposed by TThemedCustomizePopup.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 4600)	This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.
	GetDefaultColorMapClass (see page 4601)	This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.
	NCPaint (see page 4601)	This is NCPaint, a member of class TThemedCustomizePopup.

Public Methods

	Name	Description
	Create (see page 4600)	Creates and initializes a TCustomActionPopupMenu instance.

43.1.3.2 TThemedCustomizePopup.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

43.1.3.3 TThemedCustomizePopup Methods

The methods of the TThemedCustomizePopup class are listed here.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 4600)	This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.
	GetDefaultColorMapClass (see page 4601)	This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.
	NCPaint (see page 4601)	This is NCPaint, a member of class TThemedCustomizePopup.

43.1.3.3.1 TThemedCustomizePopup.GetAddRemoveItemClass Method

Pascal

```
function GetAddRemoveItemClass: TCustomAddRemoveItemClass; override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass();
```

Description

This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.

43.1.3.3.2 TThemedCustomizePopup.GetDefaultColorMapClass Method

Pascal

```
function GetDefaultColorMapClass: TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetDefaultColorMapClass();
```

Description

This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.

43.1.3.3.3 TThemedCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TThemedCustomizePopup.

43.1.4 TThemedDropDownButton Class

Class Hierarchy

Pascal

```
TThemedDropDownButton = class(TCustomDropDownButton);
```

C++

```
class TThemedDropDownButton : public TCustomDropDownButton;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedDropDownButton.

43.1.4.1 TThemedDropDownButton Members

The following tables list the members exposed by TThemedDropDownButton.

Protected Methods

	Name	Description
	DrawBackground (see page 4602)	Draws a background image on the control.
	DrawFrame (see page 4602)	This is DrawFrame, a member of class TThemedDropDownButton.
	DrawGlyph (see page 4603)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4603)	This is DrawLargeGlyph, a member of class TThemedDropDownButton.
	GetDropDownButtonWidth (see page 4603)	This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.
	GetPopupClass (see page 4603)	This is GetPopupClass, a member of class TThemedDropDownButton.

43.1.4.2 TThemedDropDownButton Methods

The methods of the TThemedDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4602)	Draws a background image on the control.
	DrawFrame (see page 4602)	This is DrawFrame, a member of class TThemedDropDownButton.
	DrawGlyph (see page 4603)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4603)	This is DrawLargeGlyph, a member of class TThemedDropDownButton.
	GetDropDownButtonWidth (see page 4603)	This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.
	GetPopupClass (see page 4603)	This is GetPopupClass, a member of class TThemedDropDownButton.

43.1.4.2.1 TThemedDropDownButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.4.2.2 TThemedDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedDropDownButton.

43.1.4.2.3 TThemedDropDownButton.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

43.1.4.2.4 TThemedDropDownButton.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedDropDownButton.

43.1.4.2.5 TThemedDropDownButton.GetDropDownButtonWidth Method

Pascal

```
function GetDropDownButtonWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDropDownButtonWidth();
```

Description

This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.

43.1.4.2.6 TThemedDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TThemedDropDownButton.

43.1.5 TThemedExpandBtn Class

Class Hierarchy

Pascal

```
TThemedExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TThemedExpandBtn : public TCustomMenuExpandBtn;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedExpandBtn.

43.1.5.1 TThemedExpandBtn Members

The following tables list the members exposed by TThemedExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 4604)	Draws a background image on the control.
	DrawFrame (see page 4605)	This is DrawFrame, a member of class TThemedExpandBtn.
	GetImageSize (see page 4605)	This is GetImageSize, a member of class TThemedExpandBtn.

43.1.5.2 TThemedExpandBtn Methods

The methods of the TThemedExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4604)	Draws a background image on the control.
	DrawFrame (see page 4605)	This is DrawFrame, a member of class TThemedExpandBtn.
	GetImageSize (see page 4605)	This is GetImageSize, a member of class TThemedExpandBtn.

43.1.5.2.1 TThemedExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.5.2.2 TThemedExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedExpandBtn.

43.1.5.2.3 TThemedExpandBtn.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedExpandBtn.

43.1.6 TThemedMenuButton Class

Class Hierarchy

Pascal

```
TThemedMenuButton = class(TCustomMenuButton);
```

C++

```
class TThemedMenuButton : public TCustomMenuButton;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedMenuButton.

43.1.6.1 TThemedMenuButton Members

The following tables list the members exposed by TThemedMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 4606)	Draws a background image on the control.
	DrawText (see page 4607)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 4606)	Calculates the size of the control.

43.1.6.2 TThemedMenuButton Methods

The methods of the TThemedMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4606)	Draws a background image on the control.
	DrawText (see page 4607)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 4606)	Calculates the size of the control.

43.1.6.2.1 TThemedMenuButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

43.1.6.2.2 TThemedMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.6.2.3 TThemedMenuButton.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

43.1.7 TThemedMenuItem Class

Class Hierarchy

Pascal

```
TThemedMenuItem = class(TCustomMenuItem);
```

C++

```
class TThemedMenuItem : public TCustomMenuItem;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedMenuItem.

43.1.7.1 TThemedMenuItem Members

The following tables list the members exposed by TThemedMenuItem.

Protected Methods

	Name	Description
	DrawBackground (see page 4608)	Draws a background image on the control.
	DrawGlyph (see page 4609)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4609)	This is DrawLargeGlyph, a member of class TThemedMenuItem.
	DrawSeparator (see page 4609)	This is DrawSeparator, a member of class TThemedMenuItem.

	DrawShadowedText (see page 4610)	This is DrawShadowedText, a member of class TThemedMenuItem.
	DrawSubMenuGlyph (see page 4610)	This is DrawSubMenuGlyph, a member of class TThemedMenuItem.
	DrawText (see page 4610)	Draws text on the control.
	DrawUnusedEdges (see page 4610)	This is DrawUnusedEdges, a member of class TThemedMenuItem.
	GetImageSize (see page 4611)	This is GetImageSize, a member of class TThemedMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 4608)	Calculates the size of the menu item.

43.1.7.2 TThemedMenuItem Methods

The methods of the TThemedMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4608)	Draws a background image on the control.
	DrawGlyph (see page 4609)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4609)	This is DrawLargeGlyph, a member of class TThemedMenuItem.
	DrawSeparator (see page 4609)	This is DrawSeparator, a member of class TThemedMenuItem.
	DrawShadowedText (see page 4610)	This is DrawShadowedText, a member of class TThemedMenuItem.
	DrawSubMenuGlyph (see page 4610)	This is DrawSubMenuGlyph, a member of class TThemedMenuItem.
	DrawText (see page 4610)	Draws text on the control.
	DrawUnusedEdges (see page 4610)	This is DrawUnusedEdges, a member of class TThemedMenuItem.
	GetImageSize (see page 4611)	This is GetImageSize, a member of class TThemedMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 4608)	Calculates the size of the menu item.

43.1.7.2.1 TThemedMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

43.1.7.2.2 TThemedMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

43.1.7.2.3 TThemedMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

43.1.7.2.4 TThemedMenuItem.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedMenuItem.

43.1.7.2.5 TThemedMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TThemedMenuItem.

43.1.7.2.6 TThemedMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TThemedMenuItem.

43.1.7.2.7 TThemedMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; override;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TThemedMenuItem.

43.1.7.2.8 TThemedMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

43.1.7.2.9 TThemedMenuItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TThemedMenuItem.

43.1.7.2.10 TThemedMenuItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedMenuItem.

43.1.8 TThemedPopupMenu Class

Class Hierarchy

Pascal

```
TThemedPopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TThemedPopupMenu : public TCustomActionPopupMenu;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedPopupMenu.

43.1.8.1 TThemedPopupMenu Members

The following tables list the members exposed by TThemedPopupMenu.

Protected Methods

	Name	Description
	GetExpandBtnClass (see page 4612)	This is GetExpandBtnClass, a member of class TThemedPopupMenu.
	NCPaint (see page 4612)	This is NCPaint, a member of class TThemedPopupMenu.
	PositionPopup (see page 4612)	This is PositionPopup, a member of class TThemedPopupMenu.
	VisibleChanging (see page 4613)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
	Create (see page 4611)	Creates and initializes a TCustomActionPopupMenu instance.

43.1.8.2 TThemedPopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

43.1.8.3 TThemedPopupMenu Methods

The methods of the TThemedPopupMenu class are listed here.

Protected Methods

	Name	Description
	GetExpandBtnClass (see page 4612)	This is GetExpandBtnClass, a member of class TThemedPopupMenu.
	NCPaint (see page 4612)	This is NCPaint, a member of class TThemedPopupMenu.
	PositionPopup (see page 4612)	This is PositionPopup, a member of class TThemedPopupMenu.
	VisibleChanging (see page 4613)	Provides the interface for a method that responds to changes in the control's visibility.

43.1.8.3.1 TThemedPopupMenu.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TThemedPopupMenu.

43.1.8.3.2 TThemedPopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TThemedPopupMenu.

43.1.8.3.3 TThemedPopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TThemedPopupMenu.

43.1.8.3.4 TThemedPopupMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible

43.1.9 TThemedToolScrollBtn Class

Class Hierarchy

Pascal

```
TThemedToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TThemedToolScrollBtn : public TCustomToolScrollBtn;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedToolScrollBtn.

43.1.9.1 TThemedToolScrollBtn Members

The following tables list the members exposed by TThemedToolScrollBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 4614)	Draws a background image on the control.

43.1.9.2 TThemedToolScrollBtn Methods

The methods of the TThemedToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4614)	Draws a background image on the control.

43.1.9.2.1 TThemedToolScrollBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

44 Themes Namespace

This is namespace Themes.

Topics

Name	Description
Classes (see page 4617)	The following table lists classes in this documentation.
Functions (see page 4634)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4636)	The following table lists structs, records, enums in this documentation.
Types (see page 4663)	The following table lists types in this documentation.
Variables (see page 4665)	The following table lists variables in this documentation.

Classes

	Name	Description
	TThemeServices (see page 4617)	This is class Themes.TThemeServices.

Functions

	Name	Description
	ThemeControl (see page 4634)	This is function Themes.ThemeControl.
	ThemeServices (see page 4634)	This is function Themes.ThemeServices.
	UnthemedDesigner (see page 4634)	This is function Themes.UnthemedDesigner.

Structs, Records, Enums

	Name	Description
	TThemedButton (see page 4636)	This is record Themes.TThemedButton.
	TThemedClock (see page 4638)	This is record Themes.TThemedClock.
	TThemedComboBox (see page 4638)	This is record Themes.TThemedComboBox.
	TThemedEdit (see page 4639)	This is record Themes.TThemedEdit.
	TThemedElement (see page 4639)	This is record Themes.TThemedElement.
	TThemedElementDetails (see page 4640)	This is record Themes.TThemedElementDetails.
	TThemedExplorerBar (see page 4641)	This is record Themes.TThemedExplorerBar.
	TThemedHeader (see page 4642)	This is record Themes.TThemedHeader.
	TThemedListview (see page 4643)	This is record Themes.TThemedListview.
	TThemedMenu (see page 4643)	This is record Themes.TThemedMenu.
	TThemedPage (see page 4644)	This is record Themes.TThemedPage.
	TThemedProgress (see page 4645)	This is record Themes.TThemedProgress.
	TThemedRebar (see page 4646)	This is record Themes.TThemedRebar.

	TThemedScrollBar (see page 4647)	This is record Themes.TThemedScrollBar.
	TThemedSpin (see page 4649)	This is record Themes.TThemedSpin.
	TThemedStartPanel (see page 4650)	This is record Themes.TThemedStartPanel.
	TThemedStatus (see page 4650)	This is record Themes.TThemedStatus.
	TThemedTab (see page 4651)	This is record Themes.TThemedTab.
	TThemedTaskBand (see page 4653)	This is record Themes.TThemedTaskBand.
	TThemedTaskBar (see page 4653)	This is record Themes.TThemedTaskBar.
	TThemedToolBar (see page 4654)	This is record Themes.TThemedToolBar.
	TThemedToolTip (see page 4655)	This is record Themes.TThemedToolTip.
	TThemedTrackBar (see page 4656)	This is record Themes.TThemedTrackBar.
	TThemedTrayNotify (see page 4657)	This is record Themes.TThemedTrayNotify.
	TThemedTreeview (see page 4658)	This is record Themes.TThemedTreeview.
	TThemedWindow (see page 4658)	This is record Themes.TThemedWindow.

Types

Name	Description
PThemedElementDetails (see page 4663)	This is type Themes.PThemedElementDetails.
TThemeData (see page 4663)	This is type Themes.TThemeData.
TThemeServicesClass (see page 4663)	This is type Themes.TThemeServicesClass.

Variables

Name	Description
ThemeServicesClass (see page 4665)	This is variable Themes.ThemeServicesClass.

44.1 Classes

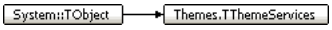
The following table lists classes in this documentation.

Classes

	Name	Description
	TThemeServices (see page 4617)	This is class Themes.TThemeServices.

44.1.1 TThemeServices Class

Class Hierarchy



Pascal

```
TThemeServices = class(TObject);
```

C++

```
class TThemeServices : public TObject;
```

File

Themes

Description

This is class Themes.TThemeServices.

44.1.1.1 TThemeServices Members

The following tables list the members exposed by TThemeServices.

Protected Methods

	Name	Description
	DoOnThemeChange (see page 4621)	This is DoOnThemeChange, a member of class TThemeServices.
	UnloadThemeData (see page 4631)	This is UnloadThemeData, a member of class TThemeServices.

Public Events

	Name	Description
	OnThemeChange (see page 4633)	This is OnThemeChange, a member of class TThemeServices.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	ApplyThemeChange (see page 4620)	This is ApplyThemeChange, a member of class TThemeServices.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ColorToRGB (see page 4620)	This is the overview for the ColorToRGB method overload.
	ContentRect (see page 4621)	This is ContentRect, a member of class TThemeServices.
V	Create (see page 4618)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 4619)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DrawEdge (see page 4621)	This is the overview for the DrawEdge method overload.
	DrawElement (see page 4622)	This is the overview for the DrawElement method overload.
	DrawIcon (see page 4623)	This is DrawIcon, a member of class TThemeServices.
	DrawParentBackground (see page 4623)	This is the overview for the DrawParentBackground method overload.
	DrawText (see page 4625)	This is DrawText, a member of class TThemeServices.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetElementDetails (see page 4625)	This is the overview for the GetElementDetails method overload.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HasTransparentParts (see page 4631)	This is HasTransparentParts, a member of class TThemeServices.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintBorder (see page 4631)	This is PaintBorder, a member of class TThemeServices.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.
	UpdateThemes (see page 4632)	This is UpdateThemes, a member of class TThemeServices.

Public Properties

	Name	Description
R	Theme (see page 4632)	This is Theme, a member of class TThemeServices.
R	ThemesAvailable (see page 4632)	This is ThemesAvailable, a member of class TThemeServices.
R	ThemesEnabled (see page 4632)	This is ThemesEnabled, a member of class TThemeServices.

44.1.1.2 TThemeServices.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TThemeServices();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

44.1.1.3 TThemeServices.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThemeServices();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

44.1.1.4 TThemeServices Methods

The methods of the TThemeServices class are listed here.

Protected Methods

	Name	Description
	DoOnThemeChange (see page 4621)	This is DoOnThemeChange, a member of class TThemeServices.
	UnloadThemeData (see page 4631)	This is UnloadThemeData, a member of class TThemeServices.

Public Methods

	Name	Description
	ApplyThemeChange (see page 4620)	This is ApplyThemeChange, a member of class TThemeServices.
	ColorToRGB (see page 4620)	This is the overview for the ColorToRGB method overload.
	ContentRect (see page 4621)	This is ContentRect, a member of class TThemeServices.
	DrawEdge (see page 4621)	This is the overview for the DrawEdge method overload.
	DrawElement (see page 4622)	This is the overview for the DrawElement method overload.
	DrawIcon (see page 4623)	This is DrawIcon, a member of class TThemeServices.
	DrawParentBackground (see page 4623)	This is the overview for the DrawParentBackground method overload.
	DrawText (see page 4625)	This is DrawText, a member of class TThemeServices.
	GetElementDetails (see page 4625)	This is the overview for the GetElementDetails method overload.
	HasTransparentParts (see page 4631)	This is HasTransparentParts, a member of class TThemeServices.
	PaintBorder (see page 4631)	This is PaintBorder, a member of class TThemeServices.
	UpdateThemes (see page 4632)	This is UpdateThemes, a member of class TThemeServices.

44.1.1.4.1 TThemeServices.ApplyThemeChange Method

Pascal

```
procedure ApplyThemeChange;
```

C++

```
__fastcall ApplyThemeChange();
```

Description

This is ApplyThemeChange, a member of class TThemeServices.

44.1.1.4.2 ColorToRGB Method

This is the overview for the ColorToRGB method overload.

Overload List

	Name	Description
	TThemeServices.ColorToRGB (TColor) (see page 4620)	This is ColorToRGB, a member of class TThemeServices.
	TThemeServices.ColorToRGB (TColor, PThemedElementDetails) (see page 4621)	This is ColorToRGB, a member of class TThemeServices.

44.1.1.4.2.1 TThemeServices.ColorToRGB Method (TColor)

Pascal

```
function ColorToRGB(Color: TColor): COLORREF; overload;
```

C++

```
__fastcall COLORREF ColorToRGB(TColor Color);
```

Description

This is ColorToRGB, a member of class TThemeServices.

44.1.1.4.2.2 TThemeServices.ColorToRGB Method (TColor, PThemedElementDetails)

Pascal

```
function ColorToRGB(Color: TColor; Details: PThemedElementDetails): COLORREF; overload;
```

C++

```
__fastcall COLORREF ColorToRGB(TColor Color, PThemedElementDetails Details);
```

Description

This is ColorToRGB, a member of class TThemeServices.

44.1.1.4.3 TThemeServices.ContentRect Method

Pascal

```
function ContentRect(DC: HDC; Details: TThemedElementDetails; BoundingRect: TRect): TRect;
```

C++

```
__fastcall TRect ContentRect(HDC DC, TThemedElementDetails Details, TRect BoundingRect);
```

Description

This is ContentRect, a member of class TThemeServices.

44.1.1.4.4 TThemeServices.DoOnThemeChange Method

Pascal

```
procedure DoOnThemeChange; virtual;
```

C++

```
virtual __fastcall DoOnThemeChange();
```

Description

This is DoOnThemeChange, a member of class TThemeServices.

44.1.1.4.5 DrawEdge Method

This is the overview for the DrawEdge method overload.

Overload List

	Name	Description
	TThemeServices.DrawEdge (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal) (see page 4622)	This is DrawEdge, a member of class TThemeServices.
	TThemeServices.DrawEdge (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal, PRect) (see page 4622)	This is DrawEdge, a member of class TThemeServices.

44.1.1.4.5.1 TThemeServices.DrawEdge Method (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal)

Pascal

```
procedure DrawEdge(DC: HDC; Details: TThemedElementDetails; const R: TRect; Edge: Cardinal;
Flags: Cardinal); overload;
```

C++

```
__fastcall DrawEdge(HDC DC, TThemedElementDetails Details, const TRect R, unsigned Edge,
unsigned Flags);
```

Description

This is DrawEdge, a member of class TThemeServices.

44.1.1.4.5.2 TThemeServices.DrawEdge Method (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal, PRect)

Pascal

```
procedure DrawEdge(DC: HDC; Details: TThemedElementDetails; const R: TRect; Edge: Cardinal;
Flags: Cardinal; ContentRect: PRect); overload;
```

C++

```
__fastcall DrawEdge(HDC DC, TThemedElementDetails Details, const TRect R, unsigned Edge,
unsigned Flags, PRect ContentRect);
```

Description

This is DrawEdge, a member of class TThemeServices.

44.1.1.4.6 DrawElement Method

This is the overview for the DrawElement method overload.

Overload List

	Name	Description
	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect) (see page 4622)	This is DrawElement, a member of class TThemeServices.
	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect, PRect) (see page 4623)	This is DrawElement, a member of class TThemeServices.
	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect, TRect) (see page 4623)	This is DrawElement, a member of class TThemeServices.

44.1.1.4.6.1 TThemeServices.DrawElement Method (HDC, TThemedElementDetails, TRect)

Pascal

```
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect); overload;
```

C++

```
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R);
```

Description

This is DrawElement, a member of class TThemeServices.

44.1.1.4.6.2 TThemeServices.DrawElement Method (HDC, TThemedElementDetails, TRect, PRect)

Pascal

```
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect; ClipRect:
PRect); overload;
```

C++

```
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R, PRect ClipRect);
```

Description

This is DrawElement, a member of class TThemeServices.

44.1.1.4.6.3 TThemeServices.DrawElement Method (HDC, TThemedElementDetails, TRect, TRect)

Pascal

```
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect; ClipRect:
TRect); overload;
```

C++

```
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R, TRect ClipRect);
```

Description

This is DrawElement, a member of class TThemeServices.

44.1.1.4.7 TThemeServices.DrawIcon Method

Pascal

```
procedure DrawIcon(DC: HDC; Details: TThemedElementDetails; const R: TRect; himl: HIMAGELIST;
Index: Integer);
```

C++

```
__fastcall DrawIcon(HDC DC, TThemedElementDetails Details, const TRect R, HIMAGELIST himl, int
Index);
```

Description

This is DrawIcon, a member of class TThemeServices.

44.1.1.4.8 DrawParentBackground Method

This is the overview for the DrawParentBackground method overload.

Overload List

	Name	Description
	TThemeServices.DrawParentBackground (HWND, HDC, PThemedElementDetails, Boolean, PRect) (see page 4624)	This is DrawParentBackground, a member of class TThemeServices.

	TThemeServices.DrawParentBackground (HWND, HDC, PThemedElementDetails, Boolean, TRect) (see page 4624)	This is DrawParentBackground, a member of class TThemeServices.
	TThemeServices.DrawParentBackground (HWND, HDC, TThemedElementDetails, Boolean, PRect) (see page 4624)	This is DrawParentBackground, a member of class TThemeServices.

44.1.1.4.8.1 TThemeServices.DrawParentBackground Method (HWND, HDC, PThemedElementDetails, Boolean, PRect)

Pascal

```
procedure DrawParentBackground(Window: HWND; Target: HDC; Details: PThemedElementDetails;
OnlyIfTransparent: Boolean; Bounds: PRect = nil); overload;
```

C++

```
__fastcall DrawParentBackground(HWND Window, HDC Target, PThemedElementDetails Details,
Boolean OnlyIfTransparent, PRect Bounds = nil);
```

Description

This is DrawParentBackground, a member of class TThemeServices.

44.1.1.4.8.2 TThemeServices.DrawParentBackground Method (HWND, HDC, PThemedElementDetails, Boolean, TRect)

Pascal

```
procedure DrawParentBackground(Window: HWND; Target: HDC; Details: PThemedElementDetails;
OnlyIfTransparent: Boolean; const Bounds: TRect); overload;
```

C++

```
__fastcall DrawParentBackground(HWND Window, HDC Target, PThemedElementDetails Details,
Boolean OnlyIfTransparent, const TRect Bounds);
```

Description

This is DrawParentBackground, a member of class TThemeServices.

44.1.1.4.8.3 TThemeServices.DrawParentBackground Method (HWND, HDC, TThemedElementDetails, Boolean, PRect)

Pascal

```
procedure DrawParentBackground(Window: HWND; Target: HDC; const Details:
TThemedElementDetails; OnlyIfTransparent: Boolean; Bounds: PRect = nil); overload;
```

C++

```
__fastcall DrawParentBackground(HWND Window, HDC Target, const TThemedElementDetails Details,
Boolean OnlyIfTransparent, PRect Bounds = nil);
```

Description

This is DrawParentBackground, a member of class TThemeServices.

44.1.1.4.9 TThemeServices.DrawText Method

Pascal

```
procedure DrawText(DC: HDC; Details: TThemedElementDetails; const S: WideString; R: TRect;
Flags: Cardinal; Flags2: Cardinal);
```

C++

```
__fastcall DrawText(HDC DC, TThemedElementDetails Details, const BSTR S, TRect R, unsigned
Flags, unsigned Flags2);
```

Description

This is DrawText, a member of class TThemeServices.

44.1.1.4.10 GetElementDetails Method

This is the overview for the GetElementDetails method overload.

Overload List

	Name	Description
	TThemeServices.GetElementDetails (TThemedButton) (see page 4626)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedClock) (see page 4626)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedComboBox) (see page 4626)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedEdit) (see page 4627)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedExplorerBar) (see page 4627)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedHeader) (see page 4627)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedListView) (see page 4627)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedMenu) (see page 4627)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedPage) (see page 4628)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedProgress) (see page 4628)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedRebar) (see page 4628)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedScrollBar) (see page 4628)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedSpin) (see page 4629)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedStartPanel) (see page 4629)	This is GetElementDetails, a member of class TThemeServices.

	TThemeServices.GetElementDetails (TThemedStatus) (see page 4629)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTab) (see page 4629)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTaskBand) (see page 4629)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTaskBar) (see page 4630)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedToolBar) (see page 4630)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedToolTip) (see page 4630)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTrackBar) (see page 4630)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTrayNotify) (see page 4630)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedTreeview) (see page 4631)	This is GetElementDetails, a member of class TThemeServices.
	TThemeServices.GetElementDetails (TThemedWindow) (see page 4631)	This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.1 TThemeServices.GetElementDetails Method (TThemedButton)

Pascal

```
function GetElementDetails(Detail: TThemedButton): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedButton Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.2 TThemeServices.GetElementDetails Method (TThemedClock)

Pascal

```
function GetElementDetails(Detail: TThemedClock): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedClock Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.3 TThemeServices.GetElementDetails Method (TThemedComboBox)

Pascal

```
function GetElementDetails(Detail: TThemedComboBox): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedComboBox Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.4 TThemeServices.GetElementDetails Method (TThemedEdit)

Pascal

```
function GetElementDetails(Detail: TThemedEdit): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedEdit Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.5 TThemeServices.GetElementDetails Method (TThemedExplorerBar)

Pascal

```
function GetElementDetails(Detail: TThemedExplorerBar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedExplorerBar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.6 TThemeServices.GetElementDetails Method (TThemedHeader)

Pascal

```
function GetElementDetails(Detail: TThemedHeader): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedHeader Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.7 TThemeServices.GetElementDetails Method (TThemedListView)

Pascal

```
function GetElementDetails(Detail: TThemedListView): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedListView Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.8 TThemeServices.GetElementDetails Method (TThemedMenu)

Pascal

```
function GetElementDetails(Detail: TThemedMenu): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedMenu Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.9 TThemeServices.GetElementDetails Method (TThemedPage)

Pascal

```
function GetElementDetails(Detail: TThemedPage): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedPage Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.10 TThemeServices.GetElementDetails Method (TThemedProgress)

Pascal

```
function GetElementDetails(Detail: TThemedProgress): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedProgress Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.11 TThemeServices.GetElementDetails Method (TThemedRebar)

Pascal

```
function GetElementDetails(Detail: TThemedRebar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedRebar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.12 TThemeServices.GetElementDetails Method (TThemedScrollBar)

Pascal

```
function GetElementDetails(Detail: TThemedScrollBar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedScrollBar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.13 TThemeServices.GetElementDetails Method (TThemedSpin)

Pascal

```
function GetElementDetails(Detail: TThemedSpin): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedSpin Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.14 TThemeServices.GetElementDetails Method (TThemedStartPanel)

Pascal

```
function GetElementDetails(Detail: TThemedStartPanel): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedStartPanel Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.15 TThemeServices.GetElementDetails Method (TThemedStatus)

Pascal

```
function GetElementDetails(Detail: TThemedStatus): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedStatus Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.16 TThemeServices.GetElementDetails Method (TThemedTab)

Pascal

```
function GetElementDetails(Detail: TThemedTab): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTab Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.17 TThemeServices.GetElementDetails Method (TThemedTaskBand)

Pascal

```
function GetElementDetails(Detail: TThemedTaskBand): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTaskBand Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.18 TThemeServices.GetElementDetails Method (TThemedTaskBar)

Pascal

```
function GetElementDetails(Detail: TThemedTaskBar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTaskBar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.19 TThemeServices.GetElementDetails Method (TThemedToolBar)

Pascal

```
function GetElementDetails(Detail: TThemedToolBar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedToolBar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.20 TThemeServices.GetElementDetails Method (TThemedToolTip)

Pascal

```
function GetElementDetails(Detail: TThemedToolTip): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedToolTip Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.21 TThemeServices.GetElementDetails Method (TThemedTrackBar)

Pascal

```
function GetElementDetails(Detail: TThemedTrackBar): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTrackBar Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.22 TThemeServices.GetElementDetails Method (TThemedTrayNotify)

Pascal

```
function GetElementDetails(Detail: TThemedTrayNotify): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTrayNotify Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.23 TThemeServices.GetElementDetails Method (TThemedTreeview)

Pascal

```
function GetElementDetails(Detail: TThemedTreeview): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedTreeview Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.10.24 TThemeServices.GetElementDetails Method (TThemedWindow)

Pascal

```
function GetElementDetails(Detail: TThemedWindow): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedWindow Detail);
```

Description

This is GetElementDetails, a member of class TThemeServices.

44.1.1.4.11 TThemeServices.HasTransparentParts Method

Pascal

```
function HasTransparentParts(Details: TThemedElementDetails): Boolean;
```

C++

```
__fastcall Boolean HasTransparentParts(TThemedElementDetails Details);
```

Description

This is HasTransparentParts, a member of class TThemeServices.

44.1.1.4.12 TThemeServices.PaintBorder Method

Pascal

```
procedure PaintBorder(Control: TWinControl; EraseLRCorner: Boolean);
```

C++

```
__fastcall PaintBorder(TWinControl * Control, Boolean EraseLRCorner);
```

Description

This is PaintBorder, a member of class TThemeServices.

44.1.1.4.13 TThemeServices.UnloadThemeData Method

Pascal

```
procedure UnloadThemeData;
```

C++

```
__fastcall UnloadThemeData();
```

Description

This is UnloadThemeData, a member of class TThemeServices.

44.1.1.4.14 TThemeServices.UpdateThemes Method

Pascal

```
procedure UpdateThemes;
```

C++

```
__fastcall UpdateThemes();
```

Description

This is UpdateThemes, a member of class TThemeServices.

44.1.1.5 TThemeServices Properties

The properties of the TThemeServices class are listed here.

Public Properties

	Name	Description
	Theme (see page 4632)	This is Theme, a member of class TThemeServices.
	ThemesAvailable (see page 4632)	This is ThemesAvailable, a member of class TThemeServices.
	ThemesEnabled (see page 4632)	This is ThemesEnabled, a member of class TThemeServices.

44.1.1.5.1 TThemeServices.Theme Property

Pascal

```
property Theme [Element: TThemedElement]: HTHEME;
```

C++

```
__property HTHEME Theme[TThemedElement Element];
```

Description

This is Theme, a member of class TThemeServices.

44.1.1.5.2 TThemeServices.ThemesAvailable Property

Pascal

```
property ThemesAvailable: Boolean;
```

C++

```
__property Boolean ThemesAvailable;
```

Description

This is ThemesAvailable, a member of class TThemeServices.

44.1.1.5.3 TThemeServices.ThemesEnabled Property

Pascal

```
property ThemesEnabled: Boolean;
```

C++

```
__property Boolean ThemesEnabled;
```

Description

This is ThemesEnabled, a member of class TThemeServices.

44.1.1.6 TThemeServices Events

The events of the TThemeServices class are listed here.

Public Events

	Name	Description
	OnThemeChange (see page 4633)	This is OnThemeChange, a member of class TThemeServices.

44.1.1.6.1 TThemeServices.OnThemeChange Event

Pascal

```
property OnThemeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnThemeChange;
```

Description

This is OnThemeChange, a member of class TThemeServices.

44.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ThemeControl (see page 4634)	This is function Themes.ThemeControl.
	ThemeServices (see page 4634)	This is function Themes.ThemeServices.
	UnthemedDesigner (see page 4634)	This is function Themes.UnthemedDesigner.

44.2.1 Themes.ThemeControl Function

Pascal

```
function ThemeControl(AControl: TControl): Boolean;
```

C++

```
Boolean ThemeControl(TControl * AControl);
```

File

Themes

Description

This is function Themes.ThemeControl.

44.2.2 Themes.ThemeServices Function

Pascal

```
function ThemeServices: TThemeServices;
```

C++

```
TThemeServices ThemeServices();
```

File

Themes

Description

This is function Themes.ThemeServices.

44.2.3 Themes.UnthemedDesigner Function

Pascal

```
function UnthemedDesigner(AControl: TControl): Boolean;
```

C++

```
Boolean UnthemedDesigner(TControl * AControl);
```

File

Themes

Description

This is function Themes.UnthemedDesigner.

44.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TThemedButton (see page 4636)	This is record Themes.TThemedButton.
	TThemedClock (see page 4638)	This is record Themes.TThemedClock.
	TThemedComboBox (see page 4638)	This is record Themes.TThemedComboBox.
	TThemedEdit (see page 4639)	This is record Themes.TThemedEdit.
	TThemedElement (see page 4639)	This is record Themes.TThemedElement.
	TThemedExplorerBar (see page 4641)	This is record Themes.TThemedExplorerBar.
	TThemedHeader (see page 4642)	This is record Themes.TThemedHeader.
	TThemedListview (see page 4643)	This is record Themes.TThemedListview.
	TThemedMenu (see page 4643)	This is record Themes.TThemedMenu.
	TThemedPage (see page 4644)	This is record Themes.TThemedPage.
	TThemedProgress (see page 4645)	This is record Themes.TThemedProgress.
	TThemedRebar (see page 4646)	This is record Themes.TThemedRebar.
	TThemedScrollBar (see page 4647)	This is record Themes.TThemedScrollBar.
	TThemedSpin (see page 4649)	This is record Themes.TThemedSpin.
	TThemedStartPanel (see page 4650)	This is record Themes.TThemedStartPanel.
	TThemedStatus (see page 4650)	This is record Themes.TThemedStatus.
	TThemedTab (see page 4651)	This is record Themes.TThemedTab.
	TThemedTaskBand (see page 4653)	This is record Themes.TThemedTaskBand.
	TThemedTaskBar (see page 4653)	This is record Themes.TThemedTaskBar.
	TThemedToolBar (see page 4654)	This is record Themes.TThemedToolBar.
	TThemedToolTip (see page 4655)	This is record Themes.TThemedToolTip.
	TThemedTrackBar (see page 4656)	This is record Themes.TThemedTrackBar.
	TThemedTrayNotify (see page 4657)	This is record Themes.TThemedTrayNotify.
	TThemedTreeview (see page 4658)	This is record Themes.TThemedTreeview.
	TThemedWindow (see page 4658)	This is record Themes.TThemedWindow.

Records

	Name	Description
	TThemedElementDetails (see page 4640)	This is record Themes.TThemedElementDetails.

44.3.1 Themes.TThemedButton Enumeration

Pascal

```
TThemedButton = (
  tbButtonDontCare,
  tbButtonRoot,
  tbPushButtonNormal,
  tbPushButtonHot,
  tbPushButtonPressed,
  tbPushButtonDisabled,
  tbPushButtonDefaulted,
  tbRadioButtonUncheckedNormal,
  tbRadioButtonUncheckedHot,
```

```
    tbRadioButtonUncheckedPressed,
    tbRadioButtonUncheckedDisabled,
    tbRadioButtonCheckedNormal,
    tbRadioButtonCheckedHot,
    tbRadioButtonCheckedPressed,
    tbRadioButtonCheckedDisabled,
    tbCheckBoxUncheckedNormal,
    tbCheckBoxUncheckedHot,
    tbCheckBoxUncheckedPressed,
    tbCheckBoxUncheckedDisabled,
    tbCheckBoxCheckedNormal,
    tbCheckBoxCheckedHot,
    tbCheckBoxCheckedPressed,
    tbCheckBoxCheckedDisabled,
    tbCheckBoxMixedNormal,
    tbCheckBoxMixedHot,
    tbCheckBoxMixedPressed,
    tbCheckBoxMixedDisabled,
    tbGroupBoxNormal,
    tbGroupBoxDisabled,
    tbUserButton
  );
```

C++

```
  enum TThemedButton {
    tbButtonDontCare,
    tbButtonRoot,
    tbPushButtonNormal,
    tbPushButtonHot,
    tbPushButtonPressed,
    tbPushButtonDisabled,
    tbPushButtonDefaulted,
    tbRadioButtonUncheckedNormal,
    tbRadioButtonUncheckedHot,
    tbRadioButtonUncheckedPressed,
    tbRadioButtonUncheckedDisabled,
    tbRadioButtonCheckedNormal,
    tbRadioButtonCheckedHot,
    tbRadioButtonCheckedPressed,
    tbRadioButtonCheckedDisabled,
    tbCheckBoxUncheckedNormal,
    tbCheckBoxUncheckedHot,
    tbCheckBoxUncheckedPressed,
    tbCheckBoxUncheckedDisabled,
    tbCheckBoxCheckedNormal,
    tbCheckBoxCheckedHot,
    tbCheckBoxCheckedPressed,
    tbCheckBoxCheckedDisabled,
    tbCheckBoxMixedNormal,
    tbCheckBoxMixedHot,
    tbCheckBoxMixedPressed,
    tbCheckBoxMixedDisabled,
    tbGroupBoxNormal,
    tbGroupBoxDisabled,
    tbUserButton
  };
```

File

Themes

Description

This is record Themes.TThemedButton.

44.3.2 Themes.TThemedClock Enumeration

Pascal

```
  TThemedClock = (
    tcClockDontCare,
    tcClockRoot,
    tcTimeNormal
  );
```

C++

```
  enum TThemedClock {
    tcClockDontCare,
    tcClockRoot,
    tcTimeNormal
  };
```

File

Themes

Description

This is record Themes.TThemedClock.

44.3.3 Themes.TThemedComboBox Enumeration

Pascal

```
  TThemedComboBox = (
    tcComboBoxDontCare,
    tcComboBoxRoot,
    tcDropDownButtonNormal,
    tcDropDownButtonHot,
    tcDropDownButtonPressed,
    tcDropDownButtonDisabled
  );
```

C++

```
  enum TThemedComboBox {
    tcComboBoxDontCare,
    tcComboBoxRoot,
    tcDropDownButtonNormal,
    tcDropDownButtonHot,
    tcDropDownButtonPressed,
    tcDropDownButtonDisabled
  };
```

File

Themes

Description

This is record Themes.TThemedComboBox.

44.3.4 Themes.TThemedEdit Enumeration

Pascal

```
TThemedEdit = (
  teEditDontCare,
  teEditRoot,
  teEditTextNormal,
  teEditTextHot,
  teEditTextSelected,
  teEditTextDisabled,
  teEditTextFocused,
  teEditTextReadOnly,
  teEditTextAssist,
  teEditCaret
);
```

C++

```
enum TThemedEdit {
  teEditDontCare,
  teEditRoot,
  teEditTextNormal,
  teEditTextHot,
  teEditTextSelected,
  teEditTextDisabled,
  teEditTextFocused,
  teEditTextReadOnly,
  teEditTextAssist,
  teEditCaret
};
```

File

Themes

Description

This is record Themes.TThemedEdit.

44.3.5 Themes.TThemedElement Enumeration

Pascal

```
TThemedElement = (
  teButton,
  teClock,
  teComboBox,
  teEdit,
  teExplorerBar,
  teHeader,
  teListView,
  teMenu,
  tePage,
  teProgress,
  teRebar,
  teScrollBar,
  teSpin,
  teStartPanel,
  teStatus,
```

```
    teTab,
    teTaskBand,
    teTaskBar,
    teToolBar,
    teToolTip,
    teTrackBar,
    teTrayNotify,
    teTreeview,
    teWindow
  );
```

C++

```
  enum TThemedElement {
    teButton,
    teClock,
    teComboBox,
    teEdit,
    teExplorerBar,
    teHeader,
    teListView,
    teMenu,
    tePage,
    teProgress,
    teRebar,
    teScrollBar,
    teSpin,
    teStartPanel,
    teStatus,
    teTab,
    teTaskBand,
    teTaskBar,
    teToolBar,
    teToolTip,
    teTrackBar,
    teTrayNotify,
    teTreeview,
    teWindow
  };
```

File

Themes

Description

This is record Themes.TThemedElement.

44.3.6 Themes.TThemedElementDetails Record

Pascal

```
  TThemedElementDetails = record
    Element: TThemedElement;
    Part: Integer;
    State: Integer;
  end;
```

C++

```
  struct TThemedElementDetails {
    TThemedElement Element;
    int Part;
    int State;
  };
```

File

Themes

Description

This is record Themes.TThemedElementDetails.

44.3.7 Themes.TThemedExplorerBar Enumeration

Pascal

```
TThemedExplorerBar = (
  tebExplorerBarDontCare,
  tebExplorerBarRoot,
  tebHeaderBackgroundNormal,
  tebHeaderBackgroundHot,
  tebHeaderBackgroundPressed,
  tebHeaderCloseNormal,
  tebHeaderCloseHot,
  tebHeaderClosePressed,
  tebHeaderPinNormal,
  tebHeaderPinHot,
  tebHeaderPinPressed,
  tebHeaderPinSelectedNormal,
  tebHeaderPinSelectedHot,
  tebHeaderPinSelectedPressed,
  tebIEBarMenuNormal,
  tebIEBarMenuHot,
  tebIEBarMenuPressed,
  tebNormalGroupBackground,
  tebNormalGroupCollapseNormal,
  tebNormalGroupCollapseHot,
  tebNormalGroupCollapsePressed,
  tebNormalGroupExpandNormal,
  tebNormalGroupExpandHot,
  tebNormalGroupExpandPressed,
  tebNormalGroupHead,
  tebSpecialGroupBackground,
  tebSpecialGroupCollapseSpecial,
  tebSpecialGroupCollapseHot,
  tebSpecialGroupCollapsePressed,
  tebSpecialGroupExpandSpecial,
  tebSpecialGroupExpandHot,
  tebSpecialGroupExpandPressed,
  tebSpecialGroupHead
);
```

C++

```
enum TThemedExplorerBar {
  tebExplorerBarDontCare,
  tebExplorerBarRoot,
  tebHeaderBackgroundNormal,
  tebHeaderBackgroundHot,
  tebHeaderBackgroundPressed,
  tebHeaderCloseNormal,
  tebHeaderCloseHot,
  tebHeaderClosePressed,
  tebHeaderPinNormal,
  tebHeaderPinHot,
  tebHeaderPinPressed,
  tebHeaderPinSelectedNormal,
```

```
    tebHeaderPinSelectedHot,
    tebHeaderPinSelectedPressed,
    tebIEBarMenuNormal,
    tebIEBarMenuHot,
    tebIEBarMenuPressed,
    tebNormalGroupBackground,
    tebNormalGroupCollapseNormal,
    tebNormalGroupCollapseHot,
    tebNormalGroupCollapsePressed,
    tebNormalGroupExpandNormal,
    tebNormalGroupExpandHot,
    tebNormalGroupExpandPressed,
    tebNormalGroupHead,
    tebSpecialGroupBackground,
    tebSpecialGroupCollapseSpecial,
    tebSpecialGroupCollapseHot,
    tebSpecialGroupCollapsePressed,
    tebSpecialGroupExpandSpecial,
    tebSpecialGroupExpandHot,
    tebSpecialGroupExpandPressed,
    tebSpecialGroupHead
  };
```

File

Themes

Description

This is record Themes.TThemedExplorerBar.

44.3.8 Themes.TThemedHeader Enumeration

Pascal

```
  TThemedHeader = (
    thHeaderDontCare,
    thHeaderRoot,
    thHeaderItemNormal,
    thHeaderItemHot,
    thHeaderItemPressed,
    thHeaderItemLeftNormal,
    thHeaderItemLeftHot,
    thHeaderItemLeftPressed,
    thHeaderItemRightNormal,
    thHeaderItemRightHot,
    thHeaderItemRightPressed,
    thHeaderSortArrowSortedUp,
    thHeaderSortArrowSortedDown
  );
```

C++

```
  enum TThemedHeader {
    thHeaderDontCare,
    thHeaderRoot,
    thHeaderItemNormal,
    thHeaderItemHot,
    thHeaderItemPressed,
    thHeaderItemLeftNormal,
    thHeaderItemLeftHot,
    thHeaderItemLeftPressed,
    thHeaderItemRightNormal,
    thHeaderItemRightHot,
```

```
    thHeaderItemRightPressed,
    thHeaderSortArrowSortedUp,
    thHeaderSortArrowSortedDown
  };
```

File

Themes

Description

This is record Themes.TThemedHeader.

44.3.9 Themes.TThemedListview Enumeration

Pascal

```
  TThemedListview = (
    tlListviewDontCare,
    tlListviewRoot,
    tlListItemNormal,
    tlListItemHot,
    tlListItemSelected,
    tlListItemDisabled,
    tlListItemSelectedNotFocus,
    tlListGroup,
    tlListDetail,
    tlListSortDetail,
    tlEmptyText
  );
```

C++

```
  enum TThemedListview {
    tlListviewDontCare,
    tlListviewRoot,
    tlListItemNormal,
    tlListItemHot,
    tlListItemSelected,
    tlListItemDisabled,
    tlListItemSelectedNotFocus,
    tlListGroup,
    tlListDetail,
    tlListSortDetail,
    tlEmptyText
  };
```

File

Themes

Description

This is record Themes.TThemedListview.

44.3.10 Themes.TThemedMenu Enumeration

Pascal

```
  TThemedMenu = (
    tmMenuDontCare,
```

```
    tmMenuRoot,
    tmMenuItemNormal,
    tmMenuItemSelected,
    tmMenuItemDemoted,
    tmMenuDropDown,
    tmMenuBarItem,
    tmMenuBarDropDown,
    tmChevron,
    tmSeparator
  );
```

C++

```
  enum TThemedMenu {
    tmMenuDontCare,
    tmMenuRoot,
    tmMenuItemNormal,
    tmMenuItemSelected,
    tmMenuItemDemoted,
    tmMenuDropDown,
    tmMenuBarItem,
    tmMenuBarDropDown,
    tmChevron,
    tmSeparator
  };
```

File

Themes

Description

This is record Themes.TThemedMenu.

44.3.11 Themes.TThemedPage Enumeration

Pascal

```
  TThemedPage = (
    tpPageDontCare,
    tpPageRoot,
    tpUpNormal,
    tpUpHot,
    tpUpPressed,
    tpUpDisabled,
    tpDownNormal,
    tpDownHot,
    tpDownPressed,
    tpDownDisabled,
    tpUpHorzNormal,
    tpUpHorzHot,
    tpUpHorzPressed,
    tpUpHorzDisabled,
    tpDownHorzNormal,
    tpDownHorzHot,
    tpDownHorzPressed,
    tpDownHorzDisabled
  );
```

C++

```
  enum TThemedPage {
    tpPageDontCare,
    tpPageRoot,
    tpUpNormal,
```

```
    tpUpHot,
    tpUpPressed,
    tpUpDisabled,
    tpDownNormal,
    tpDownHot,
    tpDownPressed,
    tpDownDisabled,
    tpUpHorzNormal,
    tpUpHorzHot,
    tpUpHorzPressed,
    tpUpHorzDisabled,
    tpDownHorzNormal,
    tpDownHorzHot,
    tpDownHorzPressed,
    tpDownHorzDisabled
  };
```

File

Themes

Description

This is record Themes.TThemedPage.

44.3.12 Themes.TThemedProgress Enumeration

Pascal

```
  TThemedProgress = (
    tpProgressDontCare,
    tpProgressRoot,
    tpBar,
    tpBarVert,
    tpChunk,
    tpChunkVert
  );
```

C++

```
  enum TThemedProgress {
    tpProgressDontCare,
    tpProgressRoot,
    tpBar,
    tpBarVert,
    tpChunk,
    tpChunkVert
  };
```

File

Themes

Description

This is record Themes.TThemedProgress.

44.3.13 Themes.TThemedRebar Enumeration

Pascal

```
  TThemedRebar = (
    trRebarDontCare,
    trRebarRoot,
    trGripper,
    trGripperVert,
    trBandNormal,
    trBandHot,
    trBandPressed,
    trBandDisabled,
    trBandChecked,
    trBandHotChecked,
    trChevronNormal,
    trChevronHot,
    trChevronPressed,
    trChevronDisabled,
    trChevronVertNormal,
    trChevronVertHot,
    trChevronVertPressed,
    trChevronVertDisabled
  );
```

C++

```
  enum TThemedRebar {
    trRebarDontCare,
    trRebarRoot,
    trGripper,
    trGripperVert,
    trBandNormal,
    trBandHot,
    trBandPressed,
    trBandDisabled,
    trBandChecked,
    trBandHotChecked,
    trChevronNormal,
    trChevronHot,
    trChevronPressed,
    trChevronDisabled,
    trChevronVertNormal,
    trChevronVertHot,
    trChevronVertPressed,
    trChevronVertDisabled
  };
```

File

Themes

Description

This is record Themes.TThemedRebar.

44.3.14 Themes.TThemedScrollBar Enumeration

Pascal

```
  TThemedScrollBar = (
    tsScrollBarDontCare,
    tsScrollBarRoot,
    tsArrowBtnUpNormal,
    tsArrowBtnUpHot,
    tsArrowBtnUpPressed,
    tsArrowBtnUpDisabled,
    tsArrowBtnDownNormal,
    tsArrowBtnDownHot,
    tsArrowBtnDownPressed,
    tsArrowBtnDownDisabled,
    tsArrowBtnLeftNormal,
    tsArrowBtnLeftHot,
    tsArrowBtnLeftPressed,
    tsArrowBtnLeftDisabled,
    tsArrowBtnRightNormal,
    tsArrowBtnRightHot,
    tsArrowBtnRightPressed,
    tsArrowBtnRightDisabled,
    tsThumbBtnHorzNormal,
    tsThumbBtnHorzHot,
    tsThumbBtnHorzPressed,
    tsThumbBtnHorzDisabled,
    tsThumbBtnVertNormal,
    tsThumbBtnVertHot,
    tsThumbBtnVertPressed,
    tsThumbBtnVertDisabled,
    tsLowerTrackHorzNormal,
    tsLowerTrackHorzHot,
    tsLowerTrackHorzPressed,
    tsLowerTrackHorzDisabled,
    tsUpperTrackHorzNormal,
    tsUpperTrackHorzHot,
    tsUpperTrackHorzPressed,
    tsUpperTrackHorzDisabled,
    tsLowerTrackVertNormal,
    tsLowerTrackVertHot,
    tsLowerTrackVertPressed,
    tsLowerTrackVertDisabled,
    tsUpperTrackVertNormal,
    tsUpperTrackVertHot,
    tsUpperTrackVertPressed,
    tsUpperTrackVertDisabled,
    tsGripperHorzNormal,
    tsGripperHorzHot,
    tsGripperHorzPressed,
    tsGripperHorzDisabled,
    tsGripperVertNormal,
    tsGripperVertHot,
    tsGripperVertPressed,
    tsGripperVertDisabled,
    tsSizeBoxRightAlign,
    tsSizeBoxLeftAlign
  );
```

C++

```
  enum TThemedScrollBar {
    tsScrollBarDontCare,
```

```
    tsScrollBarRoot,
    tsArrowBtnUpNormal,
    tsArrowBtnUpHot,
    tsArrowBtnUpPressed,
    tsArrowBtnUpDisabled,
    tsArrowBtnDownNormal,
    tsArrowBtnDownHot,
    tsArrowBtnDownPressed,
    tsArrowBtnDownDisabled,
    tsArrowBtnLeftNormal,
    tsArrowBtnLeftHot,
    tsArrowBtnLeftPressed,
    tsArrowBtnLeftDisabled,
    tsArrowBtnRightNormal,
    tsArrowBtnRightHot,
    tsArrowBtnRightPressed,
    tsArrowBtnRightDisabled,
    tsThumbBtnHorzNormal,
    tsThumbBtnHorzHot,
    tsThumbBtnHorzPressed,
    tsThumbBtnHorzDisabled,
    tsThumbBtnVertNormal,
    tsThumbBtnVertHot,
    tsThumbBtnVertPressed,
    tsThumbBtnVertDisabled,
    tsLowerTrackHorzNormal,
    tsLowerTrackHorzHot,
    tsLowerTrackHorzPressed,
    tsLowerTrackHorzDisabled,
    tsUpperTrackHorzNormal,
    tsUpperTrackHorzHot,
    tsUpperTrackHorzPressed,
    tsUpperTrackHorzDisabled,
    tsLowerTrackVertNormal,
    tsLowerTrackVertHot,
    tsLowerTrackVertPressed,
    tsLowerTrackVertDisabled,
    tsUpperTrackVertNormal,
    tsUpperTrackVertHot,
    tsUpperTrackVertPressed,
    tsUpperTrackVertDisabled,
    tsGripperHorzNormal,
    tsGripperHorzHot,
    tsGripperHorzPressed,
    tsGripperHorzDisabled,
    tsGripperVertNormal,
    tsGripperVertHot,
    tsGripperVertPressed,
    tsGripperVertDisabled,
    tsSizeBoxRightAlign,
    tsSizeBoxLeftAlign
};
```

File

Themes

Description

This is record Themes.TThemedScrollBar.

44.3.15 Themes.TThemedSpin Enumeration

Pascal

```
  TThemedSpin = (
    tsSpinDontCare,
    tsSpinRoot,
    tsUpNormal,
    tsUpHot,
    tsUpPressed,
    tsUpDisabled,
    tsDownNormal,
    tsDownHot,
    tsDownPressed,
    tsDownDisabled,
    tsUpHorzNormal,
    tsUpHorzHot,
    tsUpHorzPressed,
    tsUpHorzDisabled,
    tsDownHorzNormal,
    tsDownHorzHot,
    tsDownHorzPressed,
    tsDownHorzDisabled
  );
```

C++

```
  enum TThemedSpin {
    tsSpinDontCare,
    tsSpinRoot,
    tsUpNormal,
    tsUpHot,
    tsUpPressed,
    tsUpDisabled,
    tsDownNormal,
    tsDownHot,
    tsDownPressed,
    tsDownDisabled,
    tsUpHorzNormal,
    tsUpHorzHot,
    tsUpHorzPressed,
    tsUpHorzDisabled,
    tsDownHorzNormal,
    tsDownHorzHot,
    tsDownHorzPressed,
    tsDownHorzDisabled
  };
```

File

Themes

Description

This is record Themes.TThemedSpin.

44.3.16 Themes.TThemedStartPanel Enumeration

Pascal

```
  TThemedStartPanel = (
    tspStartPanelDontCare,
    tspStartPanelRoot,
    tspUserPane,
    tspMorePrograms,
    tspMoreProgramsArrowNormal,
    tspMoreProgramsArrowHot,
    tspMoreProgramsArrowPressed,
    tspProgList,
    tspProgListSeparator,
    tspPlacesList,
    tspPlacesListSeparator,
    tspLogOff,
    tspLogOffButtonsNormal,
    tspLogOffButtonsHot,
    tspLogOffButtonsPressed,
    tspUserPicture,
    tspPreview
  );
```

C++

```
  enum TThemedStartPanel {
    tspStartPanelDontCare,
    tspStartPanelRoot,
    tspUserPane,
    tspMorePrograms,
    tspMoreProgramsArrowNormal,
    tspMoreProgramsArrowHot,
    tspMoreProgramsArrowPressed,
    tspProgList,
    tspProgListSeparator,
    tspPlacesList,
    tspPlacesListSeparator,
    tspLogOff,
    tspLogOffButtonsNormal,
    tspLogOffButtonsHot,
    tspLogOffButtonsPressed,
    tspUserPicture,
    tspPreview
  };
```

File

Themes

Description

This is record Themes.TThemedStartPanel.

44.3.17 Themes.TThemedStatus Enumeration

Pascal

```
  TThemedStatus = (
    tsStatusDontCare,
```

```
    tsStatusRoot,
    tsPane,
    tsGripperPane,
    tsGripper
  );
```

C++

```
  enum TThemedStatus {
    tsStatusDontCare,
    tsStatusRoot,
    tsPane,
    tsGripperPane,
    tsGripper
  };
```

File

Themes

Description

This is record Themes.TThemedStatus.

44.3.18 Themes.TThemedTab Enumeration

Pascal

```
  TThemedTab = (
    ttTabDontCare,
    ttTabRoot,
    ttTabItemNormal,
    ttTabItemHot,
    ttTabItemSelected,
    ttTabItemDisabled,
    ttTabItemFocused,
    ttTabItemLeftEdgeNormal,
    ttTabItemLeftEdgeHot,
    ttTabItemLeftEdgeSelected,
    ttTabItemLeftEdgeDisabled,
    ttTabItemLeftEdgeFocused,
    ttTabItemRightEdgeNormal,
    ttTabItemRightEdgeHot,
    ttTabItemRightEdgeSelected,
    ttTabItemRightEdgeDisabled,
    ttTabItemRightEdgeFocused,
    ttTabItemBothEdgeNormal,
    ttTabItemBothEdgeHot,
    ttTabItemBothEdgeSelected,
    ttTabItemBothEdgeDisabled,
    ttTabItemBothEdgeFocused,
    ttTopTabItemNormal,
    ttTopTabItemHot,
    ttTopTabItemSelected,
    ttTopTabItemDisabled,
    ttTopTabItemFocused,
    ttTopTabItemLeftEdgeNormal,
    ttTopTabItemLeftEdgeHot,
    ttTopTabItemLeftEdgeSelected,
    ttTopTabItemLeftEdgeDisabled,
    ttTopTabItemLeftEdgeFocused,
    ttTopTabItemRightEdgeNormal,
    ttTopTabItemRightEdgeHot,
    ttTopTabItemRightEdgeSelected,
```

```
    ttTopTabItemRightEdgeDisabled,
    ttTopTabItemRightEdgeFocused,
    ttTopTabItemBothEdgeNormal,
    ttTopTabItemBothEdgeHot,
    ttTopTabItemBothEdgeSelected,
    ttTopTabItemBothEdgeDisabled,
    ttTopTabItemBothEdgeFocused,
    ttPane,
    ttBody
  );
```

C++

```
  enum TThemedTab {
    ttTabDontCare,
    ttTabRoot,
    ttTabItemNormal,
    ttTabItemHot,
    ttTabItemSelected,
    ttTabItemDisabled,
    ttTabItemFocused,
    ttTabItemLeftEdgeNormal,
    ttTabItemLeftEdgeHot,
    ttTabItemLeftEdgeSelected,
    ttTabItemLeftEdgeDisabled,
    ttTabItemLeftEdgeFocused,
    ttTabItemRightEdgeNormal,
    ttTabItemRightEdgeHot,
    ttTabItemRightEdgeSelected,
    ttTabItemRightEdgeDisabled,
    ttTabItemRightEdgeFocused,
    ttTabItemBothEdgeNormal,
    ttTabItemBothEdgeHot,
    ttTabItemBothEdgeSelected,
    ttTabItemBothEdgeDisabled,
    ttTabItemBothEdgeFocused,
    ttTopTabItemNormal,
    ttTopTabItemHot,
    ttTopTabItemSelected,
    ttTopTabItemDisabled,
    ttTopTabItemFocused,
    ttTopTabItemLeftEdgeNormal,
    ttTopTabItemLeftEdgeHot,
    ttTopTabItemLeftEdgeSelected,
    ttTopTabItemLeftEdgeDisabled,
    ttTopTabItemLeftEdgeFocused,
    ttTopTabItemRightEdgeNormal,
    ttTopTabItemRightEdgeHot,
    ttTopTabItemRightEdgeSelected,
    ttTopTabItemRightEdgeDisabled,
    ttTopTabItemRightEdgeFocused,
    ttTopTabItemBothEdgeNormal,
    ttTopTabItemBothEdgeHot,
    ttTopTabItemBothEdgeSelected,
    ttTopTabItemBothEdgeDisabled,
    ttTopTabItemBothEdgeFocused,
    ttPane,
    ttBody
  };
```

File

Themes

Description

This is record Themes.TThemedTab.

44.3.19 Themes.TThemedTaskBand Enumeration

Pascal

```
TThemedTaskBand = (
  ttbTaskBandDontCare,
  ttbTaskBandRoot,
  ttbGroupCount,
  ttbFlashButton,
  ttpFlashButtonGroupMenu
);
```

C++

```
enum TThemedTaskBand {
  ttbTaskBandDontCare,
  ttbTaskBandRoot,
  ttbGroupCount,
  ttbFlashButton,
  ttpFlashButtonGroupMenu
};
```

File

Themes

Description

This is record Themes.TThemedTaskBand.

44.3.20 Themes.TThemedTaskBar Enumeration

Pascal

```
TThemedTaskBar = (
  ttTaskBarDontCare,
  ttTaskBarRoot,
  ttbTimeNormal
);
```

C++

```
enum TThemedTaskBar {
  ttTaskBarDontCare,
  ttTaskBarRoot,
  ttbTimeNormal
};
```

File

Themes

Description

This is record Themes.TThemedTaskBar.

44.3.21 Themes.TThemedToolBar Enumeration

Pascal

```
  TThemedToolBar = (
    ttbToolBarDontCare,
    ttbToolBarRoot,
    ttbButtonNormal,
    ttbButtonHot,
    ttbButtonPressed,
    ttbButtonDisabled,
    ttbButtonChecked,
    ttbButtonCheckedHot,
    ttbDropDownButtonNormal,
    ttbDropDownButtonHot,
    ttbDropDownButtonPressed,
    ttbDropDownButtonDisabled,
    ttbDropDownButtonChecked,
    ttbDropDownButtonCheckedHot,
    ttbSplitButtonNormal,
    ttbSplitButtonHot,
    ttbSplitButtonPressed,
    ttbSplitButtonDisabled,
    ttbSplitButtonChecked,
    ttbSplitButtonCheckedHot,
    ttbSplitButtonDropDownNormal,
    ttbSplitButtonDropDownHot,
    ttbSplitButtonDropDownPressed,
    ttbSplitButtonDropDownDisabled,
    ttbSplitButtonDropDownChecked,
    ttbSplitButtonDropDownCheckedHot,
    ttbSeparatorNormal,
    ttbSeparatorHot,
    ttbSeparatorPressed,
    ttbSeparatorDisabled,
    ttbSeparatorChecked,
    ttbSeparatorCheckedHot,
    ttbSeparatorVertNormal,
    ttbSeparatorVertHot,
    ttbSeparatorVertPressed,
    ttbSeparatorVertDisabled,
    ttbSeparatorVertChecked,
    ttbSeparatorVertCheckedHot
  );
```

C++

```
  enum TThemedToolBar {
    ttbToolBarDontCare,
    ttbToolBarRoot,
    ttbButtonNormal,
    ttbButtonHot,
    ttbButtonPressed,
    ttbButtonDisabled,
    ttbButtonChecked,
    ttbButtonCheckedHot,
    ttbDropDownButtonNormal,
    ttbDropDownButtonHot,
    ttbDropDownButtonPressed,
    ttbDropDownButtonDisabled,
    ttbDropDownButtonChecked,
    ttbDropDownButtonCheckedHot,
    ttbSplitButtonNormal,
```

```
    ttbSplitButtonHot,
    ttbSplitButtonPressed,
    ttbSplitButtonDisabled,
    ttbSplitButtonChecked,
    ttbSplitButtonCheckedHot,
    ttbSplitButtonDropDownNormal,
    ttbSplitButtonDropDownHot,
    ttbSplitButtonDropDownPressed,
    ttbSplitButtonDropDownDisabled,
    ttbSplitButtonDropDownChecked,
    ttbSplitButtonDropDownCheckedHot,
    ttbSeparatorNormal,
    ttbSeparatorHot,
    ttbSeparatorPressed,
    ttbSeparatorDisabled,
    ttbSeparatorChecked,
    ttbSeparatorCheckedHot,
    ttbSeparatorVertNormal,
    ttbSeparatorVertHot,
    ttbSeparatorVertPressed,
    ttbSeparatorVertDisabled,
    ttbSeparatorVertChecked,
    ttbSeparatorVertCheckedHot
  };
```

File

Themes

Description

This is record Themes.TThemedToolBar.

44.3.22 Themes.TThemedToolTip Enumeration

Pascal

```
  TThemedToolTip = (
    tttToolTipDontCare,
    tttToolTipRoot,
    tttStandardNormal,
    tttStandardLink,
    tttStandardTitleNormal,
    tttStandardTitleLink,
    tttBaloonNormal,
    tttBaloonLink,
    tttBaloonTitleNormal,
    tttBaloonTitleLink,
    tttBaloonStemUpLeft,
    tttBaloonStemDownLeft,
    tttCloseNormal,
    tttCloseHot,
    tttClosePressed
  );
```

C++

```
  enum TThemedToolTip {
    tttToolTipDontCare,
    tttToolTipRoot,
    tttStandardNormal,
    tttStandardLink,
    tttStandardTitleNormal,
    tttStandardTitleLink,
```

```
    tttBaloonNormal,
    tttBaloonLink,
    tttBaloonTitleNormal,
    tttBaloonTitleLink,
    tttBaloonStemUpLeft,
    tttBaloonStemDownLeft,
    tttCloseNormal,
    tttCloseHot,
    tttClosePressed
  };
```

File

Themes

Description

This is record Themes.TThemedToolTip.

44.3.23 Themes.TThemedTrackBar Enumeration

Pascal

```
  TThemedTrackBar = (
    ttbTrackBarDontCare,
    ttbTrackBarRoot,
    ttbTrack,
    ttbTrackVert,
    ttbThumbNormal,
    ttbThumbHot,
    ttbThumbPressed,
    ttbThumbFocused,
    ttbThumbDisabled,
    ttbThumbBottomNormal,
    ttbThumbBottomHot,
    ttbThumbBottomPressed,
    ttbThumbBottomFocused,
    ttbThumbBottomDisabled,
    ttbThumbTopNormal,
    ttbThumbTopHot,
    ttbThumbTopPressed,
    ttbThumbTopFocused,
    ttbThumbTopDisabled,
    ttbThumbVertNormal,
    ttbThumbVertHot,
    ttbThumbVertPressed,
    ttbThumbVertFocused,
    ttbThumbVertDisabled,
    ttbThumbLeftNormal,
    ttbThumbLeftHot,
    ttbThumbLeftPressed,
    ttbThumbLeftFocused,
    ttbThumbLeftDisabled,
    ttbThumbRightNormal,
    ttbThumbRightHot,
    ttbThumbRightPressed,
    ttbThumbRightFocused,
    ttbThumbRightDisabled,
    ttbThumbTics,
    ttbThumbTicsVert
  );
```

C++

```
enum TThemedTrackBar {
  ttbTrackBarDontCare,
  ttbTrackBarRoot,
  ttbTrack,
  ttbTrackVert,
  ttbThumbNormal,
  ttbThumbHot,
  ttbThumbPressed,
  ttbThumbFocused,
  ttbThumbDisabled,
  ttbThumbBottomNormal,
  ttbThumbBottomHot,
  ttbThumbBottomPressed,
  ttbThumbBottomFocused,
  ttbThumbBottomDisabled,
  ttbThumbTopNormal,
  ttbThumbTopHot,
  ttbThumbTopPressed,
  ttbThumbTopFocused,
  ttbThumbTopDisabled,
  ttbThumbVertNormal,
  ttbThumbVertHot,
  ttbThumbVertPressed,
  ttbThumbVertFocused,
  ttbThumbVertDisabled,
  ttbThumbLeftNormal,
  ttbThumbLeftHot,
  ttbThumbLeftPressed,
  ttbThumbLeftFocused,
  ttbThumbLeftDisabled,
  ttbThumbRightNormal,
  ttbThumbRightHot,
  ttbThumbRightPressed,
  ttbThumbRightFocused,
  ttbThumbRightDisabled,
  ttbThumbTics,
  ttbThumbTicsVert
};
```

File

Themes

Description

This is record Themes.TThemedTrackBar.

44.3.24 Themes.TThemedTrayNotify Enumeration

Pascal

```
TThemedTrayNotify = (
  ttnTrayNotifyDontCare,
  ttnTrayNotifyRoot,
  ttnBackground,
  ttnAnimBackground
);
```

C++

```
enum TThemedTrayNotify {
  ttnTrayNotifyDontCare,
```

```
    ttnTrayNotifyRoot,
    ttnBackground,
    ttnAnimBackground
  };
```

File

Themes

Description

This is record Themes.TThemedTrayNotify.

44.3.25 Themes.TThemedTreeview Enumeration

Pascal

```
  TThemedTreeview = (
    ttTreeviewDontCare,
    ttTreeviewRoot,
    ttItemNormal,
    ttItemHot,
    ttItemSelected,
    ttItemDisabled,
    ttItemSelectedNotFocus,
    ttGlyphClosed,
    ttGlyphOpened,
    ttBranch
  );
```

C++

```
  enum TThemedTreeview {
    ttTreeviewDontCare,
    ttTreeviewRoot,
    ttItemNormal,
    ttItemHot,
    ttItemSelected,
    ttItemDisabled,
    ttItemSelectedNotFocus,
    ttGlyphClosed,
    ttGlyphOpened,
    ttBranch
  };
```

File

Themes

Description

This is record Themes.TThemedTreeview.

44.3.26 Themes.TThemedWindow Enumeration

Pascal

```
  TThemedWindow = (
    twWindowDontCare,
    twWindowRoot,
    twCaptionActive,
```

```
    twCaptionInactive,
    twCaptionDisabled,
    twSmallCaptionActive,
    twSmallCaptionInactive,
    twSmallCaptionDisabled,
    twMinCaptionActive,
    twMinCaptionInactive,
    twMinCaptionDisabled,
    twSmallMinCaptionActive,
    twSmallMinCaptionInactive,
    twSmallMinCaptionDisabled,
    twMaxCaptionActive,
    twMaxCaptionInactive,
    twMaxCaptionDisabled,
    twSmallMaxCaptionActive,
    twSmallMaxCaptionInactive,
    twSmallMaxCaptionDisabled,
    twFrameLeftActive,
    twFrameLeftInactive,
    twFrameRightActive,
    twFrameRightInactive,
    twFrameBottomActive,
    twFrameBottomInactive,
    twSmallFrameLeftActive,
    twSmallFrameLeftInactive,
    twSmallFrameRightActive,
    twSmallFrameRightInactive,
    twSmallFrameBottomActive,
    twSmallFrameBottomInactive,
    twSysButtonNormal,
    twSysButtonHot,
    twSysButtonPushed,
    twSysButtonDisabled,
    twMDISysButtonNormal,
    twMDISysButtonHot,
    twMDISysButtonPushed,
    twMDISysButtonDisabled,
    twMinButtonNormal,
    twMinButtonHot,
    twMinButtonPushed,
    twMinButtonDisabled,
    twMDIMinButtonNormal,
    twMDIMinButtonHot,
    twMDIMinButtonPushed,
    twMDIMinButtonDisabled,
    twMaxButtonNormal,
    twMaxButtonHot,
    twMaxButtonPushed,
    twMaxButtonDisabled,
    twCloseButtonNormal,
    twCloseButtonHot,
    twCloseButtonPushed,
    twCloseButtonDisabled,
    twSmallCloseButtonNormal,
    twSmallCloseButtonHot,
    twSmallCloseButtonPushed,
    twSmallCloseButtonDisabled,
    twMDICloseButtonNormal,
    twMDICloseButtonHot,
    twMDICloseButtonPushed,
    twMDICloseButtonDisabled,
    twRestoreButtonNormal,
    twRestoreButtonHot,
    twRestoreButtonPushed,
    twRestoreButtonDisabled,
    twMDIRestoreButtonNormal,
```

```
    twMDIRestoreButtonHot,
    twMDIRestoreButtonPushed,
    twMDIRestoreButtonDisabled,
    twHelpButtonNormal,
    twHelpButtonHot,
    twHelpButtonPushed,
    twHelpButtonDisabled,
    twMDIHelpButtonNormal,
    twMDIHelpButtonHot,
    twMDIHelpButtonPushed,
    twMDIHelpButtonDisabled,
    twHorzScrollNormal,
    twHorzScrollHot,
    twHorzScrollPushed,
    twHorzScrollDisabled,
    twHorzThumbNormal,
    twHorzThumbHot,
    twHorzThumbPushed,
    twHorzThumbDisabled,
    twVertScrollNormal,
    twVertScrollHot,
    twVertScrollPushed,
    twVertScrollDisabled,
    twVertThumbNormal,
    twVertThumbHot,
    twVertThumbPushed,
    twVertThumbDisabled,
    twDialog,
    twCaptionSizingTemplate,
    twSmallCaptionSizingTemplate,
    twFrameLeftSizingTemplate,
    twSmallFrameLeftSizingTemplate,
    twFrameRightSizingTemplate,
    twSmallFrameRightSizingTemplate,
    twFrameBottomSizingTemplate,
    twSmallFrameBottomSizingTemplate
  );
```

C++

```
  enum TThemedWindow {
    twWindowDontCare,
    twWindowRoot,
    twCaptionActive,
    twCaptionInactive,
    twCaptionDisabled,
    twSmallCaptionActive,
    twSmallCaptionInactive,
    twSmallCaptionDisabled,
    twMinCaptionActive,
    twMinCaptionInactive,
    twMinCaptionDisabled,
    twSmallMinCaptionActive,
    twSmallMinCaptionInactive,
    twSmallMinCaptionDisabled,
    twMaxCaptionActive,
    twMaxCaptionInactive,
    twMaxCaptionDisabled,
    twSmallMaxCaptionActive,
    twSmallMaxCaptionInactive,
    twSmallMaxCaptionDisabled,
    twFrameLeftActive,
    twFrameLeftInactive,
    twFrameRightActive,
    twFrameRightInactive,
    twFrameBottomActive,
    twFrameBottomInactive,
```

```
    twSmallFrameLeftActive,
    twSmallFrameLeftInactive,
    twSmallFrameRightActive,
    twSmallFrameRightInactive,
    twSmallFrameBottomActive,
    twSmallFrameBottomInactive,
    twSysButtonNormal,
    twSysButtonHot,
    twSysButtonPushed,
    twSysButtonDisabled,
    twMDISysButtonNormal,
    twMDISysButtonHot,
    twMDISysButtonPushed,
    twMDISysButtonDisabled,
    twMinButtonNormal,
    twMinButtonHot,
    twMinButtonPushed,
    twMinButtonDisabled,
    twMDIMinButtonNormal,
    twMDIMinButtonHot,
    twMDIMinButtonPushed,
    twMDIMinButtonDisabled,
    twMaxButtonNormal,
    twMaxButtonHot,
    twMaxButtonPushed,
    twMaxButtonDisabled,
    twCloseButtonNormal,
    twCloseButtonHot,
    twCloseButtonPushed,
    twCloseButtonDisabled,
    twSmallCloseButtonNormal,
    twSmallCloseButtonHot,
    twSmallCloseButtonPushed,
    twSmallCloseButtonDisabled,
    twMDICloseButtonNormal,
    twMDICloseButtonHot,
    twMDICloseButtonPushed,
    twMDICloseButtonDisabled,
    twRestoreButtonNormal,
    twRestoreButtonHot,
    twRestoreButtonPushed,
    twRestoreButtonDisabled,
    twMDIRestoreButtonNormal,
    twMDIRestoreButtonHot,
    twMDIRestoreButtonPushed,
    twMDIRestoreButtonDisabled,
    twHelpButtonNormal,
    twHelpButtonHot,
    twHelpButtonPushed,
    twHelpButtonDisabled,
    twMDIHelpButtonNormal,
    twMDIHelpButtonHot,
    twMDIHelpButtonPushed,
    twMDIHelpButtonDisabled,
    twHorzScrollNormal,
    twHorzScrollHot,
    twHorzScrollPushed,
    twHorzScrollDisabled,
    twHorzThumbNormal,
    twHorzThumbHot,
    twHorzThumbPushed,
    twHorzThumbDisabled,
    twVertScrollNormal,
    twVertScrollHot,
    twVertScrollPushed,
    twVertScrollDisabled,
```

```
    twVertThumbNormal,
    twVertThumbHot,
    twVertThumbPushed,
    twVertThumbDisabled,
    twDialog,
    twCaptionSizingTemplate,
    twSmallCaptionSizingTemplate,
    twFrameLeftSizingTemplate,
    twSmallFrameLeftSizingTemplate,
    twFrameRightSizingTemplate,
    twSmallFrameRightSizingTemplate,
    twFrameBottomSizingTemplate,
    twSmallFrameBottomSizingTemplate
  };
```

File

Themes

Description

This is record Themes.TThemedWindow.

44.4 Types

The following table lists types in this documentation.

Types

Name	Description
PThemedElementDetails (see page 4663)	This is type Themes.PThemedElementDetails.
TThemeData (see page 4663)	This is type Themes.TThemeData.
TThemeServicesClass (see page 4663)	This is type Themes.TThemeServicesClass.

44.4.1 Themes.PThemedElementDetails Type

Pascal

```
PThemedElementDetails = ^TThemedElementDetails;
```

C++

```
^TThemedElementDetails PThemedElementDetails;
```

File

Themes

Description

This is type Themes.PThemedElementDetails.

44.4.2 Themes.TThemeData Type

Pascal

```
TThemeData = array[TThemedElement] of HTHEME;
```

C++

```
array[TThemedElement] of HTHEME TThemeData;
```

File

Themes

Description

This is type Themes.TThemeData.

44.4.3 Themes.TThemeServicesClass Type

Pascal

```
TThemeServicesClass = class of TThemeServices;
```

C++

```
TThemeServices TThemeServicesClass;
```

File

Themes

Description

This is type Themes.TThemeServicesClass.

44.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ThemeServicesClass (see page 4665)	This is variable Themes.ThemeServicesClass.

44.5.1 Themes.ThemeServicesClass Variable

Pascal

```
ThemeServicesClass: TThemeServicesClass = TThemeServices;
```

C++

```
TThemeServicesClass ThemeServicesClass = TThemeServices;
```

File

Themes

Description

This is variable Themes.ThemeServicesClass.

45 ToolWin Namespace

This is namespace ToolWin.

Topics

Name	Description
Classes (see page 4668)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4678)	The following table lists structs, records, enums in this documentation.
Types (see page 4681)	The following table lists types in this documentation.

Classes

	Name	Description
	TToolDockForm (see page 4668)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 4671)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 4673)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Structs, Records, Enums

	Name	Description
	TEdgeBorder (see page 4678)	TEdgeBorders (see page 4681) is the type of the EdgeBorders property.
	TEdgeStyle (see page 4679)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 4679)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Types

Name	Description
TEdgeBorders (see page 4681)	This is type ToolWin.TEdgeBorders.

45.1 Classes

The following table lists classes in this documentation.

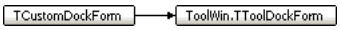
Classes

	Name	Description
	TToolDockForm (see page 4668)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 4671)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 4673)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

45.1.1 TToolDockForm Class

TToolDockForm is a default floating dock site class for tool buttons.

Class Hierarchy



Pascal

```
TToolDockForm = class(TCustomDockForm);
```

C++

```
class TToolDockForm : public TCustomDockForm;
```

File

ToolWin

Description

Assign TToolDockForm to the FloatingDockSiteClass property of tool buttons to get appropriate docking behavior.

45.1.1.1 TToolDockForm Members

The following tables list the members exposed by TToolDockForm.

Protected Methods

	Name	Description
V	CanResize (see page 4669)	Specifies whether the control can resize itself.
V	CreateParams (see page 4670)	Initializes the window-creation parameter record when the form window is created.
V	DoAddDockClient (see page 4670)	Performs actions when a control is docked to the form.
	WMNCCreate (see page 4670)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 4670)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 4671)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 4671)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 4671)	This is WMSysCommand, a member of class TToolDockForm.

Public Methods

	Name	Description
	Create (see page 4669)	Creates an instance of TToolDockForm (see page 4668).

45.1.1.2 TToolDockForm.Create Constructor

Creates an instance of TToolDockForm (see page 4668).

Pascal

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

C++

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

Description

AOwner is the dock form's owner, which is responsible for freeing the tool dock form.

The constructor initializes the BorderWidth property to 1.

See Also

BorderWidth

Owner

45.1.1.3 TToolDockForm Methods

The methods of the TToolDockForm class are listed here.

Protected Methods

	Name	Description
	CanResize (see page 4669)	Specifies whether the control can resize itself.
	CreateParams (see page 4670)	Initializes the window-creation parameter record when the form window is created.
	DoAddDockClient (see page 4670)	Performs actions when a control is docked to the form.
	WMNCCreate (see page 4670)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 4670)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 4671)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 4671)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 4671)	This is WMSysCommand, a member of class TToolDockForm.

45.1.1.3.1 TToolDockForm.CanResize Method

Specifies whether the control can resize itself.

Pascal

```
function CanResize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanResize(int NewWidth, int NewHeight);
```

Description

CanResize is called automatically when an attempt is made to resize the control, after any autosizing has occurred. The NewWidth and NewHeight parameters specify the attempted new values for the control's height and width. CanResize optionally

adjusts the values of NewWidth and NewHeight, and returns true if the control can be resized to the returned values. CanResize returns false if the control can't be resized.

45.1.1.3.2 TToolDockForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams calls the inherited CreateParams method, and then adds WS_POPUP to the Style property.

45.1.1.3.3 TToolDockForm.DoAddDockClient Method

Performs actions when a control is docked to the form.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient sets the client's Align property to alNone and its BoundsRect property to ClientRect. Then DoAddDockClient calls the inherited DoAddDockClient method and sets the form's AutoSize property to true.

45.1.1.3.4 TToolDockForm.WMNCCreate Method

Pascal

```
procedure WMNCCreate(var Message: TWMNCCreate); message WM_NCCREATE;
```

C++

```
__fastcall WMNCCreate(TWMNCCreate Message);
```

Description

This is WMNCCreate, a member of class TToolDockForm.

45.1.1.3.5 TToolDockForm.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TToolDockForm.

45.1.1.3.6 TToolDockForm.WMNCLButtonDown Method

Pascal

```
procedure WMNCLButtonDown(var Message: TWMNCLButtonDown); message WM_NCLBUTTONDOWN;
```

C++

```
__fastcall WMNCLButtonDown(TWMNCLButtonDown Message);
```

Description

This is WMNCLButtonDown, a member of class TToolDockForm.

45.1.1.3.7 TToolDockForm.WMSize Method

Pascal

```
procedure WMSize(var Message: TWMSize); message WM_SIZE;
```

C++

```
__fastcall WMSize(TWMSize Message);
```

Description

This is WMSize, a member of class TToolDockForm.

45.1.1.3.8 TToolDockForm.WMSysCommand Method

Pascal

```
procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
```

C++

```
__fastcall WMSysCommand(TWMSysCommand Message);
```

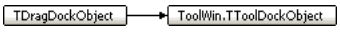
Description

This is WMSysCommand, a member of class TToolDockForm.

45.1.2 TToolDockObject Class

Handles drag-and-dock operations for toolbar controls.

Class Hierarchy



Pascal

```
TToolDockObject = class(TDragDockObject);
```

C++

```
class TToolDockObject : public TDragDockObject;
```

File

ToolWin

Description

Use TToolDockObject to handle drag-and-dock operations for toolbar controls that are docked to a TControlBar object.

Note: TToolDockObject is designed for tool bars that are docked to TControlBar. It may not work as expected when the application is docking a tool bar to a form or panel.

See Also

TToolBar

TControlBar

45.1.2.1 TToolDockObject Members

The following tables list the members exposed by TToolDockObject.

Protected Methods

	Name	Description
	AdjustDockRect (see page 4673)	This is AdjustDockRect, a member of class TToolDockObject.
	DrawDragDockImage (see page 4673)	Called when the dock image for a toolbar control is drawn.
	EraseDragDockImage (see page 4673)	Called when the dock image for a toolbar control is erased.

Public Methods

	Name	Description
	Create (see page 4672)	Creates and initializes an instance of TDragDockObject.

45.1.2.2 TToolDockObject.Create Constructor

Creates and initializes an instance of TDragDockObject.

Pascal

```
constructor Create(AControl: TControl); override;
```

C++

```
virtual __fastcall TToolDockObject(TControl * AControl);
```

Description

Create calls the inherited constructor and allocates memory for the Brush property.

45.1.2.3 TToolDockObject Methods

The methods of the TToolDockObject class are listed here.

Protected Methods

	Name	Description
	AdjustDockRect (see page 4673)	This is AdjustDockRect, a member of class TToolDockObject.
	DrawDragDockImage (see page 4673)	Called when the dock image for a toolbar control is drawn.
	EraseDragDockImage (see page 4673)	Called when the dock image for a toolbar control is erased.

45.1.2.3.1 TToolDockObject.AdjustDockRect Method

Pascal

```
procedure AdjustDockRect(ARect: TRect); override;
```

C++

```
virtual __fastcall AdjustDockRect(TRect ARect);
```

Description

This is AdjustDockRect, a member of class TToolDockObject.

45.1.2.3.2 TToolDockObject.DrawDragDockImage Method

Called when the dock image for a toolbar control is drawn.

Pascal

```
procedure DrawDragDockImage; override;
```

C++

```
virtual __fastcall DrawDragDockImage();
```

Description

DrawDragDockImage draws a control that is moved in a toolbar.

45.1.2.3.3 TToolDockObject.EraseDragDockImage Method

Called when the dock image for a toolbar control is erased.

Pascal

```
procedure EraseDragDockImage; override;
```

C++

```
virtual __fastcall EraseDragDockImage();
```

Description

EraseDragDockImage erases a control that is moved in a toolbar.

45.1.3 TToolWindow Class

TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Class Hierarchy

Pascal

```
TToolWindow = class(TWinControl);
```

C++

```
class TToolWindow : public TWinControl;
```

File

ToolWin

Description

TToolWindow is the ancestor of the TCoolBar and TToolBar components, which facilitate the arrangement of buttons and other visual controls.

See Also

TCoolBar

TToolBar

45.1.3.1 TToolWindow Members

The following tables list the members exposed by TToolWindow.

Protected Methods

	Name	Description
	CMBorderChanged (see page 4675)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 4675)	This is CMCtl3DChanged, a member of class TToolWindow.
V	NCPaint (see page 4675)	This is NCPaint, a member of class TToolWindow.
	WMNCCalcSize (see page 4675)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 4676)	This is WMNCPaint, a member of class TToolWindow.

Public Methods

	Name	Description
V	Create (see page 4674)	Creates and initializes a TToolWindow (see page 4673) instance.

Public Properties

	Name	Description
	EdgeBorders (see page 4676)	Determines which sides of the control have visible borders.
	EdgeInner (see page 4676)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 4677)	Determines the border style for the control's outer edge.

45.1.3.2 TToolWindow.Create Constructor

Creates and initializes a TToolWindow (see page 4673) instance.

Pascal

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

C++

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

Description

Create generates a TToolWindow (see page 4673) and initializes its properties.

45.1.3.3 TToolWindow Methods

The methods of the TToolWindow class are listed here.

Protected Methods

	Name	Description
	CMBorderChanged (see page 4675)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 4675)	This is CMCtl3DChanged, a member of class TToolWindow.
V	NCPaint (see page 4675)	This is NCPaint, a member of class TToolWindow.
	WMNCCalcSize (see page 4675)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 4676)	This is WMNCPaint, a member of class TToolWindow.

45.1.3.3.1 TToolWindow.CMBorderChanged Method

Pascal

```
procedure CMBorderChanged(var Message: TMessage); message CM_BORDERCHANGED;
```

C++

```
__fastcall CMBorderChanged(TMessage Message);
```

Description

This is CMBorderChanged, a member of class TToolWindow.

45.1.3.3.2 TToolWindow.CMCtl3DChanged Method

Pascal

```
procedure CMCtl3DChanged(var Message: TMessage); message CM_CTL3DCHANGED;
```

C++

```
__fastcall CMCtl3DChanged(TMessage Message);
```

Description

This is CMCtl3DChanged, a member of class TToolWindow.

45.1.3.3.3 TToolWindow.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); virtual;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TToolWindow.

45.1.3.3.4 TToolWindow.WMNCCalcSize Method

Pascal

```
procedure WMNCCalcSize(var Message: TWMNCCalcSize); message WM_NCCALCSIZE;
```

C++

```
__fastcall WMNCCalcSize(TWMNCCalcSize Message);
```

Description

This is WMNCCalcSize, a member of class TToolWindow.

45.1.3.3.5 TToolWindow.WMNCPaint Method

Pascal

```
procedure WMNCPaint(var Message: TWMNCPaint); message WM_NCPAINT;
```

C++

```
__fastcall WMNCPaint(TWMNCPaint Message);
```

Description

This is WMNCPaint, a member of class TToolWindow.

45.1.3.4 TToolWindow Properties

The properties of the TToolWindow class are listed here.

Public Properties

	Name	Description
	EdgeBorders (see page 4676)	Determines which sides of the control have visible borders.
	EdgeInner (see page 4676)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 4677)	Determines the border style for the control's outer edge.

45.1.3.4.1 TToolWindow.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner (see page 4676) and EdgeOuter (see page 4677) properties.

See Also

EdgeInner (see page 4676)

EdgeOuter (see page 4677)

45.1.3.4.2 TToolWindow.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side,

an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders (see page 4676) property.

Use EdgeInner with EdgeOuter (see page 4677) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 4676)

EdgeOuter (see page 4677)

45.1.3.4.3 TToolWindow.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner (see page 4676) property has an effect only on the sides specified by the EdgeBorders (see page 4676) property.

Use EdgeOuter with EdgeInner (see page 4676) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 4676)

EdgeInner (see page 4676)

45.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgeBorder (see page 4678)	TEdgeBorders (see page 4681) is the type of the EdgeBorders property.
	TEdgeStyle (see page 4679)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 4679)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

45.2.1 ToolWin.TEdgeBorder Enumeration

TEdgeBorders (see page 4681) is the type of the EdgeBorders property.

Pascal

```
TEdgeBorder = (
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
);
```

C++

```
enum TEdgeBorder {
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
};
```

File

ToolWin

Description

TEdgeBorders (see page 4681) is a set of TEdgeBorder values. Each TEdgeBorder value indicates whether a beveled edge appears on a particular side of a control. The following table lists the TEdgeBorder values:

Value	Meaning
ebLeft	The control has a beveled left edge.
ebTop	The control has a beveled top edge.
ebRight	The control has a beveled right edge.
ebBottom	The control has a beveled bottom edge.

In addition to the values in the preceding table, the QGraphics unit defines the ebRect constant to represent all the edges of a control:

const ebRect = [ebLeft, ebTop, ebRight, ebBottom];

#define ebRect (System::Set<TEdgeBorder, ebLeft, ebBottom> () << TEdgeBorder(0) << TEdgeBorder(1) << TEdgeBorder(2)

<< TEdgeBorder(3))

45.2.2 ToolWin.TEdgeStyle Enumeration

TEdgeStyle describes the appearance of an edge border.

Pascal

```
TEdgeStyle = (
  esNone,
  esRaised,
  esLowered
);
```

C++

```
enum TEdgeStyle {
  esNone,
  esRaised,
  esLowered
};
```

File

ToolWin

Description

TEdgeStyle indicates the appearance of a particular border. The following table lists the possible values:

Value	Meaning
esNone	The border is not drawn.
esRaised	The border appears raised.
esLowered	The border appears lowered.

45.2.3 ToolWin.TSizingOrientation Enumeration

TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Pascal

```
TSizingOrientation = (
  soNone,
  soHorizontal,
  soVertical
);
```

C++

```
enum TSizingOrientation {
  soNone,
  soHorizontal,
  soVertical
};
```

File

ToolWin

Description

TSizingOrientation can have one of the following values:

Value	Meaning
soNone	The control's size is not readjusted when it is docked.
soHorizontal	The control's width is adjusted when it is docked.
soVertical	The control's height is adjusted when it is docked.

45.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEdgeBorders (see page 4681)	This is type ToolWin.TEdgeBorders.

45.3.1 ToolWin.TEdgeBorders Type

Pascal

```
TEdgeBorders = set of TEdgeBorder;
```

C++

```
TEdgeBorder TEdgeBorders;
```

File

ToolWin

Description

This is type ToolWin.TEdgeBorders.

46 Types Namespace

This is namespace Types.

Topics

Name	Description
Functions (see page 4685)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4693)	The following table lists structs, records, enums in this documentation.
Types (see page 4697)	The following table lists types in this documentation.
Constants (see page 4709)	The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 4709)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 4709)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 4710)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 4710)	This is constant Types.RT_RCDATA.

Functions

	Name	Description
	Bounds (see page 4685)	Returns the TRect (see page 4694) for a rectangle of given dimensions.
	CenteredRect (see page 4686)	This is function Types.CenteredRect.
	CenterPoint (see page 4686)	Returns the point in the center of a rectangle.
	EqualRect (see page 4686)	Indicates whether two TRect (see page 4694) values are the same.
	IntersectRect (see page 4687)	Returns the intersection of two rectangles
	IsRectEmpty (see page 4687)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 4688)	Changes the origin of a rectangle by a specified amount.
	Point (see page 4688)	Creates a TPoint (see page 4694) structure from a pair of coordinates.
	PtInRect (see page 4689)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 4689)	Creates a TRect (see page 4694) structure from a set of coordinates.
	RectHeight (see page 4690)	This is function Types.RectHeight.
	RectWidth (see page 4690)	This is function Types.RectWidth.
	SmallPoint (see page 4690)	Creates a TSmallPoint (see page 4695) structure from a pair of coordinates.
	SmallPoint (see page 4690)	Creates a TSmallPoint (see page 4695) structure from a pair of coordinates.
	SplitRect (see page 4691)	This is function Types.SplitRect.

	SplitRect (see page 4691)	This is function Types.SplitRect.
	UnionRect (see page 4692)	Returns the union of two rectangles

Structs, Records, Enums

	Name	Description
	tagSIZE (see page 4693)	tagSIZE is a record holding width and height values.
	TPoint (see page 4694)	TPoint type defines a pixel location onscreen.
	TRect (see page 4694)	TRect defines a rectangle.
	TSmallPoint (see page 4695)	TSmallPoint type defines a point with two 16-bit coordinates.
	TSplitRectType (see page 4696)	This is record Types.TSplitRectType.

Types

Name	Description
DWORD (see page 4697)	DWORD is equivalent to LongWord.
PByte (see page 4698)	PByte is a pointer to a Byte variable.
PDouble (see page 4698)	PDouble is a pointer to a Double variable.
PInteger (see page 4699)	PInteger is a pointer to a Integer variable.
PLongint (see page 4699)	This is type Types.PLongint.
PPoint (see page 4699)	PPoint is a pointer to a TPoint (see page 4694) variable.
PRect (see page 4700)	PRect is a pointer to a TRect (see page 4694) variable.
PSize (see page 4700)	PSize is a pointer to a TSize (see page 4706) variable.
PSmallInt (see page 4701)	This is type Types.PSmallInt.
PSmallPoint (see page 4701)	This is type Types.PSmallPoint.
SIZE (see page 4701)	This is type Types.SIZE.
tagPOINT (see page 4702)	This is type Types.tagPOINT.
TBooleanDynArray (see page 4702)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 4702)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 4703)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 4703)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 4704)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 4704)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 4704)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 4705)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 4705)	TSingleDynArray defines an array of Single elements.
TSize (see page 4706)	TSize represents a width and height.
TSmallIntDynArray (see page 4706)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 4706)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 4707)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 4707)	TWideStringDynArray defines an array of WideString (see page 3476) elements.
TWordDynArray (see page 4708)	TWordDynArray defines an array of Word elements.

46.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Bounds (see page 4685)	Returns the TRect (see page 4694) for a rectangle of given dimensions.
	CenteredRect (see page 4686)	This is function Types.CenteredRect.
	CenterPoint (see page 4686)	Returns the point in the center of a rectangle.
	EqualRect (see page 4686)	Indicates whether two TRect (see page 4694) values are the same.
	IntersectRect (see page 4687)	Returns the intersection of two rectangles
	IsRectEmpty (see page 4687)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 4688)	Changes the origin of a rectangle by a specified amount.
	Point (see page 4688)	Creates a TPoint (see page 4694) structure from a pair of coordinates.
	PtInRect (see page 4689)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 4689)	Creates a TRect (see page 4694) structure from a set of coordinates.
	RectHeight (see page 4690)	This is function Types.RectHeight.
	RectWidth (see page 4690)	This is function Types.RectWidth.
	SmallPoint (see page 4690)	Creates a TSmallPoint (see page 4695) structure from a pair of coordinates.
	SmallPoint (see page 4690)	Creates a TSmallPoint (see page 4695) structure from a pair of coordinates.
	SplitRect (see page 4691)	This is function Types.SplitRect.
	SplitRect (see page 4691)	This is function Types.SplitRect.
	UnionRect (see page 4692)	Returns the union of two rectangles

46.1.1 Types.Bounds Function

Returns the TRect (see page 4694) for a rectangle of given dimensions.

Pascal

```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer): TRect;
```

C++

```
TRect Bounds(int ALeft, int ATop, int AWidth, int AHeight);
```

File

Types

Description

Call Bounds to obtain a TRect (see page 4694) for a rectangle with upper left corner at the point (ALeft, ATop) with width AWidth and height AHeight.

See Also

Rect (see page 4689)

46.1.2 Types.CenteredRect Function

Pascal

```
function CenteredRect(SourceRect: TRect; CenteredRect: TRect): TRect;
```

C++

```
TRect CenteredRect(TRect SourceRect, TRect CenteredRect);
```

File

Types

Description

This is function Types.CenteredRect.

46.1.3 Types.CenterPoint Function

Returns the point in the center of a rectangle.

Pascal

```
function CenterPoint(const Rect: TRect): TPoint;
```

C++

```
TPoint CenterPoint(const TRect Rect);
```

File

Types

Description

Call CenterPoint to obtain the point in the center of a rectangle.

Rect (see page 4689) is the rectangle whose center you want to locate.

See Also

PtInRect (see page 4689)

IsRectEmpty (see page 4687)

UnionRect (see page 4692)

IntersectRect (see page 4687)

46.1.4 Types.EqualRect Function

Indicates whether two TRect (see page 4694) values are the same.

Pascal

```
function EqualRect(const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean EqualRect(const TRect R1, const TRect R2);
```

File

Types

Description

Call EqualRect to compare two TRect (see page 4694) values. EqualRect returns true if R1 and R2 are identical (same position and size), false if they are different.

See Also

Rect (see page 4689)

Bounds (see page 4685)

IntersectRect (see page 4687)

46.1.5 Types.IntersectRect Function

Returns the intersection of two rectangles

Pascal

```
function IntersectRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean IntersectRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call IntersectRect to obtain the rectangle that is the intersection of R1 and R2. The intersection is returned as the Rect (see page 4689) parameter.

IntersectRect returns true if the two rectangles have a nonempty intersection. If R1 and R2 do not overlap, IntersectRect returns false, and the Rect (see page 4689) parameter is set to a rectangle at position (0,0) with 0 Width and 0 Height.

See Also

PtInRect (see page 4689)

IsRectEmpty (see page 4687)

UnionRect (see page 4692)

46.1.6 Types.IsRectEmpty Function

Indicates wither a specified rectangle has a positive width and height.

Pascal

```
function IsRectEmpty(const Rect: TRect): Boolean;
```

C++

```
Boolean IsRectEmpty(const TRect Rect);
```

File

Types

Description

Call IsRectEmpty to determine whether the rectangle specified by Rect (see page 4689) encloses a positive amount of space.

IsRectEmpty returns true if both the width and height of Rect (see page 4689) are greater than 0. If either the width or height is less than or equal to 0, IsRectEmpty returns false. Note that this means non-normalized rectangles (rectangles with negative height or width) are considered empty.

See Also

PtInRect (see page 4689)

IntersectRect (see page 4687)

46.1.7 Types.OffsetRect Function

Changes the origin of a rectangle by a specified amount.

Pascal

```
function OffsetRect(var Rect: TRect; DX: Integer; DY: Integer): Boolean;
```

C++

```
Boolean OffsetRect(TRect Rect, int DX, int DY);
```

File

Types

Description

Call OffsetRect to move the upper left corner of the rectangle specified by Rect (see page 4689) while maintaining the same width and height. The left of the rectangle moves by DX while the top moves by DY.

OffsetRect returns true if the Rect (see page 4689) parameter is not nil (Delphi) or NULL (C++), false if it can't return a new rectangle because Rect (see page 4689) is nil(Delphi) or NULL (C++).

See Also

Rect (see page 4689)

Bounds (see page 4685)

46.1.8 Types.Point Function

Creates a TPoint (see page 4694) structure from a pair of coordinates.

Pascal

```
function Point(X: Integer; Y: Integer): TPoint;
```

C++

```
TPoint Point(int X, int Y);
```

File

Types

Description

Call Point to create a TPoint (see page 4694) that represents the specified coordinates. Use Point to construct parameters for functions that require a point, rather than setting up local variables for each parameter.

See Also

SmallPoint (see page 4690)

Rect (see page 4689)

46.1.9 Types.PtInRect Function

Indicates whether a specified point lies inside a specified rectangle.

Pascal

```
function PtInRect(const Rect: TRect; const P: TPoint): Boolean;
```

C++

```
Boolean PtInRect(const TRect Rect, const TPoint P);
```

File

Types

Description

PtInRect returns true if the point, P, lies inside the rectangle, Rect (see page 4689), and false if P is not in Rect (see page 4689).

PtInRect does not consider a point inside a rectangle if the rectangle has a negative width or height. A point is considered inside a rectangle if it lies on the left or top edge, but not if it lies on the right or bottom edge.

See Also

IntersectRect (see page 4687)

IsRectEmpty (see page 4687)

CenterPoint (see page 4686)

46.1.10 Types.Rect Function

Creates a TRect (see page 4694) structure from a set of coordinates.

Pascal

```
function Rect(Left: Integer; Top: Integer; Right: Integer; Bottom: Integer): TRect;
```

C++

```
TRect Rect(int Left, int Top, int Right, int Bottom);
```

File

Types

Description

Call Rect to create a TRect (see page 4694) that represents the rectangle with the specified coordinates. Use Rect to construct parameters for functions that require TRect (see page 4694), rather than setting up local variables for each parameter.

See Also

Bounds (see page 4685)

Point (see page 4688)

46.1.11 Types.RectHeight Function

Pascal

```
function RectHeight(const Rect: TRect): Integer;
```

C++

```
int RectHeight(const TRect Rect);
```

File

Types

Description

This is function Types.RectHeight.

46.1.12 Types.RectWidth Function

Pascal

```
function RectWidth(const Rect: TRect): Integer;
```

C++

```
int RectWidth(const TRect Rect);
```

File

Types

Description

This is function Types.RectWidth.

46.1.13 Types.SmallPoint Function

Creates a TSmallPoint (see page 4695) structure from a pair of coordinates.

Pascal

```
function SmallPoint(X: Integer; Y: Integer): TSmallPoint; overload;
function SmallPoint(XY: LongWord): TSmallPoint; overload;
```

C++

```
TSmallPoint SmallPoint(int X, int Y);
TSmallPoint SmallPoint(LongWord XY);
```

File

Types

Description

Call SmallPoint to create a TSmallPoint (see page 4695) that represents the specified coordinates.

See Also

Point

46.1.14 Types.SplitRect Function

Pascal

```
function SplitRect(Rect: TRect; SplitType: TSplitRectType; Percent: Double): TRect; overload;
```

C++

```
TRect SplitRect(TRect Rect, TSplitRectType SplitType, Double Percent);
```

File

Types

Description

This is function Types.SplitRect.

46.1.15 Types.SplitRect Function

Pascal

```
function SplitRect(Rect: TRect; SplitType: TSplitRectType; Size: Integer): TRect; overload;
```

C++

```
TRect SplitRect(TRect Rect, TSplitRectType SplitType, int Size);
```

File

Types

Description

This is function Types.SplitRect.

46.1.16 Types.UnionRect Function

Returns the union of two rectangles

Pascal

```
function UnionRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean UnionRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call UnionRect to obtain the smallest rectangle that includes both R1 and R2. The union is returned as the Rect (see page 4689) parameter.

UnionRect returns true if the resulting rectangle is nonempty. If the resulting rectangle as a width or height that is less than or equal to 0, UnionRect returns false.

See Also

IntersectRect (see page 4687)

OffsetRect (see page 4688)

IsRectEmpty (see page 4687)

46.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSplitRectType (see page 4696)	This is record Types.TSplitRectType.

Records

	Name	Description
	tagSIZE (see page 4693)	tagSIZE is a record holding width and height values.
	TPoint (see page 4694)	TPoint type defines a pixel location onscreen.
	TRect (see page 4694)	TRect defines a rectangle.
	TSmallPoint (see page 4695)	TSmallPoint type defines a point with two 16-bit coordinates.

46.2.1 Types.tagSIZE Record

tagSIZE is a record holding width and height values.

Pascal

```
tagSIZE = record
  cx: Longint;
  cy: Longint;
end;
```

C++

```
struct tagSIZE {
  Longint cx;
  Longint cy;
};
```

File

Types

Description

The SIZE (see page 4701) equates to a TSize (see page 4706) type, which in turn equates to a tagSIZE record.

tagSIZE holds the width (cx) and height (cy) of something, typically in pixels. For example, TextExtent returns the width and height, in pixels, of a string rendered in the current font.

See Also

SIZE (see page 4701)

TextExtent

TSize (see page 4706)

46.2.2 Types.TPoint Record

TPoint type defines a pixel location onscreen.

Pascal

```
TPoint = record
  X: Longint;
  Y: Longint;
end;
```

C++

```
struct TPoint {
  Longint X;
  Longint Y;
};
```

File

Types

Description

The TPoint type defines a pixel location onscreen, with the origin in the top left corner. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Brush->Color = clTeal;
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Polygon(points, 3);
}
```

See Also

Point (see page 4688)

46.2.3 Types.TRect Record

TRect defines a rectangle.

Pascal

```
TRect = record
  case Integer of
    0: (Bottom: Longint;
        Left: Longint;
        Right: Longint;
        Top: Longint;);
```

```
      1: (BottomRight: TPoint;
          TopLeft: TPoint;);
  end;
```

C++

```
  struct TRect {
    Longint Bottom;
    Longint Left;
    Longint Right;
    Longint Top;
    TPoint BottomRight;
    TPoint TopLeft;
  };
```

File

Types

Description

TRect represents the dimensions of a rectangle. The coordinates are specified as either four separate integers representing the left, top, right, and bottom sides, or as two points representing the locations of the top left and bottom right corners.

Typically, TRect values represent pixel locations, where the origin of the pixel coordinate system is in the top left corner of the screen (screen coordinates) or the top left corner of a control's client area (client coordinates). When a TRect value represents a rectangle on the screen, by convention the top and left edges are considered inside the rectangle and the bottom and right edges are considered outside the rectangle. This convention allows the width of the rectangle to be Right - Left and the height to be Bottom - Top.

46.2.4 Types.TSmallPoint Record

TSmallPoint type defines a point with two 16-bit coordinates.

Pascal

```
  TSmallPoint = record
    x: SmallInt;
    y: SmallInt;
  end;
```

C++

```
  struct TSmallPoint {
    SmallInt x;
    SmallInt y;
  };
```

File

Types

Description

The TSmallPoint type defines a point where each coordinate is a 16-bit integer. x specifies the horizontal coordinate of the point, y specifies the vertical coordinate.

46.2.5 Types.TSplitRectType Enumeration

Pascal

```
TSplitRectType = (
  srLeft,
  srRight,
  srTop,
  srBottom
);
```

C++

```
enum TSplitRectType {
  srLeft,
  srRight,
  srTop,
  srBottom
};
```

File

Types

Description

This is record Types.TSplitRectType.

46.3 Types

The following table lists types in this documentation.

Types

Name	Description
DWORD (see page 4697)	DWORD is equivalent to LongWord.
PByte (see page 4698)	PByte is a pointer to a Byte variable.
PDouble (see page 4698)	PDouble is a pointer to a Double variable.
PInteger (see page 4699)	PInteger is a pointer to a Integer variable.
PLongint (see page 4699)	This is type Types.PLongint.
PPoint (see page 4699)	PPoint is a pointer to a TPoint (see page 4694) variable.
PRect (see page 4700)	PRect is a pointer to a TRect (see page 4694) variable.
PSize (see page 4700)	PSize is a pointer to a TSize (see page 4706) variable.
PSmallInt (see page 4701)	This is type Types.PSmallInt.
PSmallPoint (see page 4701)	This is type Types.PSmallPoint.
SIZE (see page 4701)	This is type Types.SIZE.
tagPOINT (see page 4702)	This is type Types.tagPOINT.
TBooleanDynArray (see page 4702)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 4702)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 4703)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 4703)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 4704)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 4704)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 4704)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 4705)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 4705)	TSingleDynArray defines an array of Single elements.
TSize (see page 4706)	TSize represents a width and height.
TSmallIntDynArray (see page 4706)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 4706)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 4707)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 4707)	TWideStringDynArray defines an array of WideString (see page 3476) elements.
TWordDynArray (see page 4708)	TWordDynArray defines an array of Word elements.

46.3.1 Types.DWORD Type

DWORD is equivalent to LongWord.

Pascal

```
DWORD = LongWord;
```

C++

```
LongWord DWORD;
```

File

Types

Description

The DWORD holds positive integers up to a maximum value of 4,294,967,295.

See Also

LongWord

46.3.2 Types.PByte Type

PByte is a pointer to a Byte variable.

Pascal

```
PByte = System.PByte;
```

C++

```
System.PByte PByte;
```

File

Types

Description

Types.PByte is the same as System.PByte (see page 3830).

See Also

Byte

PByte (see page 3830)

46.3.3 Types.PDouble Type

PDouble is a pointer to a Double variable.

Pascal

```
PDouble = System.PDouble;
```

C++

```
System.PDouble PDouble;
```

File

Types

Description

Types.PDouble is the same as System.PDouble (see page 3834).

See Also

Double

PDouble (see page 3834)

46.3.4 Types.PInteger Type

PInteger is a pointer to a Integer variable.

Pascal

```
PInteger = System.PInteger;
```

C++

```
System.PInteger PInteger;
```

File

Types

Description

Types.PInteger is the same as System.PInteger (see page 3837).

See Also

Integer

PInteger (see page 3837)

46.3.5 Types.PLongint Type

Pascal

```
PLongint = System.PLongint;
```

C++

```
System.PLongint PLongint;
```

File

Types

Description

This is type Types.PLongint.

46.3.6 Types.PPoint Type

PPoint is a pointer to a TPoint (see page 4694) variable.

Pascal

```
PPoint = ^TPoint;
```

C++

```
^TPoint PPoint;
```

File

Types

Description

The TPoint (see page 4694) type defines a pixel location onscreen, with the origin in the top left corner.

See Also

TPoint (see page 4694)

46.3.7 Types.PRect Type

PRect is a pointer to a TRect (see page 4694) variable.

Pascal

```
PRect = ^TRect;
```

C++

```
^TRect PRect;
```

File

Types

Description

The TRect (see page 4694) type represents the dimensions of a rectangle.

See Also

TRect (see page 4694)

46.3.8 Types.PSize Type

PSize is a pointer to a TSize (see page 4706) variable.

Pascal

```
PSize = ^TSize;
```

C++

```
^TSize PSize;
```

File

Types

Description

The TSize (see page 4706) type holds the width and height of something, typically in pixels.

See Also

TSize (see page 4706)

46.3.9 Types.PSmallInt Type

Pascal

```
PSmallInt = System.PSmallInt;
```

C++

```
System.PSmallInt PSmallInt;
```

File

Types

Description

This is type Types.PSmallInt.

46.3.10 Types.PSmallPoint Type

Pascal

```
PSmallPoint = ^TSmallPoint;
```

C++

```
^TSmallPoint PSmallPoint;
```

File

Types

Description

This is type Types.PSmallPoint.

46.3.11 Types.SIZE Type

Pascal

```
SIZE = tagSIZE;
```

C++

```
tagSIZE SIZE;
```

File

Types

Description

This is type Types.SIZE.

46.3.12 Types.tagPOINT Type

Pascal

```
tagPOINT = TPoint;
```

C++

```
TPoint tagPOINT;
```

File

Types

Description

This is type Types.tagPOINT.

46.3.13 Types.TBooleanDynArray Type

TBooleanDynArray defines an array of Boolean elements.

Pascal

```
TBooleanDynArray = array of Boolean;
```

C++

```
array of Boolean TBooleanDynArray;
```

File

Types

Description

The TBooleanDynArray defines a dynamic array of Boolean elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.14 Types.TByteDynArray Type

Pascal

```
TByteDynArray = array of Byte;
```

C++

```
array of Byte TByteDynArray;
```

File

Types

Description

This is type Types.TByteDynArray.

46.3.15 Types.TCardinalDynArray Type

TCardinalDynArray defines an array of Cardinal elements.

Pascal

```
TCardinalDynArray = array of Cardinal;
```

C++

```
array of Cardinal TCardinalDynArray;
```

File

Types

Description

The TCardinalDynArray defines a dynamic array of Cardinal elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.16 Types.TDoubleDynArray Type

TDoubleDynArray defines an array of Double elements.

Pascal

```
TDoubleDynArray = array of Double;
```

C++

```
array of Double TDoubleDynArray;
```

File

Types

Description

The TDoubleDynArray defines a dynamic array of Double elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.17 Types.TInt64DynArray Type

TInt64DynArray defines an array of Int64 elements.

Pascal

```
TInt64DynArray = array of Int64;
```

C++

```
array of Int64 TInt64DynArray;
```

File

Types

Description

The TInt64DynArray defines a dynamic array of Int64 elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.18 Types.TIntegerDynArray Type

Pascal

```
TIntegerDynArray = array of Integer;
```

C++

```
array of Integer TIntegerDynArray;
```

File

Types

Description

This is type Types.TIntegerDynArray.

46.3.19 Types.TLongWordDynArray Type

TLongWordDynArray defines an array of LongWord elements.

Pascal

```
TLongWordDynArray = array of LongWord;
```

C++

```
array of LongWord TLongWordDynArray;
```

File

Types

Description

The TLongWordDynArray defines a dynamic array of LongWord elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.20 Types.TShortIntDynArray Type

TShortIntDynArray defines an array of ShortInt elements.

Pascal

```
TShortIntDynArray = array of ShortInt;
```

C++

```
array of ShortInt TShortIntDynArray;
```

File

Types

Description

The TShortIntDynArray defines a dynamic array of ShortInt elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.21 Types.TSingleDynArray Type

TSingleDynArray defines an array of Single elements.

Pascal

```
TSingleDynArray = array of Single;
```

C++

```
array of Single TSingleDynArray;
```

File

Types

Description

The TSingleDynArray defines a dynamic array of Single elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.22 Types.TSize Type

TSize represents a width and height.

Pascal

```
TSize = tagSIZE;
```

C++

```
tagSIZE TSize;
```

File

Types

Description

TSize represents the size of something. Typically, the implied units of measurement are pixels.

cx is the width.

cy is the height.

46.3.23 Types.TSmallIntDynArray Type

TSmallIntDynArray defines an array of SmallInt elements.

Pascal

```
TSmallIntDynArray = array of SmallInt;
```

C++

```
array of SmallInt TSmallIntDynArray;
```

File

Types

Description

The TSmallIntDynArray defines a dynamic array of SmallInt elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.24 Types.TStringDynArray Type

TStringDynArray defines an array of String elements.

Pascal

```
TStringDynArray = array of string;
```

C++

```
array of string TStringDynArray;
```

File

Types

Description

The TStringDynArray defines a dynamic array of String elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.25 Types.TValueRelationship Type

TValueRelationship defines value comparison possibilities.

Pascal

```
TValueRelationship = -1..1;
```

C++

```
-1..1 TValueRelationship;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship value as follows :

Value	Associated constant	Meaning
-1	LessThanValue (see page 4710)	The first value is less than the second value.
0	EqualsValue (see page 4709)	The two values are equal.
+1	GreaterThanValue (see page 4709)	The first value is greater than the second value.

See Also

LessThanValue (see page 4710)

EqualsValue (see page 4709)

GreaterThanValue (see page 4709)

TValueRelationship

46.3.26 Types.TWideStringDynArray Type

TWideStringDynArray defines an array of WideString (see page 3476) elements.

Pascal

```
TWideStringDynArray = array of WideString;
```

C++

```
array of WideString TWideStringDynArray;
```

File

Types

Description

The TWideStringDynArray defines a dynamic array of WideString (see page 3476) elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.3.27 Types.TWordDynArray Type

TWordDynArray defines an array of Word elements.

Pascal

```
TWordDynArray = array of Word;
```

C++

```
array of Word TWordDynArray;
```

File

Types

Description

The TWordDynArray defines a dynamic array of Word elements.

Use SetLength (see page 3748) to allocate storage for such an array.

See Also

TSize (see page 4706)

46.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 4709)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 4709)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 4710)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 4710)	This is constant Types.RT_RCDATA.

46.4.1 Types.EqualsValue Constant

EqualsValue indicates that both values are equal in a comparison operation.

Pascal

```
EqualsValue = 0;
```

C++

```
EqualsValue = 0;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4707) value. If the first value is equal to the second, the value is EqualsValue

See Also

CompareValue

GreaterThanValue

LessThanValue

TValueRelationship

46.4.2 Types.GreaterThanValue Constant

GreaterThanValue indicates that one value is greater than an other in a comparison operation.

Pascal

```
GreaterThanValue = High(TValueRelationship);
```

C++

```
GreaterThanValue = High(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4707) value. If the first value is greater than the second, the value is GreaterThanValue

See Also

EqualsValue

CompareValue

LessThanValue

TValueRelationship

46.4.3 Types.LessThanValue Constant

LessThanValue indicates that one value is less than an other in a comparison operation.

Pascal

```
LessThanValue = Low(TValueRelationship);
```

C++

```
LessThanValue = Low(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4707) value. If the first value is less than the second, the value is LessThanValue

See Also

EqualsValue

CompareValue

GreaterThanValue

TValueRelationship

46.4.4 Types.RT_RCDATA Constant

Pascal

```
RT_RCDATA = PChar(10);
```

C++

```
RT_RCDATA = PChar(10);
```

File

Types

Description

This is constant Types.RT_RCDATA.

47 TypeTrans Namespace

This is namespace TypeTrans.

Topics

Name	Description
Classes (see page 4714)	The following table lists classes in this documentation.
Functions (see page 4718)	The following table lists functions in this documentation.
Variables (see page 4721)	The following table lists variables in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 4714)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 4714)	This is class TypeTrans.TTypeTranslator.

Functions

	Name	Description
	FloatToStrEx (see page 4718)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 4718)	This is function TypeTrans.GetEnumValueEx.
	GetEnumValueExW (see page 4719)	This is function TypeTrans.GetEnumValueExW.
	SetEnumPropEx (see page 4719)	This is function TypeTrans.SetEnumPropEx.
	SetEnumPropExW (see page 4719)	This is function TypeTrans.SetEnumPropExW.
	StrToFloatEx (see page 4720)	This is function TypeTrans.StrToFloatEx.

Variables

Name	Description
TypeTranslator (see page 4721)	This is variable TypeTrans.TypeTranslator.

47.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 4714)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 4714)	This is class TypeTrans.TTypeTranslator.

47.1.1 ETypeTransException Class

ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.

Class Hierarchy

Pascal

```
ETypeTransException = class(Exception);
```

C++

```
class ETypeTransException : public Exception;
```

File

TypeTrans

Description

ETypeTransException is thrown when an application attempts to convert a type in a SOAP packet to an Object Pascal type and fails for some reason.

See Also

TOPToSoapDomConvert

47.1.2 TTypeTranslator Class

Class Hierarchy

Pascal

```
TTypeTranslator = class;
```

C++

```
class TTypeTranslator;
```

File

TypeTrans

Description

This is class TypeTrans.TTypeTranslator.

47.1.2.1 TTypeTranslator Members

The following tables list the members exposed by TTypeTranslator.

Public Methods

	Name	Description
	Base64ToVar (see page 4716)	This is the overview for the Base64ToVar method overload.
	CastNativeToSoap (see page 4716)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 4717)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 4717)	This is the overview for the CastSoapToVariant method overload.
	Create (see page 4715)	This is Create, a member of class TTypeTranslator.
	Destroy (see page 4715)	This is Destroy, a member of class TTypeTranslator.

47.1.2.2 TTypeTranslator.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TTypeTranslator();
```

Description

This is Create, a member of class TTypeTranslator.

47.1.2.3 TTypeTranslator.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTypeTranslator();
```

Description

This is Destroy, a member of class TTypeTranslator.

47.1.2.4 TTypeTranslator Methods

The methods of the TTypeTranslator class are listed here.

Public Methods

	Name	Description
	Base64ToVar (see page 4716)	This is the overview for the Base64ToVar method overload.

	CastNativeToSoap (see page 4716)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 4717)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 4717)	This is the overview for the CastSoapToVariant method overload.

47.1.2.4.1 Base64ToVar Method

This is the overview for the Base64ToVar method overload.

Overload List

	Name	Description
	TTypeTranslator.Base64ToVar (Pointer, InvString) (see page 4716)	This is Base64ToVar, a member of class TTypeTranslator.
	TTypeTranslator.Base64ToVar (Variant, InvString) (see page 4716)	This is Base64ToVar, a member of class TTypeTranslator.

47.1.2.4.1.1 TTypeTranslator.Base64ToVar Method (Pointer, InvString)

Pascal

```
procedure Base64ToVar(NatData: Pointer; const SoapData: InvString); overload;
```

C++

```
__fastcall Base64ToVar(void * NatData, const InvString SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

47.1.2.4.1.2 TTypeTranslator.Base64ToVar Method (Variant, InvString)

Pascal

```
procedure Base64ToVar(var V: Variant; const SoapData: InvString); overload;
```

C++

```
__fastcall Base64ToVar(Variant V, const InvString SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

47.1.2.4.2 TTypeTranslator.CastNativeToSoap Method

Pascal

```
procedure CastNativeToSoap(Info: PTypeInfo; var SoapData: InvString; NatData: Pointer; var
IsNull: Boolean);
```

C++

```
__fastcall CastNativeToSoap(PTypeInfo Info, InvString SoapData, void * NatData, Boolean
IsNull);
```

Description

This is CastNativeToSoap, a member of class TTypeTranslator.

47.1.2.4.3 TTypeTranslator.CastSoapToNative Method

Pascal

```
function CastSoapToNative(Info: PTypeInfo; const SoapData: InvString; NatData: Pointer;
IsNull: Boolean): Boolean;
```

C++

```
__fastcall Boolean CastSoapToNative(PTypeInfo Info, const InvString SoapData, void * NatData,
Boolean IsNull);
```

Description

This is CastSoapToNative, a member of class TTypeTranslator.

47.1.2.4.4 CastSoapToVariant Method

This is the overview for the CastSoapToVariant method overload.

Overload List

	Name	Description
	TTypeTranslator.CastSoapToVariant (PTypeInfo, InvString) (see page 4717)	This is CastSoapToVariant, a member of class TTypeTranslator.
	TTypeTranslator.CastSoapToVariant (PTypeInfo, InvString, Pointer) (see page 4717)	This is CastSoapToVariant, a member of class TTypeTranslator.

47.1.2.4.4.1 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, InvString)

Pascal

```
function CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: InvString): Variant; overload;
```

C++

```
__fastcall Variant CastSoapToVariant(PTypeInfo SoapInfo, const InvString SoapData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

47.1.2.4.4.2 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, InvString, Pointer)

Pascal

```
procedure CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: InvString; NatData: Pointer);
overload;
```

C++

```
__fastcall CastSoapToVariant(PTypeInfo SoapInfo, const InvString SoapData, void * NatData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

47.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FloatToStrEx (see page 4718)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 4718)	This is function TypeTrans.GetEnumValueEx.
	GetEnumValueExW (see page 4719)	This is function TypeTrans.GetEnumValueExW.
	SetEnumPropEx (see page 4719)	This is function TypeTrans.SetEnumPropEx.
	SetEnumPropExW (see page 4719)	This is function TypeTrans.SetEnumPropExW.
	StrToFloatEx (see page 4720)	This is function TypeTrans.StrToFloatEx.

47.2.1 TypeTrans.FloatToStrEx Function

Pascal

```
function FloatToStrEx(Value: Extended): string;
```

C++

```
UnicodeString FloatToStrEx(Extended Value);
```

File

TypeTrans

Description

This is function TypeTrans.FloatToStrEx.

47.2.2 TypeTrans.GetEnumValueEx Function

Pascal

```
function GetEnumValueEx(TypInfo: PTypeInfo; const Name: string): Integer; deprecated;
```

C++

```
int GetEnumValueEx(PTypeInfo TypInfo, const UnicodeString Name);
```

File

TypeTrans

Description

This is function TypeTrans.GetEnumValueEx.

47.2.3 TypeTrans.GetEnumValueExW Function

Pascal

```
function GetEnumValueExW(TypInfo: PTypeInfo; const Name: InvString): Integer;
```

C++

```
int GetEnumValueExW(PTypeInfo TypInfo, const InvString Name);
```

File

TypeTrans

Description

This is function TypeTrans.GetEnumValueExW.

47.2.4 TypeTrans.SetEnumPropEx Function

Pascal

```
procedure SetEnumPropEx(Instance: TObject; PropInfo: PPropInfo; const Value: string);
deprecated;
```

C++

```
SetEnumPropEx(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypeTrans

Description

This is function TypeTrans.SetEnumPropEx.

47.2.5 TypeTrans.SetEnumPropExW Function

Pascal

```
procedure SetEnumPropExW(Instance: TObject; PropInfo: PPropInfo; const Value: InvString);
```

C++

```
SetEnumPropExW(TObject * Instance, PPropInfo PropInfo, const InvString Value);
```

File

TypeTrans

Description

This is function TypeTrans.SetEnumPropExW.

47.2.6 TypeTrans.StrToFloatEx Function

Pascal

```
function StrToFloatEx(const S: string): Extended;
```

C++

```
Extended StrToFloatEx(const UnicodeString S);
```

File

TypeTrans

Description

This is function TypeTrans.StrToFloatEx.

47.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
TypeTranslator (see page 4721)	This is variable TypeTrans.TypeTranslator.

47.3.1 TypeTrans.TypeTranslator Variable

Pascal

```
TypeTranslator: TTypeTranslator;
```

C++

```
TTypeTranslator TypeTranslator;
```

File

TypeTrans

Description

This is variable TypeTrans.TypeTranslator.

48 TypInfo Namespace

This is namespace TypInfo.

Topics

Name	Description
Classes (see page 4727)	The following table lists classes in this documentation.
Functions (see page 4735)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4779)	The following table lists structs, records, enums in this documentation.
Types (see page 4789)	The following table lists types in this documentation.
Variables (see page 4801)	The following table lists variables in this documentation.
Constants (see page 4802)	The following table lists constants in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 4727)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 4727)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 4728)	TPublishableVariantType is the base class for custom Variant (see page 3430) types that surface the published properties of an object that represents the Variant (see page 3430)'s data.

Constants

Name	Description
tkAny (see page 4802)	This is constant TypInfo.tkAny.
tkMethods (see page 4802)	This is constant TypInfo.tkMethods.
tkProperties (see page 4802)	This is constant TypInfo.tkProperties.

Functions

	Name	Description
	FreeAndNilProperties (see page 4737)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetAnsiStrProp (see page 4737)	This is function TypInfo.GetAnsiStrProp.
	GetAnsiStrProp (see page 4738)	This is function TypInfo.GetAnsiStrProp.
	GetDynArrayProp (see page 4738)	Returns the value of a component property that is a dynamic array type.
	GetDynArrayProp (see page 4738)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 4739)	Returns the name of a Delphi enumerated type constant given its value.

	GetEnumProp (see page 4742)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 4742)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 4744)	Returns the value of an enumerated type constant given its string representation.
	GetFloatProp (see page 4745)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 4745)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 4746)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 4746)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 4746)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 4746)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 4747)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 4747)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 4748)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 4748)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 4750)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 4750)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfos (see page 4751)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropName (see page 4753)	This is function TypInfo.GetPropName.
	GetPropValue (see page 4753)	Retrieves a component's property value as a variant.
	GetPropValue (see page 4753)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 4754)	This is function TypInfo.GetSetElementName.
	GetSetElementValue (see page 4754)	This is function TypInfo.GetSetElementValue.
	GetSetProp (see page 4755)	Returns the value of a component property that is a set type.
	GetSetProp (see page 4755)	Returns the value of a component property that is a set type.
	GetStrProp (see page 4756)	Returns the value of a component property that is a string type.
	GetStrProp (see page 4756)	Returns the value of a component property that is a string type.
	GetTypeData (see page 4756)	Returns a pointer to information about a type.
	GetTypeName (see page 4757)	This is function TypInfo.GetTypeName.
	GetUnicodeStrProp (see page 4757)	This is function TypInfo.GetUnicodeStrProp.
	GetUnicodeStrProp (see page 4758)	This is function TypInfo.GetUnicodeStrProp.
	GetVariantProp (see page 4758)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 4758)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 4759)	Returns the value of a component property that is a WideString (see page 3476) type.
	GetWideStrProp (see page 4759)	Returns the value of a component property that is a WideString (see page 3476) type.
	IsPublishedProp (see page 4759)	Determines if a specified component property is a published property.
	IsPublishedProp (see page 4759)	Determines if a specified component property is a published property.
	IsStoredProp (see page 4760)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 4760)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 4761)	Determines if a component property is of a particular data type.
	PropIsType (see page 4761)	Determines if a component property is of a particular data type.
	PropType (see page 4761)	Retrieves a component's property type.
	PropType (see page 4761)	Retrieves a component's property type.
	SamePropTypeName (see page 4762)	This is function TypInfo.SamePropTypeName.
	SetAnsiStrProp (see page 4762)	This is function TypInfo.SetAnsiStrProp.

	SetAnsiStrProp (see page 4763)	This is function TypInfo.SetAnsiStrProp.
	SetDynArrayProp (see page 4763)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 4763)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 4764)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 4764)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 4766)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 4766)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 4767)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 4767)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 4768)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 4768)	Sets the value of a component property that is an Interface type.
	SetMethodProp (see page 4769)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 4769)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 4770)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 4770)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 4770)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 4770)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 4771)	Sets the value of a component property using a variant value.
	SetPropValue (see page 4771)	Sets the value of a component property using a variant value.
	SetSetProp (see page 4772)	Sets the value of a component property that is a set type.
	SetSetProp (see page 4772)	Sets the value of a component property that is a set type.
	SetStrProp (see page 4773)	Sets the value of a component property that is a string type.
	SetStrProp (see page 4773)	Sets the value of a component property that is a string type.
	SetToString (see page 4774)	Returns one or more of the set values of a component's property.
	SetUnicodeStrProp (see page 4775)	This is function TypInfo.SetUnicodeStrProp.
	SetUnicodeStrProp (see page 4775)	This is function TypInfo.SetUnicodeStrProp.
	SetVariantProp (see page 4775)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 4775)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 4776)	Sets the value of a component property that is a WideString (see page 3476) type.
	SetWideStrProp (see page 4776)	Sets the value of a component property that is a WideString (see page 3476) type.
	SortPropList (see page 4777)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 4778)	Returns a bit mask of set values for a set type component property.

Structs, Records, Enums

	Name	Description
	TFloatType (see page 4779)	This is record TypInfo.TFloatType.
	TIntfFlag (see page 4780)	This is record TypInfo.TIntfFlag.
	TManagedField (see page 4780)	This is record TypInfo.TManagedField.
	TMethodKind (see page 4780)	Enumerates floating point number types in the TTypeData (see page 4785) record.
	TOrdType (see page 4782)	Enumerates floating point number types in the TTypeData (see page 4785) record.
	TParamFlag (see page 4783)	TParamFlag and TParamFlags (see page 4799) indicates the attributes of a method parameter.
	TPropData (see page 4783)	This is record TypInfo.TPropData.
	TPropInfo (see page 4784)	TPropInfo is a record of component property type information.
	TTypeData (see page 4785)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 4786)	Holds information for a data type as returned by TypeInfo.
	TTypeKind (see page 4787)	These types describe a type that is manipulated in the forms designer.

Types

Name	Description
PPropInfo (see page 4789)	PPropInfo points to a TPropInfo (see page 4784) record that holds component property information.

PPropList (see page 4790)	Defines a pointer to a TPropList (see page 4800) array.
PPTypeInfo (see page 4790)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 4791)	PTypeData points to a TTypeData (see page 4785) record that holds component property type information.
PTypeInfo (see page 4794)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 4797)	Defines a fixed length string for general use in the TTypeData (see page 4785) record.
TIntfFlags (see page 4797)	Enumerates interface flag types in the TTypeData (see page 4785) record.
TIntfFlagsBase (see page 4798)	Set of interface flag types in the TTypeData (see page 4785) record.
TParamFlags (see page 4799)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 4799)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 4799)	This is type TypInfo.TPropInfoProc.
TPropList (see page 4800)	Defines an array of PPropInfo (see page 4789) values.
TTypeKinds (see page 4800)	Defines a set of TTypeKind (see page 4787) values.

Variables

Name	Description
BooleanIdents (see page 4801)	An array of string representations of true and false.
DotSep (see page 4801)	This is variable TypInfo.DotSep.

48.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 4727)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 4727)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 4728)	TPublishableVariantType is the base class for custom Variant (see page 3430) types that surface the published properties of an object that represents the Variant (see page 3430)'s data.

48.1.1 EPropertyConvertError Class

EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.

Class Hierarchy

Pascal

```
EPropertyConvertError = class(Exception);
```

C++

```
class EPropertyConvertError : public Exception;
```

File

TypInfo

Description

EPropertyConvertError is raised when an application tries to set a property to a value of the wrong type or to read a property that has invalid type information.

See Also

EPropertyError (see page 4727)

TPropertyEditor

48.1.2 EPropertyError Class

EPropertyError is the exception class for unsuccessful attempts to set the value of a property.

Class Hierarchy

Pascal

```
EPropertyError = class(Exception);
```

C++

```
class EPropertyError : public Exception;
```

File

TypInfo

Description

EPropertyError is raised by the SetValue (or Edit) methods of property-editor objects when they cannot set a requested property value. This usually occurs because the requested value is out of range or of an invalid data type.

See Also

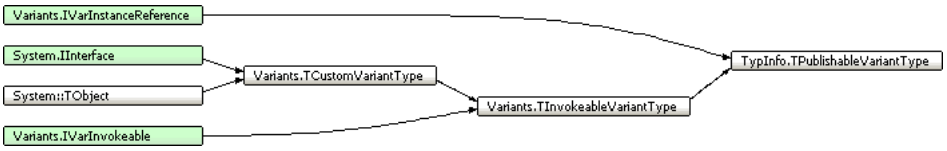
TPropertyEditor

TPropertyEditor

48.1.3 TPublishableVariantType Class

TPublishableVariantType is the base class for custom Variant (see page 3430) types that surface the published properties of an object that represents the Variant (see page 3430)'s data.

Class Hierarchy



Pascal

```
TPublishableVariantType = class(TInvokeableVariantType, IVarInstanceReference);
```

C++

```
class TPublishableVariantType : public TInvokeableVariantType, public IVarInstanceReference;
```

File

TypInfo

Description

TPublishableVariantType is the base class for new data types for Variants (see page 4997) that have properties and methods. TPublishableVariantType descendants do not represent the Variants (see page 4997) themselves. Rather, they act as utility classes that the custom Variants (see page 4997) call on to perform their operations. TPublishableVariantType adds to its ancestor's ability to implement properties and methods on Variants (see page 4997) of the custom type by automatically implementing all published properties of the object it uses to store its data.

Note: To create a Variant (see page 3430) type that descends from TPublishableVariantType,

Note: 1. Map the storage of the Variant (see page 3430)'s data on to the TVarData (see page 3813) record.

Note: 2. Write (see page 3789) a method that creates instances of your custom Variant (see page 3430). This fills in the Variant (see page 3430)'s data as defined in step 1.

Note: 3. Declare a class that descends from TPublishableVariantType.

Note: 4. Override the GetInstance (see page 4732) method to return the object that stores the custom Variant (see page 3430)'s data.

Note: 5. Implement all other required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction, DoProcedure, BinaryOp, UnaryOp, CompareOp or Compare, Cast, CastTo, Clear, Copy (see page 3648), IsClear, LeftPromotion, and RightPromotion).

Note: 6. In the initialization section of the unit that defines your TPublishableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant (see page 3430)-handling system so that the new Variant (see page 3430) type is enabled.

Note: 7. In the finalization section of the unit that defines your TPublishableVariantType descendant, free the instance of your class. This automatically unregisters the Variant (see page 3430) type.

Note: For an example, see the VarCmplx (see page 4925) (Windows) or FMTBcd (Linux) unit, which implements a custom Variant (see page 3430) type for complex (Windows) or binary-coded decimal (Linux) numbers.

Note: Do not create your own custom Variant (see page 3430) types in C++. They are only supported in Delphi.

See Also

IDH

TInvokeableVariantType (see page 5037)

Supporting Properties and Methods in Custom Variants

48.1.3.1 TPublishableVariantType Members

The following tables list the members exposed by TPublishableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 5017)	Increments the reference count for this interface.
	_Release (see page 5017)	Decrements the reference count for this interface.
V	DispInvoke (see page 5022)	Invokes a method call when the Variant (see page 3430) supports such operations.
V	DispInvoke (see page 5041)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5037) method.
V	FixupIdent (see page 5043)	Fixes up dispatch identifier.
A	GetInstance (see page 4732)	Returns the object instance that represents the data from a Variant (see page 3430) of the custom type.

	LeftPromotion (see page 5023)	Indicates whether the left argument to a binary operation should be coerced to a different type.
	OlePromotion (see page 5024)	Indicates the type to which a Variant (see page 3430) value must be cast to make it Automation-compatible.
	QueryInterface (see page 5025)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5025)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5026)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5026)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 5027)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5028)	Clears a TVarData (see page 3813) record by re-initializing it.
	SimplisticCopy (see page 5028)	Performs a simple block copy of one TVarData (see page 3813) record to another.
	VarDataCast (see page 5029)	Performs a cast from one Variant (see page 3430) type to another.
	VarDataCastTo (see page 5030)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5030)	Performs a cast to a Variant (see page 3430) type of varOleStr (see page 3936).
	VarDataClear (see page 5031)	Clears the data from a built-in Variant (see page 3430) type.
	VarDataCopy (see page 5031)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataCopyNoInd (see page 5032)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataFromOleStr (see page 5032)	Creates a TVarData (see page 3813) record to represent a Unicode string.
	VarDataFromStr (see page 5033)	Creates a TVarData (see page 3813) record to represent a string.
	VarDataInit (see page 5033)	Initializes a TVarData (see page 3813) record.
	VarDataIsArray (see page 5033)	Indicates whether a TVarData (see page 3813) record represents the data from a Variant (see page 3430) array.
	VarDataIsByRef (see page 5034)	Indicates whether a TVarData (see page 3813) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5034)	Indicates whether a TVarData (see page 3813) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5034)	Indicates whether a TVarData (see page 3813) record represents a floating-point value.
	VarDataIsNumeric (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a numeric value.
	VarDataIsOrdinal (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a ordinal value.
	VarDataIsStr (see page 5036)	Indicates whether a TVarData (see page 3813) record represents a string (normal or Unicode).
	VarDataToStr (see page 5036)	Returns a string that represents the value from a TVarData (see page 3813) record.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	BinaryOp (see page 5018)	Implements any binary operations of which the custom Variant (see page 3430) type is capable.
	Cast (see page 5018)	Converts the data from another Variant (see page 3430) type so that it represents this custom type's data format.
	CastTo (see page 5019)	This is CastTo, a member of class TCustomVariantType.
	CastToOle (see page 5019)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 5019)	Sets the data for this custom Variant (see page 3430) type to a blank or unassigned value.
	Compare (see page 5020)	Implements comparisons that involve the custom Variant (see page 3430) type.
	CompareOp (see page 5021)	Implements any comparison operations of which the custom Variant (see page 3430) type is capable.
	Copy (see page 5022)	Copies the data from a Variant (see page 3430) of this custom type.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 5014)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 5015)	Frees an instance of TCustomVariantType (see page 5011).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoFunction (see page 5047)	This is DoFunction, a member of class IVarInvokeable.
	DoFunction (see page 5041)	Implements all methods that return values.
	DoProcedure (see page 5047)	This is DoProcedure, a member of class IVarInvokeable.
	DoProcedure (see page 5042)	Implements all methods that do not return values.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetProperty (see page 4732)	Returns the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 5023)	Indicates whether the data from a custom Variant (see page 3430) represents a blank or unassigned value.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetProperty (see page 4733)	Sets the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).
	ToString (see page 3391)	Returns a string containing the class name.
	UnaryOp (see page 5029)	Implements any unary operations of which the custom Variant (see page 3430) type is capable.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	VarType (see page 5037)	Indicates the Variant (see page 3430) type code for the custom Variant (see page 3430) type.

48.1.3.2 TPublishableVariantType Methods

The methods of the TPublishableVariantType class are listed here.

Protected Methods

	Name	Description
	GetInstance (see page 4732)	Returns the object instance that represents the data from a Variant (see page 3430) of the custom type.

Public Methods

	Name	Description
	GetProperty (see page 4732)	Returns the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).
	SetProperty (see page 4733)	Sets the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).

48.1.3.2.1 TPublishableVariantType.GetInstance Method

Returns the object instance that represents the data from a Variant (see page 3430) of the custom type.

Pascal

```
function GetInstance(const V: TVarData): TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * GetInstance(const TVarData V) = 0;
```

Description

Override GetInstance to return the object that represents the data from a Variant (see page 3430) of the custom Type. TPublishableVariantType (see page 4728) implements all the published properties of that object so that they are published properties of the Variant (see page 3430).

V is the data from a Variant (see page 3430) of the custom type. This data must include a reference to an object that represents the Variant (see page 3430)'s data, and which GetInstance returns.

GetInstance is abstract or, in C++ terminology, pure virtual in TPublishableVariantType (see page 4728), which means it has no implementation. Descendant classes override this method to enable Variants (see page 4997) to publish the properties from the object that implements their data.

See Also

GetProperty (see page 4732)

SetProperty (see page 4733)

Supporting Properties and Methods in Custom Variants

48.1.3.2.2 TPublishableVariantType.GetProperty Method

Returns the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
override;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString
Name);
```

Description

GetProperty enables an application to read the properties of instances of the publishable Variant (see page 3430) type. It

implements all the properties that are published by the object that GetInstance (see page 4732) returns.

When the application contains a line such as

```
PropValue := MyVariant.SpecialProperty;
```

(Delphi) or calls the Exec method of the Variant (see page 3430) (C++), a call is automatically generated to GetProperty. If SpecialProperty is a published property on the object that implements the Variant (see page 3430)'s data, GetProperty fills in the Dest parameter with the property value.

Dest is a TVarData (see page 3813) record (Delphi) or structure (C++) that receives the property value.

V is the TVarData (see page 3813) record or structure from the Variant (see page 3430) instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, and raises an exception if it does not implement the specified property.

See Also

DoFunction (see page 5041)

DoProcedure (see page 5042)

GetInstance (see page 4732)

Exec

48.1.3.2.3 TPublishableVariantType.SetProperty Method

Sets the value of a published property on the object that GetInstance (see page 4732) returns, so that it is available in custom Variants (see page 4997).

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
override;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const
TVarData Value);
```

Description

SetProperty enables an application to set the properties of instances of the publishable Variant (see page 3430) type. It implements all the settable properties that are published by the object that GetInstance (see page 4732) returns.

When the application contains a line such as

```
MyVariant.SpecialProperty := NewValue;
```

(Delphi) or calls the Exec method of the Variant (see page 3430) (C++), a call is automatically generated to SetProperty. If SpecialProperty is a published settable property on the object that implements the Variant (see page 3430)'s data, SetProperty sets the property on the object included in V so that the Variant (see page 3430)'s value is changed accordingly.

V is the TVarData (see page 3813) record (Delphi) or structure (C++) from the Variant (see page 3430) instance whose property is being set. It includes the object that the GetInstance (see page 4732) method returns.

Name is the name of the property.

Value is a TVarData (see page 3813) record or structure that contains the new property value.

GetProperty (see page 4732) returns true if it sets a property value, and raises an exception of Name does not specify a published settable property on the object that GetInstance (see page 4732) returns.

See Also

DoFunction (see page 5041)

DoProcedure (see page 5042)

GetInstance (see page 4732)

Exec

48.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeAndNilProperties (see page 4737)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetAnsiStrProp (see page 4737)	This is function TypInfo.GetAnsiStrProp.
	GetAnsiStrProp (see page 4738)	This is function TypInfo.GetAnsiStrProp.
	GetDynArrayProp (see page 4738)	Returns the value of a component property that is a dynamic array type.
	GetDynArrayProp (see page 4738)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 4739)	Returns the name of a Delphi enumerated type constant given its value.
	GetEnumProp (see page 4742)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 4742)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 4744)	Returns the value of an enumerated type constant given its string representation.
	GetFloatProp (see page 4745)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 4745)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 4746)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 4746)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 4746)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 4746)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 4747)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 4747)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 4748)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 4748)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4749)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 4750)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 4750)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfo (see page 4751)	Retrieves information about a component's property.
	GetPropInfos (see page 4751)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropList (see page 4752)	Retrieves a list of component properties.
	GetPropName (see page 4753)	This is function TypInfo.GetPropName.
	GetPropValue (see page 4753)	Retrieves a component's property value as a variant.
	GetPropValue (see page 4753)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 4754)	This is function TypInfo.GetSetElementName.
	GetSetElementValue (see page 4754)	This is function TypInfo.GetSetElementValue.
	GetSetProp (see page 4755)	Returns the value of a component property that is a set type.
	GetSetProp (see page 4755)	Returns the value of a component property that is a set type.
	GetStrProp (see page 4756)	Returns the value of a component property that is a string type.
	GetStrProp (see page 4756)	Returns the value of a component property that is a string type.
	GetTypeData (see page 4756)	Returns a pointer to information about a type.
	GetTypeName (see page 4757)	This is function TypInfo.GetTypeName.
	GetUnicodeStrProp (see page 4757)	This is function TypInfo.GetUnicodeStrProp.

	GetUnicodeStrProp (see page 4758)	This is function TypInfo.GetUnicodeStrProp.
	GetVariantProp (see page 4758)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 4758)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 4759)	Returns the value of a component property that is a WideString (see page 3476) type.
	GetWideStrProp (see page 4759)	Returns the value of a component property that is a WideString (see page 3476) type.
	IsPublishedProp (see page 4759)	Determines if a specified component property is a published property.
	IsPublishedProp (see page 4759)	Determines if a specified component property is a published property.
	IsStoredProp (see page 4760)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 4760)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 4761)	Determines if a component property is of a particular data type.
	PropIsType (see page 4761)	Determines if a component property is of a particular data type.
	PropType (see page 4761)	Retrieves a component's property type.
	PropType (see page 4761)	Retrieves a component's property type.
	SamePropTypeName (see page 4762)	This is function TypInfo.SamePropTypeName.
	SetAnsiStrProp (see page 4762)	This is function TypInfo.SetAnsiStrProp.
	SetAnsiStrProp (see page 4763)	This is function TypInfo.SetAnsiStrProp.
	SetDynArrayProp (see page 4763)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 4763)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 4764)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 4764)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 4766)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 4766)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 4767)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 4767)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 4768)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 4768)	Sets the value of a component property that is an Interface type.
	SetMethodProp (see page 4769)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 4769)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 4770)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 4770)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 4770)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 4770)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 4771)	Sets the value of a component property using a variant value.
	SetPropValue (see page 4771)	Sets the value of a component property using a variant value.
	SetSetProp (see page 4772)	Sets the value of a component property that is a set type.
	SetSetProp (see page 4772)	Sets the value of a component property that is a set type.
	SetStrProp (see page 4773)	Sets the value of a component property that is a string type.
	SetStrProp (see page 4773)	Sets the value of a component property that is a string type.
	SetToString (see page 4774)	Returns one or more of the set values of a component's property.
	SetUnicodeStrProp (see page 4775)	This is function TypInfo.SetUnicodeStrProp.
	SetUnicodeStrProp (see page 4775)	This is function TypInfo.SetUnicodeStrProp.
	SetVariantProp (see page 4775)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 4775)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 4776)	Sets the value of a component property that is a WideString (see page 3476) type.
	SetWideStrProp (see page 4776)	Sets the value of a component property that is a WideString (see page 3476) type.
	SortPropList (see page 4777)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 4778)	Returns a bit mask of set values for a set type component property.

48.2.1 TypInfo.FreeAndNilProperties Function

Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNilProperties(AObject: TObject);
```

C++

```
FreeAndNilProperties(TObject * AObject);
```

File

TypInfo

Description

Call FreeAndNilProperties from a destructor to free all internally created objects and set the corresponding property values to nil (Delphi) or NULL (C++). FreeAndNilProperties searches the properties of AObject, identifying any that are objects. It frees those objects, and sets the corresponding property to nil (Delphi) or NULL (C++).

. By default, all persistent objects (TPersistent and descendants) are compiled with RTTI. Other TObject (see page 3364) descendants can be compiled with RTTI by using the $M+ compiler switch in Delphi or the __declspec(delphirtti) directive in C++.

Warning: Calling FreeAndNilProperties frees all object properties, which may lead to unintended results. FreeAndNilProperties can't distinguish between objects that were created by AObject (and typically stored in a class member) from objects that are created independently and which should not be freed with AObject. To avoid unintentionally freeing independently-created objects, set properties that refer to external objects to nil (Delphi) or NULL (C++) before calling FreeAndNilProperties.

See Also

FreeAndNil

48.2.2 TypInfo.GetAnsiStrProp Function

Pascal

```
function GetAnsiStrProp(Instance: TObject; PropInfo: PPropInfo): AnsiString; overload;
```

C++

```
AnsiString GetAnsiStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

This is function TypInfo.GetAnsiStrProp.

48.2.3 TypInfo.GetAnsiStrProp Function

Pascal

```
function GetAnsiStrProp(Instance: TObject; const PropName: string): AnsiString; overload;
```

C++

```
AnsiString GetAnsiStrProp(TObject * Instance, const UnicodeString PropName);
```

File

TypInfo

Description

This is function TypInfo.GetAnsiStrProp.

48.2.4 TypInfo.GetDynArrayProp Function

Returns the value of a component property that is a dynamic array type.

Pascal

```
function GetDynArrayProp(Instance: TObject; const PropName: string): Pointer; overload;
function GetDynArrayProp(Instance: TObject; PropInfo: PPropInfo): Pointer; overload;
```

C++

```
void * GetDynArrayProp(TObject * Instance, const UnicodeString PropName);
void * GetDynArrayProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetDynArrayProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a pointer to the array.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetDynArrayProp (see page 4763)

48.2.5 TypInfo.GetEnumName Function

Returns the name of a Delphi enumerated type constant given its value.

Pascal

```
function GetEnumName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
UnicodeString GetEnumName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

Use GetEnumName to convert a Delphi enumerated value into the symbolic name that represents it in code.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

Value is the enumerated value, represented as an integer.

GetEnumName returns the string that represents the specified value.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
```

```
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
```

```
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;

    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;

  end;
  memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
end;


procedure TForm2.FormCreate(Sender: TObject);
var
  GenericString : TGenericClass<UnicodeString>;
  GenericByte : TGenericClass<Byte>;
  GenericDouble : TGenericClass<Double>;
//  GenericStringArray : TGenericClass<array of string>;
begin
  { Prints type info for string type}

  GenericString := TGenericClass<UnicodeString>.Create;
  GenericString.PrintTypeInfo(Memo1);

  { Prints type info for Byte type}
  GenericByte := TGenericClass<Byte>.Create;
  GenericByte.PrintTypeInfo(Memo1);

  { Prints type info for Double type}
  GenericDouble := TGenericClass<Double>.Create;
  GenericDouble.PrintTypeInfo(Memo1);
end;

{
Expected results are:

Type name: string
Type kind: tkUString
Size of type: 4

Type name: Byte
Type kind: tkInteger
Integer kind: otUByte
Size of type: 1

Type name: Double
Type kind: tkFloat
Float kind: ftDouble
Size of type: 8

Type name: .1
Type kind: tkDynArray
Array element type name: string
Array element type size: 4
```

```
Size of type: 4
}
```

See Also

IntToIdent

TypeInfo

GetEnumValue (see page 4744)

48.2.6 TypInfo.GetEnumProp Function

Returns the value of a component property that is an enumerated type.

Pascal

```
function GetEnumProp(Instance: TObject; const PropName: string): string; overload;
function GetEnumProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
UnicodeString GetEnumProp(TObject * Instance, const UnicodeString PropName);
UnicodeString GetEnumProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetEnumProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

C++ Examples:

```
/*
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
*/
void __fastcall TForm2::Button1Click(TObject *Sender)
{
    //getting the current color of the workspace
    String currentPropColor = GetEnumProp(this,"Color");
    int currentColorInt = StrToInt(currentPropColor);
```

```
    //getting the first button align enum and if different,
    //setting it to alLeft
    String currentAlignProp = GetEnumProp(Button1, "Align");
    if (currentAlignProp != "alLeft")
    {
         SetEnumProp(Button1, "Align", "alLeft");
     }

    //checking if the form background color was set.
    if(currentColorInt < 0)
    {
        currentColorInt = GetSysColor(COLOR_APPWORKSPACE);
    }

    //setting the form background color as the negative value
    //of the current background color
    SetEnumProp(this, "Color",
                IntToStr((int)(clWhite - currentColorInt)));
}

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    int p[5] = {clYellow, clGreen, clRed, clBlue, clBlack};
    SetEnumProp(this, "Color", IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    String currentAlignProp = GetEnumProp(Button2, "Align");
    if (currentAlignProp != "alRight")
    {
         SetEnumProp(Button2, "Align", "alRight");
     }
}
```

Delphi Examples:

```
{
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
}
procedure TForm3.Button1Click(Sender: TObject);
var
  currentPropColor : String;
  currentAlignProp : String;
  currentColorInt : Integer;
begin
  //getting the current color of the workspace
  currentPropColor := GetEnumProp(Self,'Color');
  currentColorInt := StrToInt(currentPropColor);

  //getting the first button align enum and if different,
  //setting it to alLeft
  currentAlignProp := GetEnumProp(Button1, 'Align');
  if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alLeft) then
     SetEnumProp(Button1, 'Align', 'alLeft');

  //checking if the form background color was set.
    if currentColorInt < 0 then
    begin
        currentColorInt := GetSysColor(COLOR_APPWORKSPACE);
    end;

  //setting the form background color as the negative value
  // of the current background color
  currentColorInt := clWhite - currentColorInt;
```

```
    SetEnumProp(Self, 'Color', IntToStr(currentColorInt));
  end;

  procedure TForm3.Button2Click(Sender: TObject);
  var
    p : array [0..4] of Integer;
    currentAlignProp : String;
  begin
    p[0] := clYellow;
    p[1] := clRed;
    p[2] := clBlue;
    p[3] := clBlack;
    p[4] := clGreen;

      SetEnumProp(Self, 'Color', IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    currentAlignProp := GetEnumProp(Button2, 'Align');
    if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alRight) then
       SetEnumProp(Button2, 'Align', 'alRight');

  end;
```

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetEnumProp (see page 4764)

48.2.7 TypInfo.GetEnumValue Function

Returns the value of an enumerated type constant given its string representation.

Pascal

```
function GetEnumValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetEnumValue(PTypeInfo TypeInfo, const UnicodeString Name);
```

File

TypInfo

Description

Use GetEnumValue to convert the string that represents an enumerated value into the corresponding integer that is its implemented value.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by calling the TypeInfo function in Delphi or using the __delphirtti operator in C++.

Name is the string that is used in code to represent the enumerated value.

GetEnumValue returns the corresponding integer value.

See Also

IdentToInt

TypeInfo

GetEnumName (see page 4739)

48.2.8 TypInfo.GetFloatProp Function

Returns the value of a component property that is a float type.

Pascal

```
function GetFloatProp(Instance: TObject; const PropName: string): Extended; overload;
function GetFloatProp(Instance: TObject; PropInfo: PPropInfo): Extended; overload;
```

C++

```
Extended GetFloatProp(TObject * Instance, const UnicodeString PropName);
Extended GetFloatProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetFloatProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a floating point number type. 0.0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a float value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetFloatProp (see page 4766)

48.2.9 TypInfo.GetInt64Prop Function

Returns the value of a component property that is an Int64 type.

Pascal

```
function GetInt64Prop(Instance: TObject; const PropName: string): Int64; overload;
function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo): Int64; overload;
```

C++

```
Int64 GetInt64Prop(TObject * Instance, const UnicodeString PropName);
Int64 GetInt64Prop(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInt64Prop utilises Delphi's RTTI (Run Time see page 4338) Type Information) to return the current value of a component's property where that property is an Int64 type. 0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an Int64 value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetInt64Prop (see page 4767)

48.2.10 TypInfo.GetInterfaceProp Function

Returns the value of a component property that is an interface type.

Pascal

```
function GetInterfaceProp(Instance: TObject; const PropName: string): IInterface; overload;
function GetInterfaceProp(Instance: TObject; PropInfo: PPropInfo): IInterface; overload;
```

C++

```
IInterface GetInterfaceProp(TObject * Instance, const UnicodeString PropName);
IInterface GetInterfaceProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInterfaceProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is an interface type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an interface value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetInterfaceProp (see page 4768)

48.2.11 TypInfo.GetMethodProp Function

Returns the value of a component property that is a method type.

Pascal

```
function GetMethodProp(Instance: TObject; const PropName: string): TMethod; overload;
function GetMethodProp(Instance: TObject; PropInfo: PPropInfo): TMethod; overload;
```

C++

```
TMethod GetMethodProp(TObject * Instance, const UnicodeString PropName);
TMethod GetMethodProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetMethodProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a method type : an event routine.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a method value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetMethodProp (see page 4769)

48.2.12 TypInfo.GetObjectProp Function

Returns the value of a component property that is an object type.

Pascal

```
function GetObjectProp(Instance: TObject; const PropName: string; MinClass: TClass = nil):
TObject; overload;
function GetObjectProp(Instance: TObject; PropInfo: PPropInfo; MinClass: TClass = nil):
TObject; overload;
```

C++

```
TObject * GetObjectProp(TObject * Instance, const UnicodeString PropName, TClass MinClass =
nil);
TObject * GetObjectProp(TObject * Instance, PPropInfo PropInfo, TClass MinClass = nil);
```

File

TypInfo

Description

GetObjectProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is an object type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an object, or nil if the property is not an object type.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetObjectProp (see page 4770)

48.2.13 TypInfo.GetObjectPropClass Function

Returns the class type of a component property that is an object type.

Pascal

```
function GetObjectPropClass(Instance: TObject; const PropName: string): TClass; overload;
function GetObjectPropClass(Instance: TObject; PropInfo: PPropInfo): TClass; overload;
function GetObjectPropClass(PropInfo: PPropInfo): TClass; overload;
```

C++

```
TClass GetObjectPropClass(TObject * Instance, const UnicodeString PropName);
TClass GetObjectPropClass(TObject * Instance, PPropInfo PropInfo);
TClass GetObjectPropClass(PPropInfo PropInfo);
```

File

TypInfo

Description

GetObjectPropClass utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the class type of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the class type of the property object value is returned.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

48.2.14 TypInfo.GetOrdProp Function

Returns the value of a component property that is an ordinal type.

Pascal

```
function GetOrdProp(Instance: TObject; const PropName: string): Longint; overload;
function GetOrdProp(Instance: TObject; PropInfo: PPropInfo): Longint; overload;
```

C++

```
Longint GetOrdProp(TObject * Instance, const UnicodeString PropName);
Longint GetOrdProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetOrdProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an LongInt value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetOrdProp (see page 4770)

48.2.15 TypInfo.GetPropInfo Function

Retrieves information about a component's property.

Pascal

```
function GetPropInfo(Instance: TObject; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(AClass: TClass; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string): PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string; AKinds: TTypeKinds):
PPropInfo; overload;
```

C++

```
PPropInfo GetPropInfo(TObject * Instance, const UnicodeString PropName, TTypeKinds AKinds =
[]);
PPropInfo GetPropInfo(TClass AClass, const UnicodeString PropName, TTypeKinds AKinds = []);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const UnicodeString PropName);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const UnicodeString PropName, TTypeKinds AKinds);
```

File

TypInfo

Description

GetPropInfo utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to retrieve a pointer to a property information record for a component.

In the first form of this function, **TypeInfo** specifies the component type whose property you want to retrieve. **PropName** specifies the property name as a string. You may optionally specify a **AKinds** set of TTypeKind (see page 4787) property value types that the property must belong to.

The second form of the function is similar to the first, with **AClass** replacing **TypeInfo** as the means of specifying the component type. **AKinds** is now a required value.

The final form of the function is like the second form, with **Instance** specifying a component instance.

In all forms of the function, the returned value is a pointer to a TPropInfo (see page 4784) value. This provides rich data about the property being investigated at run time. This enables more elegant component processing than can be achieved with hard coded component names/properties.

Note: If the specified property does not exist, a nil pointer is returned.

See Also

PTypeInfo (see page 4794)

PPropInfo (see page 4789)

TTypeKinds (see page 4800)

48.2.16 TypInfo.GetPropInfos Function

Retrieves a list of component properties.

Pascal

```
procedure GetPropInfos(TypeInfo: PTypeInfo; PropList: PPropList);
```

C++

```
GetPropInfos(PTypeInfo TypeInfo, PPropList PropList);
```

File

TypInfo

Description

GetPropInfos utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to retrieve a list of property information records for a component or component type.

The **TypeInfo** defines the component type and **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 4800) variable

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

Note: If you want the count of properties returned in PropList

, use GetPropList (see page 4752) instead.

See Also

GetPropList (see page 4752)

PTypeInfo (see page 4794)

PPropList (see page 4790)

TPropList (see page 4800)

TTypeKinds (see page 4800)

48.2.17 TypInfo.GetPropList Function

Retrieves a list of component properties.

Pascal

```
function GetPropList(TypeInfo: PTypeInfo; TypeKinds: TTypeKinds; PropList: PPropList;
SortList: Boolean = True): Integer; overload;
function GetPropList(TypeInfo: PTypeInfo; out PropList: PPropList): Integer; overload;
function GetPropList(AObject: TObject; out PropList: PPropList): Integer; overload;
```

C++

```
int GetPropList(PTypeInfo TypeInfo, TTypeKinds TypeKinds, PPropList PropList, Boolean SortList
= True);
int GetPropList(PTypeInfo TypeInfo, PPropList PropList);
int GetPropList(TObject * AObject, PPropList PropList);
```

File

TypInfo

Description

GetPropList utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to retrieve a list of property information records for a component or component type.

In one form of this function, **TypeInfo** defines the component type, **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 4800) variable, and **TypeKinds** defines the type of properties that will be returned. Use tkAny (see page 4802) to return all types. The list may optionally be sorted by setting **SortList** true.

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

In the other form of this function, you specify a component instance rather than component type. The **TypeKinds** and **SortList** parameters can no longer be specified.

In both forms of the function, the returned value is the number of properties in **PropList**.

Note: If you do not need the count of properties returned, use GetPropInfos (see page 4751).

See Also

GetPropInfos (see page 4751)

PTypeInfo (see page 4794)

PPropList (see page 4790)

TPropList (see page 4800)

TTypeKinds (see page 4800)

48.2.18 TypInfo.GetPropName Function

Pascal

```
function GetPropName(PropInfo: PPropInfo): string;
```

C++

```
UnicodeString GetPropName(PPropInfo PropInfo);
```

File

TypInfo

Description

This is function TypInfo.GetPropName.

48.2.19 TypInfo.GetPropValue Function

Retrieves a component's property value as a variant.

Pascal

```
function GetPropValue(Instance: TObject; const PropName: string; PreferStrings: Boolean =
True): Variant; overload;
function GetPropValue(Instance: TObject; PropInfo: PPropInfo; PreferStrings: Boolean = True):
Variant; overload;
```

C++

```
Variant GetPropValue(TObject * Instance, const UnicodeString PropName, Boolean PreferStrings =
True);
Variant GetPropValue(TObject * Instance, PPropInfo PropInfo, Boolean PreferStrings = True);
```

File

TypInfo

Description

GetPropValue utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to retrieve a a component property as a variant.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the returned value the property value as a variant.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

PTypeInfo (see page 4794)

PPropInfo (see page 4789)

TTypeKinds (see page 4800)

SetPropValue (see page 4771)

48.2.20 TypInfo.GetSetElementName Function

Pascal

```
function GetSetElementName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
UnicodeString GetSetElementName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

This is function TypInfo.GetSetElementName.

48.2.21 TypInfo.GetSetElementValue Function

Pascal

```
function GetSetElementValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetSetElementValue(PTypeInfo TypeInfo, const UnicodeString Name);
```

File

TypInfo

Description

This is function TypInfo.GetSetElementValue.

48.2.22 TypInfo.GetSetProp Function

Returns the value of a component property that is a set type.

Pascal

```
function GetSetProp(Instance: TObject; const PropName: string; Brackets: Boolean = False):
string; overload;
function GetSetProp(Instance: TObject; PropInfo: PPropInfo; Brackets: Boolean = False):
string; overload;
```

C++

```
UnicodeString GetSetProp(TObject * Instance, const UnicodeString PropName, Boolean Brackets =
False);
UnicodeString GetSetProp(TObject * Instance, PPropInfo PropInfo, Boolean Brackets = False);
```

File

TypInfo

Description

GetSetProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a set of values type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the list if set values is returned as a comma separated string. You may optionally set **Brackets** true so as to enclose this within square brackets, as shown in the IDE for the property.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetSetProp (see page 4772)

48.2.23 TypInfo.GetStrProp Function

Returns the value of a component property that is a string type.

Pascal

```
function GetStrProp(Instance: TObject; const PropName: string): string; overload;
function GetStrProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
UnicodeString GetStrProp(TObject * Instance, const UnicodeString PropName);
UnicodeString GetStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetStrProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a String, LongString or WideString (see page 3476) type. An empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetStrProp (see page 4773)

48.2.24 TypInfo.GetTypeData Function

Returns a pointer to information about a type.

Pascal

```
function GetTypeData(TypeInfo: PTypeInfo): PTypeData;
```

C++

```
PTypeData GetTypeData(PTypeInfo TypeInfo);
```

File

TypInfo

Description

GetTypeData utilises Delphi's RTTI (Run Time see page 4338) Type Information) to return a pointer to TTypeData (see page 4785) record.

The TTypeData (see page 4785) record is a variant type, but does not contain the variant as a tag. To identify which variant applies to a particular type, you should retrieve the TTypeKind (see page 4787) value using the TypeInfo routine.

The TTypeData (see page 4785) record holds meta data about a property. For example, the name of the unit a class type is stored in.

See Also

PTypeData (see page 4791)

TTypeData (see page 4785)

TTypeKind (see page 4787)

48.2.25 TypInfo.GetTypeName Function

Pascal

```
function GetTypeName(TypeInfo: PTypeInfo): string;
```

C++

```
UnicodeString GetTypeName(PTypeInfo TypeInfo);
```

File

TypInfo

Description

This is function TypInfo.GetTypeName.

48.2.26 TypInfo.GetUnicodeStrProp Function

Pascal

```
function GetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo): UnicodeString; overload;
```

C++

```
UnicodeString GetUnicodeStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

This is function TypInfo.GetUnicodeStrProp.

48.2.27 TypInfo.GetUnicodeStrProp Function

Pascal

```
function GetUnicodeStrProp(Instance: TObject; const PropName: string): UnicodeString; overload;
```

C++

```
UnicodeString GetUnicodeStrProp(TObject * Instance, const UnicodeString PropName);
```

File

TypInfo

Description

This is function TypInfo.GetUnicodeStrProp.

48.2.28 TypInfo.GetVariantProp Function

Returns the value of a component property that is a variant type.

Pascal

```
function GetVariantProp(Instance: TObject; const PropName: string): Variant; overload;
function GetVariantProp(Instance: TObject; PropInfo: PPropInfo): Variant; overload;
```

C++

```
Variant GetVariantProp(TObject * Instance, const UnicodeString PropName);
Variant GetVariantProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetVariantProp utilises Delphi's RTTI (Run Time see page 4338) Type Information) to return the current value of a component's property that is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a variant.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown. If the property is not a variant type, an EVariantBadVarTypeError (see page 5004) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetVariantProp (see page 4775)

48.2.29 TypInfo.GetWideStrProp Function

Returns the value of a component property that is a WideString (see page 3476) type.

Pascal

```
function GetWideStrProp(Instance: TObject; const PropName: string): WideString; overload;
function GetWideStrProp(Instance: TObject; PropInfo: PPropInfo): WideString; overload;
```

C++

```
BSTR GetWideStrProp(TObject * Instance, const UnicodeString PropName);
BSTR GetWideStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetWideStrProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the current value of a component's property where that property is a WideString (see page 3476) type. A string or LongString type is returned via a call to the GetWideStrProp routine. Otherwise, for other types, an empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a WideString (see page 3476).

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4753) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3430) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropList (see page 4752)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

SetWideStrProp (see page 4776)

48.2.30 TypInfo.IsPublishedProp Function

Determines if a specified component property is a published property.

Pascal

```
function IsPublishedProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsPublishedProp(AClass: TClass; const PropName: string): Boolean; overload;
```

C++

```
Boolean IsPublishedProp(TObject * Instance, const UnicodeString PropName);
Boolean IsPublishedProp(TClass AClass, const UnicodeString PropName);
```

File

TypInfo

Description

IsPublishedProp returns true if the specified property is published. Delphi's RTTI (Run Time (see page 4338) Type Information) facility requires that properties are published.

In one form of this function, the component is defined by the **TClass (see page 3856)** parameter.

In the other form of this function, an **Instance** of the component is used.

In both forms of the function, **PropName** specifies the property as a string.

See Also

GetPropInfo (see page 4751)

GetPropValue (see page 4753)

48.2.31 TypInfo.IsStoredProp Function

Determines if a specified component property has the stored attribute.

Pascal

```
function IsStoredProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsStoredProp(Instance: TObject; PropInfo: PPropInfo): Boolean; overload;
```

C++

```
Boolean IsStoredProp(TObject * Instance, const UnicodeString PropName);
Boolean IsStoredProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

IsStoredProp returns true if the specified property is saved in the form file. The stored attribute is set true by default : only those that have the keyword stored defined as false do not.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the value of the stored attribute is returned.

See Also

GetPropInfo (see page 4751)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

48.2.32 TypInfo.PropIsType Function

Determines if a component property is of a particular data type.

Pascal

```
function PropIsType(Instance: TObject; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
function PropIsType(AClass: TClass; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
```

C++

```
Boolean PropIsType(TObject * Instance, const UnicodeString PropName, TTypeKind TypeKind);
Boolean PropIsType(TClass AClass, const UnicodeString PropName, TTypeKind TypeKind);
```

File

TypInfo

Description

PropIsType returns true if the property is of the specified data type.

In the first form of this function, **AClass** specifies the component class whose property you want to look at.

The second form of the function a class **Instance** replaces **AClass**.

In both forms of the function, **PropName** specifies the property name as a string. If the property has the **TypeKind** data type, then true is returned. Otherwise false is returned.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropInfo (see page 4751)

GetPropValue (see page 4753)

PPropInfo (see page 4789)

TTypeKinds (see page 4800)

48.2.33 TypInfo.PropType Function

Retrieves a component's property type.

Pascal

```
function PropType(Instance: TObject; const PropName: string): TTypeKind; overload;
function PropType(AClass: TClass; const PropName: string): TTypeKind; overload;
```

C++

```
TTypeKind PropType(TObject * Instance, const UnicodeString PropName);
TTypeKind PropType(TClass AClass, const UnicodeString PropName);
```

File

TypInfo

Description

PropType utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return the type of a component property in a TTypeKind (see page 4787) value.

In the first form of this function, **AClass** specifies the component class whose property you want to retrieve. **PropName** specifies the property name as a string.

The second form of the function is similar to the first, with **Instance** replacing **AClass** as the means of specifying the component.

In both forms of the function, the returned value is a TTypeKind (see page 4787) value.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

PTypeInfo (see page 4794)

PPropInfo (see page 4789)

TTypeKind (see page 4787)

48.2.34 TypInfo.SamePropTypeName Function

Pascal

```
function SamePropTypeName(const Name1: ShortString; const Name2: ShortString): Boolean;
```

C++

```
Boolean SamePropTypeName(const ShortString Name1, const ShortString Name2);
```

File

TypInfo

Description

This is function TypInfo.SamePropTypeName.

48.2.35 TypInfo.SetAnsiStrProp Function

Pascal

```
procedure SetAnsiStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: AnsiString);
overload;
```

C++

```
SetAnsiStrProp(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

This is function TypInfo.SetAnsiStrProp.

48.2.36 TypInfo.SetAnsiStrProp Function

Pascal

```
procedure SetAnsiStrProp(Instance: TObject; const PropName: string; const Value: AnsiString);
overload;
```

C++

```
SetAnsiStrProp(TObject * Instance, const UnicodeString PropName, const AnsiString Value);
```

File

TypInfo

Description

This is function TypInfo.SetAnsiStrProp.

48.2.37 TypInfo.SetDynArrayProp Function

Sets the value of a component property that is a dynamic array type.

Pascal

```
procedure SetDynArrayProp(Instance: TObject; const PropName: string; const Value: Pointer);
overload;
procedure SetDynArrayProp(Instance: TObject; PropInfo: PPropInfo; const Value: Pointer);
overload;
```

C++

```
SetDynArrayProp(TObject * Instance, const UnicodeString PropName, const void * Value);
SetDynArrayProp(TObject * Instance, PPropInfo PropInfo, const void * Value);
```

File

TypInfo

Description

SetDynArrayProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetEnumProp (see page 4742)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.38 TypInfo.SetEnumProp Function

Sets the value of a component property that is an enumerated type.

Pascal

```
procedure SetEnumProp(Instance: TObject; const PropName: string; const Value: string);
overload;
procedure SetEnumProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetEnumProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetEnumProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetEnumProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

C++ Examples:

```
/*
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
```

```
*/
void __fastcall TForm2::Button1Click(TObject *Sender)
{
    //getting the current color of the workspace
    String currentPropColor = GetEnumProp(this,"Color");
    int currentColorInt = StrToInt(currentPropColor);

    //getting the first button align enum and if different,
    //setting it to alLeft
    String currentAlignProp = GetEnumProp(Button1, "Align");
    if (currentAlignProp != "alLeft")
    {
         SetEnumProp(Button1, "Align", "alLeft");
     }

    //checking if the form background color was set.
    if(currentColorInt < 0)
    {
        currentColorInt = GetSysColor(COLOR_APPWORKSPACE);
    }

    //setting the form background color as the negative value
    //of the current background color
    SetEnumProp(this, "Color",
                IntToStr((int)(clWhite - currentColorInt)));
}

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    int p[5] = {clYellow, clGreen, clRed, clBlue, clBlack};
    SetEnumProp(this, "Color", IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    String currentAlignProp = GetEnumProp(Button2, "Align");
    if (currentAlignProp != "alRight")
    {
         SetEnumProp(Button2, "Align", "alRight");
     }
}
```

Delphi Examples:

```
{
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
}
procedure TForm3.Button1Click(Sender: TObject);
var
  currentPropColor : String;
  currentAlignProp : String;
  currentColorInt : Integer;
begin
  //getting the current color of the workspace
  currentPropColor := GetEnumProp(Self,'Color');
  currentColorInt := StrToInt(currentPropColor);

  //getting the first button align enum and if different,
  //setting it to alLeft
  currentAlignProp := GetEnumProp(Button1, 'Align');
  if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alLeft) then
     SetEnumProp(Button1, 'Align', 'alLeft');

  //checking if the form background color was set.
    if currentColorInt < 0 then
    begin
```

```
          currentColorInt := GetSysColor(COLOR_APPWORKSPACE);
      end;

    //setting the form background color as the negative value
    // of the current background color
    currentColorInt := clWhite - currentColorInt;

    SetEnumProp(Self, 'Color', IntToStr(currentColorInt));
  end;

  procedure TForm3.Button2Click(Sender: TObject);
  var
    p : array [0..4] of Integer;
    currentAlignProp : String;
  begin
    p[0] := clYellow;
    p[1] := clRed;
    p[2] := clBlue;
    p[3] := clBlack;
    p[4] := clGreen;

      SetEnumProp(Self, 'Color', IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    currentAlignProp := GetEnumProp(Button2, 'Align');
    if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alRight) then
       SetEnumProp(Button2, 'Align', 'alRight');

  end;
```

See Also

GetEnumProp (see page 4742)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.39 TypInfo.SetFloatProp Function

Sets the value of a component property that is a float type.

Pascal

```
procedure SetFloatProp(Instance: TObject; const PropName: string; const Value: Extended);
overload;
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo; const Value: Extended);
overload;
```

C++

```
SetFloatProp(TObject * Instance, const UnicodeString PropName, const Extended Value);
SetFloatProp(TObject * Instance, PPropInfo PropInfo, const Extended Value);
```

File

TypInfo

Description

SetFloatProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a floating point number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetFloatProp (see page 4745)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.40 TypInfo.SetInt64Prop Function

Sets the value of a component property that is an Int64 type.

Pascal

```
procedure SetInt64Prop(Instance: TObject; const PropName: string; const Value: Int64);
overload;
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo; const Value: Int64); overload;
```

C++

```
SetInt64Prop(TObject * Instance, const UnicodeString PropName, const Int64 Value);
SetInt64Prop(TObject * Instance, PPropInfo PropInfo, const Int64 Value);
```

File

TypInfo

Description

SetInt64Prop utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an Int64 number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetInt64Prop (see page 4746)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.41 TypInfo.SetInterfaceProp Function

Sets the value of a component property that is an Interface type.

Pascal

```
procedure SetInterfaceProp(Instance: TObject; const PropName: string; const Value:
IInterface); overload;
procedure SetInterfaceProp(Instance: TObject; PropInfo: PPropInfo; const Value: IInterface);
overload;
```

C++

```
SetInterfaceProp(TObject * Instance, const UnicodeString PropName, const IInterface Value);
SetInterfaceProp(TObject * Instance, PPropInfo PropInfo, const IInterface Value);
```

File

TypInfo

Description

SetInterfaceProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an Interface.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetInterfaceProp (see page 4746)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.42 TypInfo.SetMethodProp Function

Sets the value of a component property that is a method type.

Pascal

```
procedure SetMethodProp(Instance: TObject; const PropName: string; const Value: TMethod);
overload;
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo; const Value: TMethod);
overload;
```

C++

```
SetMethodProp(TObject * Instance, const UnicodeString PropName, const TMethod Value);
SetMethodProp(TObject * Instance, PPropInfo PropInfo, const TMethod Value);
```

File

TypInfo

Description

SetMethodProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an Event (Method property).

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetMethodProp (see page 4747)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.43 TypInfo.SetObjectProp Function

Sets the value of a component property that is an object type.

Pascal

```
procedure SetObjectProp(Instance: TObject; const PropName: string; Value: TObject); overload;
procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo; Value: TObject; ValidateClass:
Boolean = True); overload;
```

C++

```
SetObjectProp(TObject * Instance, const UnicodeString PropName, TObject * Value);
SetObjectProp(TObject * Instance, PPropInfo PropInfo, TObject * Value, Boolean ValidateClass =
True);
```

File

TypInfo

Description

SetObjectProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetObjectProp (see page 4748)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.44 TypInfo.SetOrdProp Function

Sets the value of a component property that is an ordinal type.

Pascal

```
procedure SetOrdProp(Instance: TObject; const PropName: string; Value: Longint); overload;
procedure SetOrdProp(Instance: TObject; PropInfo: PPropInfo; Value: Longint); overload;
```

C++

```
SetOrdProp(TObject * Instance, const UnicodeString PropName, Longint Value);
SetOrdProp(TObject * Instance, PPropInfo PropInfo, Longint Value);
```

File

TypInfo

Description

SetOrdProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetOrdProp (see page 4750)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.45 TypInfo.SetPropValue Function

Sets the value of a component property using a variant value.

Pascal

```
procedure SetPropValue(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo; const Value: Variant); overload;
```

C++

```
SetPropValue(TObject * Instance, const UnicodeString PropName, const Variant Value);
SetPropValue(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetPropValue utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property using a variant value.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. Note that only certain property assignments are possible using variants. It is normally be used when using GetPropValue (see page 4753) from another property.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetPropValue (see page 4753)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue

PPropInfo (see page 4789)

48.2.46 TypInfo.SetSetProp Function

Sets the value of a component property that is a set type.

Pascal

```
procedure SetSetProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetSetProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetSetProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetSetProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetSetProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a set type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. You specify the set of values as a comma separated string. You may optionally enclose these in square brackets. This allows the returned value from one GetSetProp (see page 4755) call to be used as the assignment value in a SetSetProp call.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetSetProp (see page 4755)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.47 TypInfo.SetStrProp Function

Sets the value of a component property that is a string type.

Pascal

```
procedure SetStrProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetStrProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetStrProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetStrProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a String, LongString or WideString (see page 3476) type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is

thrown.

See Also

GetStrProp (see page 4756)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.48 TypInfo.SetToString Function

Returns one or more of the set values of a component's property.

Pascal

```
function SetToString(PropInfo: PPropInfo; Value: Integer; Brackets: Boolean = False): string;
```

C++

```
UnicodeString SetToString(PPropInfo PropInfo, int Value, Boolean Brackets = False);
```

File

TypInfo

Description

SetToString utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to retrieve one or more of the values of a component property that is of the set type.

The property to interrogate is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

Value defines a bit mask that selects the possible values of this property. Each bit corresponds to a set value. The low order bit corresponds to the first set value. A string is returned with the selected set values as a comma separated value string. Note that the current property set values are not taken into account - the set values selected by **Value** are those in the complete set.

You may optionally set **Brackets** true so as to enclose this returned string within square brackets, as shown in the IDE for the property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetSetProp (see page 4755)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

PPropInfo (see page 4789)

SetPropValue (see page 4771)

SetSetProp (see page 4772)

48.2.49 TypInfo.SetUnicodeStrProp Function

Pascal

```
procedure SetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo; const Value:
UnicodeString); overload;
```

C++

```
SetUnicodeStrProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

This is function TypInfo.SetUnicodeStrProp.

48.2.50 TypInfo.SetUnicodeStrProp Function

Pascal

```
procedure SetUnicodeStrProp(Instance: TObject; const PropName: string; const Value:
UnicodeString); overload;
```

C++

```
SetUnicodeStrProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
```

File

TypInfo

Description

This is function TypInfo.SetUnicodeStrProp.

48.2.51 TypInfo.SetVariantProp Function

Sets the value of a component property that is a variant type.

Pascal

```
procedure SetVariantProp(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo; const Value: Variant);
overload;
```

C++

```
SetVariantProp(TObject * Instance, const UnicodeString PropName, const Variant Value);
SetVariantProp(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetVariantProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetVariantProp (see page 4758)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.52 TypInfo.SetWideStrProp Function

Sets the value of a component property that is a WideString (see page 3476) type.

Pascal

```
procedure SetWideStrProp(Instance: TObject; const PropName: string; const Value: WideString);
overload;
procedure SetWideStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: WideString);
overload;
```

C++

```
SetWideStrProp(TObject * Instance, const UnicodeString PropName, const BSTR Value);
SetWideStrProp(TObject * Instance, PPropInfo PropInfo, const BSTR Value);
```

File

TypInfo

Description

SetWideStrProp utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to set the value of a component's property where that property is a WideString (see page 3476) type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4771) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4753) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4727) exception is thrown.

See Also

GetWideStrProp (see page 4759)

GetPropInfo (see page 4751)

GetPropList (see page 4752)

SetPropValue (see page 4771)

PPropInfo (see page 4789)

48.2.53 TypInfo.SortPropList Function

Sorts a list of component properties into alphanumeric sequence.

Pascal

```
procedure SortPropList(PropList: PPropList; PropCount: Integer);
```

C++

```
SortPropList(PPropList PropList, int PropCount);
```

File

TypInfo

Description

SortPropList is used to sort a PPropList (see page 4790) that was originally created in natural rather than sorted sequence (by GetPropList (see page 4752)).

PropList defines a pointer to the array of TPropInfo (see page 4784) pointers (normally obtained using GetPropList (see page 4752)). **Count** specifies the number of TPropList (see page 4800)

See Also

GetPropList (see page 4752)

PTypeInfo (see page 4794)

PPropList (see page 4790)

PPropInfo (see page 4789)

TPropInfo (see page 4784)

48.2.54 TypInfo.StringToSet Function

Returns a bit mask of set values for a set type component property.

Pascal

```
function StringToSet(PropInfo: PPropInfo; const Value: string): Integer;
```

C++

```
int StringToSet(PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

StringToSet utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a bit mask of set indices for component's property where that property is a set type.

The property is defined by the **PropInfo** pointer. (GetPropInfo (see page 4751) and GetPropList (see page 4752) can be used to obtain such a record).

Value defines a comma separated string of set values the property may have.

The resulting integer value is a bit mask. The 0 bit relates to the 1st set value for the property. If that property value is included in the **Value** string, it is set to 1, otherwise 0. This continues for bits 1, 2 and so on referring to set positions 2, 3 and so on.

See Also

GetPropList (see page 4752)

SetToString (see page 4774)

PPropInfo (see page 4789)

TPropInfo (see page 4784)

48.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFloatType (see page 4779)	This is record TypInfo.TFloatType.
	TIntfFlag (see page 4780)	This is record TypInfo.TIntfFlag.
	TMethodKind (see page 4780)	Enumerates floating point number types in the TTypeData (see page 4785) record.
	TOrdType (see page 4782)	Enumerates floating point number types in the TTypeData (see page 4785) record.
	TParamFlag (see page 4783)	TParamFlag and TParamFlags (see page 4799) indicates the attributes of a method parameter.
	TTypeKind (see page 4787)	These types describe a type that is manipulated in the forms designer.

Records

	Name	Description
	TManagedField (see page 4780)	This is record TypInfo.TManagedField.
	TPropData (see page 4783)	This is record TypInfo.TPropData.
	TPropInfo (see page 4784)	TPropInfo is a record of component property type information.
	TTypeData (see page 4785)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 4786)	Holds information for a data type as returned by TypeInfo.

48.3.1 TypInfo.TFloatType Enumeration

Pascal

```
TFloatType = (
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
);
```

C++

```
enum TFloatType {
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
};
```

File

TypInfo

Description

This is record TypInfo.TFloatType.

48.3.2 TypInfo.TIntfFlag Enumeration

Pascal

```
TIntfFlag = (
  ifHasGuid,
  ifDispInterface,
  ifDispatch
);
```

C++

```
enum TIntfFlag {
  ifHasGuid,
  ifDispInterface,
  ifDispatch
};
```

File

TypInfo

Description

This is record TypInfo.TIntfFlag.

48.3.3 TypInfo.TManagedField Record

Pascal

```
TManagedField = record
  TypeRef: PTypeInfo;
  FldOffset: Integer;
end;
```

C++

```
struct TManagedField {
  PTypeInfo TypeRef;
  int FldOffset;
};
```

File

TypInfo

Description

This is record TypInfo.TManagedField.

48.3.4 TypInfo.TMethodKind Enumeration

Enumerates floating point number types in the TTypeData (see page 4785) record.

Pascal

```
TMethodKind = (
```

```
    mkProcedure,
    mkFunction,
    mkConstructor,
    mkDestructor,
    mkClassProcedure,
    mkClassFunction,
    mkClassConstructor,
    mkOperatorOverload,
    mkSafeProcedure,
    mkSafeFunction
  );
```

C++

```
  enum TMethodKind {
    mkProcedure,
    mkFunction,
    mkConstructor,
    mkDestructor,
    mkClassProcedure,
    mkClassFunction,
    mkClassConstructor,
    mkOperatorOverload,
    mkSafeProcedure,
    mkSafeFunction
  };
```

File

TypInfo

Description

The GetTypeData (see page 4756) routine utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a pointer to TTypeData (see page 4785) record that describes a component property data type.

TTypeData (see page 4785) is a variant record : the layout depends on the property TTypeKind (see page 4787) value. When it is tkMethod, the data is one of the TMethodKind enumerated values.

The TMethodKind value meanings are as follows :

Value	Meaning
mkProcedure	Stand alone procedure.
mkFunction	Stand alone function.
mkDestructor	Destructor.
mkConstructor	Constructor.
mkClassProcedure	Class procedure.
mkClassFunction	Class function.
mkClassConstructor	Class constructor.
mkOperatorOverload	Operator overload.
mkSafeProcedure	Safe procedure.
mkSafeFunction	Safe function.

See Also

GetTypeData (see page 4756)

TTypeData (see page 4785)

TTypeKind (see page 4787)

48.3.5 TypInfo.TOrdType Enumeration

Enumerates floating point number types in the TTypeData (see page 4785) record.

Pascal

```
TOrdType = (
  otSByte,
  otUByte,
  otSWord,
  otUWord,
  otSLong,
  otULong
);
```

C++

```
enum TOrdType {
  otSByte,
  otUByte,
  otSWord,
  otUWord,
  otSLong,
  otULong
};
```

File

TypInfo

Description

The GetTypeData (see page 4756) routine utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a pointer to TTypeData (see page 4785) record that describes a component property data type.

TTypeData (see page 4785) is a variant record : the layout depends on the property TTypeKind (see page 4787) value. When it is tkWChar, the data is one of the TOrdType enumerated values.

The TOrdType value meanings are as follows :

Value	Meaning
otSByte	Signed byte.
otUByte	Unsigned byte
otSWord	Signed Word
otUWord	Unsigned Word
otSLong	Signed Long
otULong	Unsigned Long

See Also

GetTypeData (see page 4756)

TTypeData (see page 4785)

TTypeKind (see page 4787)

48.3.6 TypInfo.TParamFlag Enumeration

TParamFlag and TParamFlags (see page 4799) indicates the attributes of a method parameter.

Pascal

```
TParamFlag = (
  pfVar,
  pfConst,
  pfArray,
  pfAddress,
  pfReference,
  pfOut
);
```

C++

```
enum TParamFlag {
  pfVar,
  pfConst,
  pfArray,
  pfAddress,
  pfReference,
  pfOut
};
```

File

TypInfo

Description

TParamFlags (see page 4799) indicates the attributes of a parameter on a Delphi language function or procedure. It is a set of TParamFlag values:

Value	Meaning
pfVar	The parameter is input/output. Caller must supply a variable to receive output.
pfConst	The parameter can't be changed by the function or procedure.
pfArray	The parameter represents an open array of values.
pfAddress	The parameter is a memory address (pointer).
pfReference	The parameter is passed by reference.
pfOut	The parameter is output. Caller supplies a variable to receive the output.

48.3.7 TypInfo.TPropData Record

Pascal

```
TPropData = packed record
  PropCount: Word;
  PropList: record;
end;
```

C++

```
struct TPropData {
```

```
  Word PropCount;
  record PropList;
};
```

File

TypInfo

Description

This is record TypInfo.TPropData.

48.3.8 TypInfo.TPropInfo Record

TPropInfo is a record of component property type information.

Pascal

```
TPropInfo = packed record
  PropType: PPTypeInfo;
  GetProc: Pointer;
  SetProc: Pointer;
  StoredProc: Pointer;
  Index: Integer;
  Default: Longint;
  NameIndex: SmallInt;
  Name: ShortString;
end;
```

C++

```
struct TPropInfo {
  PPTypeInfo PropType;
  void * GetProc;
  void * SetProc;
  void * StoredProc;
  int Index;
  Longint Default;
  SmallInt NameIndex;
  ShortString Name;
};
```

File

TypInfo

Description

TPropInfo is principally used to hold the results of a call to GetPropInfo (see page 4751). It provides meta details about a component property as part of Delphi's RTTI (Run Time (see page 4338) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values.

The TPropInfo fields have the foillowing meanings :

Field	Meaning
PropType	Points to a data structure that gives information about the property type.
GetProc	Method pointer for the method that gets the component property value.
SetProc	Method pointer for the method that sets the component property value.
Index	The index of the property. It is used as a parameter to the GetProc and SetProc methods on indexed properties.

Default	The default value of the property.
NameIndex	The offset into the Name field where the property's name starts.
Name	The qualified property name.

See Also

GetPropInfo (see page 4751)

PPropInfo (see page 4789)

48.3.9 TypInfo.TTypeData Record

TTypeData is used internally to represent type information.

Pascal

```
TTypeData = packed record
  case TTypeKind of
    tkClass: (ClassType: TClass;
              ParentInfo: PPTypeInfo;
              PropCount: SmallInt;
              UnitName: ShortStringBase;);
    tkDynArray: (elSize: Longint;
                 elType: PPTypeInfo;
                 varType: Integer;
                 elType2: PPTypeInfo;
                 DynUnitName: ShortStringBase;);
    tkEnumeration: (BaseType: PPTypeInfo;
                    NameList: ShortStringBase;
                    EnumUnitName: ShortStringBase;);
    tkFloat: (FloatType: TFloatType;);
    tkInt64: (MaxInt64Value: Int64;
              MinInt64Value: Int64;);
    tkLString: (CodePage: Word;);
    tkMethod: (MethodKind: TMethodKind;
               ParamCount: Byte;
               ParamList: array[0..1023] of AnsiChar );
               tkInterface: ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid :
TGUID; IntfUnit : ShortStringBase;;);
    tkRecord: (RecSize: Integer;
               ManagedFldCount: Integer;);
    tkSet: (CompType: PPTypeInfo;);
    tkString: (MaxLength: Byte;);
    tkWChar: (OrdType: TOrdType;
              MinValue: Longint;
              MaxValue: Longint;);
end;
```

C++

```
struct TTypeData {
  Word CodePage;
  TOrdType OrdType;
  Longint MinValue;
  Longint MaxValue;
  PPTypeInfo BaseType;
  ShortStringBase NameList;
  ShortStringBase EnumUnitName;
  PPTypeInfo CompType;
  TFloatType FloatType;
```

```
    Byte MaxLength;
    TClass ClassType;
    PPTypeInfo ParentInfo;
    SmallInt PropCount;
    ShortStringBase UnitName;
    TMethodKind MethodKind;
    Byte ParamCount;
    array[0..1023] of AnsiChar ) ParamList;
    ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid : TGUID; IntfUnit :
  ShortStringBase; tkInterface;
    Int64 MaxInt64Value;
    Int64 MinInt64Value;
    Longint elSize;
    PPTypeInfo elType;
    int varType;
    PPTypeInfo elType2;
    ShortStringBase DynUnitName;
    int RecSize;
    int ManagedFldCount;
  };
```

File

TypInfo

Description

A TTypeData structure represents a category of properties, events, or methods in parameters passed to several routines in the TypInfo (see page 4723) unit.

48.3.10 TypInfo.TTypeInfo Record

Holds information for a data type as returned by TypeInfo.

Pascal

```
TTypeInfo = record
  Kind: TTypeKind;
  Name: ShortString;
end;
```

C++

```
struct TTypeInfo {
  TTypeKind Kind;
  ShortString Name;
};
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4338) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 4794) pointer to a TTypeInfo value.

The TTypeInfo holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 4785) returned by the GetTypeData (see page 4756).

Name	The name of the data type.

See Also

GetTypeData (see page 4756)

PTypeInfo (see page 4794)

PPTypeInfo (see page 4790)

TTypeKind (see page 4787)

TTypeData (see page 4785)

48.3.11 TypInfo.TTypeKind Enumeration

These types describe a type that is manipulated in the forms designer.

Pascal

```
TTypeKind = (
  tkUnknown,
  tkInteger,
  tkChar,
  tkEnumeration,
  tkFloat,
  tkString,
  tkSet,
  tkClass,
  tkMethod,
  tkWChar,
  tkLString,
  tkWString,
  tkVariant,
  tkArray,
  tkRecord,
  tkInterface,
  tkInt64,
  tkDynArray,
  tkUString
);
```

C++

```
enum TTypeKind {
  tkUnknown,
  tkInteger,
  tkChar,
  tkEnumeration,
  tkFloat,
  tkString,
  tkSet,
  tkClass,
  tkMethod,
  tkWChar,
  tkLString,
  tkWString,
  tkVariant,
  tkArray,
  tkRecord,
  tkInterface,
  tkInt64,
```

```
    tkDynArray,
    tkUString
};
```

File

TypInfo

Description

TTypeKind is an enumerated type that classifies a type.

TTypeInfo (see page 4786) describes a type by giving its classification and its name.

PTypeInfo (see page 4794) is a pointer to a TTypeInfo (see page 4786) value.

PPTypeInfo (see page 4790) is a pointer to a PTypeInfo (see page 4794) pointer.

48.4 Types

The following table lists types in this documentation.

Types

Name	Description
PPropInfo (see page 4789)	PPropInfo points to a TPropInfo (see page 4784) record that holds component property information.
PPropList (see page 4790)	Defines a pointer to a TPropList (see page 4800) array.
PPTypeInfo (see page 4790)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 4791)	PTypeData points to a TTypeData (see page 4785) record that holds component property type information.
PTypeInfo (see page 4794)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 4797)	Defines a fixed length string for general use in the TTypeData (see page 4785) record.
TIntfFlags (see page 4797)	Enumerates interface flag types in the TTypeData (see page 4785) record.
TIntfFlagsBase (see page 4798)	Set of interface flag types in the TTypeData (see page 4785) record.
TParamFlags (see page 4799)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 4799)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 4799)	This is type TypInfo.TPropInfoProc.
TPropList (see page 4800)	Defines an array of PPropInfo (see page 4789) values.
TTypeKinds (see page 4800)	Defines a set of TTypeKind (see page 4787) values.

48.4.1 TypInfo.PPropInfo Type

PPropInfo points to a TPropInfo (see page 4784) record that holds component property information.

Pascal

```
PPropInfo = ^TPropInfo;
```

C++

```
^TPropInfo PPropInfo;
```

File

TypInfo

Description

TPropInfo (see page 4784) is principally used to hold the results of a call to GetPropInfo (see page 4751). It provides meta details about a component property as part of Delphi's RTTI (Run Time (see page 4338) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PPropInfo is a pointer to a TPropInfo (see page 4784) record.

See Also

GetPropInfo (see page 4751)

TPropInfo (see page 4784)

48.4.2 TypInfo.PPropList Type

Defines a pointer to a TPropList (see page 4800) array.

Pascal

```
PPropList = ^TPropList;
```

C++

```
^TPropList PPropList;
```

File

TypInfo

Description

PPropList defines a pointer to an array of PPropInfo (see page 4789) values. This TPropList (see page 4800) array is principally used to hold the results of a call to GetPropList (see page 4752). This list (array) holds pointers to TPropInfo (see page 4784) records that describe the nature of the properties of the component provided to GetPropList (see page 4752).

See Also

GetPropList (see page 4752)

PPropInfo (see page 4789)

TPropInfo (see page 4784)

TPropList (see page 4800)

48.4.3 TypInfo.PPTypeInfo Type

Holds a pointer to a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PPTypeInfo = ^PTypeInfo;
```

C++

```
^PTypeInfo PPTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4338) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 4794) pointer to a TTypeInfo (see page 4786) value. PPTypeInfo is a PTypeInfo (see page 4794) pointer.

See Also

GetTypeData (see page 4756)

PTypeInfo (see page 4794)

TTypeInfo (see page 4786)

TTypeKind (see page 4787)

TTypeData (see page 4785)

48.4.4 TypInfo.PTypeData Type

PTypeData points to a TTypeData (see page 4785) record that holds component property type information.

Pascal

```
PTypeData = ^TTypeData;
```

C++

```
^TTypeData PTypeData;
```

File

TypInfo

Description

TTypeData (see page 4785) is principally used to hold the results of a call to GetTypeData (see page 4756). It provides meta details about a component property type as part of Delphi's RTTI (Run Time (see page 4338) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PTypeData is a pointer to a TTypeData (see page 4785) record.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
```

```
  This example will print useful information about any given
  type including it's name, size, kind and sub-kind.
  }

  type
    TForm2 = class(TForm)
      Memo1: TMemo;
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;
  { Defining a generic type that contains a single class (static)
    method that will print information about the type it operates on.
  }
    TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
    public
      procedure PrintTypeInfo(memo: TMemo);
    end;
    TGenericString = class(TGenericClass<UnicodeString>)
    end;
    TGenericByte = class(TGenericClass<Byte>)
    end;
    TGenericDouble = class(TGenericClass<Double>)
    end;

  var
    Form2: TForm2;

  implementation

  {$R *.dfm}

  { Method defintition }
  procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
  var
    Info     : PTypeInfo;
    Data     : PTypeData;
    KindName : String;
    SubName  : String;
  begin
    { Get type info for the "yet unknown type" }
    Info := System.TypeInfo(T);

    { There is no RTTI attached for some types like Records for example. }
    if Info <> nil then
    begin
      memo.Lines.Add('Type name: ' + Info^.Name);

      { Find out the name of an enum item from it's ordinal value }
      KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

      memo.Lines.Add('Type kind: ' + KindName);

      Data := GetTypeData(Info);

      if Info^.Kind = tkInteger then
      begin
        { In case of integer let's see the actual sub-type name }
        SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
        memo.Lines.Add('Integer kind: ' + SubName);
      end;

      if Info^.Kind = tkFloat then
      begin
```

```
        { In case of float let's see the actual sub-type name }
        SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
        memo.Lines.Add('Float kind: ' + SubName);
      end;

      if Info^.Kind = tkDynArray then
      begin
        { Let's check out the element size  }
        memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
        memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
      end;

    end;
    memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
  end;


  procedure TForm2.FormCreate(Sender: TObject);
  var
    GenericString : TGenericClass<UnicodeString>;
    GenericByte : TGenericClass<Byte>;
    GenericDouble : TGenericClass<Double>;
  //  GenericStringArray : TGenericClass<array of string>;
  begin
    { Prints type info for string type}

    GenericString := TGenericClass<UnicodeString>.Create;
    GenericString.PrintTypeInfo(Memo1);

    { Prints type info for Byte type}
    GenericByte := TGenericClass<Byte>.Create;
    GenericByte.PrintTypeInfo(Memo1);

    { Prints type info for Double type}
    GenericDouble := TGenericClass<Double>.Create;
    GenericDouble.PrintTypeInfo(Memo1);
  end;

  {
  Expected results are:

  Type name: string
  Type kind: tkUString
  Size of type: 4

  Type name: Byte
  Type kind: tkInteger
  Integer kind: otUByte
  Size of type: 1

  Type name: Double
  Type kind: tkFloat
  Float kind: ftDouble
  Size of type: 8

  Type name: .1
  Type kind: tkDynArray
  Array element type name: string
  Array element type size: 4
  Size of type: 4
  }
```

See Also

GetTypeData (see page 4756)

TTypeData (see page 4785)

48.4.5 TypInfo.PTypeInfo Type

Holds a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PTypeInfo = ^TTypeInfo;
```

C++

```
^TTypeInfo PTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4338) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo pointer to a TTypeInfo (see page 4786) value.

The TTypeInfo (see page 4786) holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 4785) returned by the GetTypeData (see page 4756).
Name	The name of the data type.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;
```

```
    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;

  end;
  memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
end;


procedure TForm2.FormCreate(Sender: TObject);
var
  GenericString : TGenericClass<UnicodeString>;
  GenericByte : TGenericClass<Byte>;
  GenericDouble : TGenericClass<Double>;
//  GenericStringArray : TGenericClass<array of string>;
begin
  { Prints type info for string type}

  GenericString := TGenericClass<UnicodeString>.Create;
  GenericString.PrintTypeInfo(Memo1);

  { Prints type info for Byte type}
  GenericByte := TGenericClass<Byte>.Create;
  GenericByte.PrintTypeInfo(Memo1);

  { Prints type info for Double type}
  GenericDouble := TGenericClass<Double>.Create;
  GenericDouble.PrintTypeInfo(Memo1);
end;

{
Expected results are:

Type name: string
Type kind: tkUString
Size of type: 4

Type name: Byte
Type kind: tkInteger
Integer kind: otUByte
Size of type: 1

Type name: Double
Type kind: tkFloat
Float kind: ftDouble
Size of type: 8

Type name: .1
Type kind: tkDynArray
Array element type name: string
Array element type size: 4
Size of type: 4
}
```

See Also

GetTypeData (see page 4756)

PPTypeInfo (see page 4790)

TTypeInfo (see page 4786)

TTypeKind (see page 4787)

TTypeData (see page 4785)

48.4.6 TypInfo.ShortStringBase Type

Defines a fixed length string for general use in the TTypeData (see page 4785) record.

Pascal

```
ShortStringBase = string[255];
```

C++

```
string[255] ShortStringBase;
```

File

TypInfo

Description

The GetTypeData (see page 4756) routine utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a pointer to TTypeData (see page 4785) record that describes a component property data type.

A number of the variant data sections in the TTypeData (see page 4785) record use the ShortStringBase to define a 255 character string to hold Name values such as UnitName for a class data type.

See Also

GetTypeData (see page 4756)

TTypeData (see page 4785)

48.4.7 TypInfo.TIntfFlags Type

Enumerates interface flag types in the TTypeData (see page 4785) record.

Pascal

```
TIntfFlags = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlags;
```

File

TypInfo

Description

The GetTypeData (see page 4756) routine utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a

pointer to TTypeData (see page 4785) record that describes a component property data type.

TTypeData (see page 4785) is a variant record : the layout depends on the property TTypeKind (see page 4787) value. When it is tkInterface, the data contains a TIntfFlagsBase (see page 4798) set of TIntfFlags values.

The TIntfFlags value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.
ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 4756)

TIntfFlagsBase (see page 4798)

TTypeData (see page 4785)

TTypeKind (see page 4787)

48.4.8 TypInfo.TIntfFlagsBase Type

Set of interface flag types in the TTypeData (see page 4785) record.

Pascal

```
TIntfFlagsBase = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlagsBase;
```

File

TypInfo

Description

The GetTypeData (see page 4756) routine utilises Delphi's RTTI (Run Time (see page 4338) Type Information) to return a pointer to TTypeData (see page 4785) record that describes a component property data type.

TTypeData (see page 4785) is a variant record : the layout depends on the property TTypeKind (see page 4787) value. When it is tkInterface, the data contains a TIntfFlagsBase set of TIntfFlag (see page 4780) values.

The TIntfFlag (see page 4780) value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.
ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 4756)

TIntfFlag (see page 4780)

TTypeData (see page 4785)

TTypeKind (see page 4787)

48.4.9 TypInfo.TParamFlags Type

Pascal

```
TParamFlags = set of TParamFlag;
```

C++

```
TParamFlag TParamFlags;
```

File

TypInfo

Description

This is type TypInfo.TParamFlags.

48.4.10 TypInfo.TParamFlagsBase Type

Pascal

```
TParamFlagsBase = set of TParamFlag;
```

C++

```
TParamFlag TParamFlagsBase;
```

File

TypInfo

Description

This is type TypInfo.TParamFlagsBase.

48.4.11 TypInfo.TPropInfoProc Type

Pascal

```
TPropInfoProc = procedure (PropInfo: PPropInfo) of object;
```

C++

```
(PropInfo: PPropInfo) ( TPropInfoProc)();
```

File

TypInfo

Description

This is type TypInfo.TPropInfoProc.

48.4.12 TypInfo.TPropList Type

Defines an array of PPropInfo (see page 4789) values.

Pascal

```
TPropList = array[0..16379] of PPropInfo;
```

C++

```
array[0..16379] of PPropInfo TPropList;
```

File

TypInfo

Description

TPropList is principally used to hold the results of a call to GetPropList (see page 4752). This list (array) holds pointers to TPropInfo (see page 4784) records that describe the nature of the properties of the component provided to GetPropList (see page 4752).

See Also

GetPropList (see page 4752)

TPropInfo (see page 4784)

48.4.13 TypInfo.TTypeKinds Type

Defines a set of TTypeKind (see page 4787) values.

Pascal

```
TTypeKinds = set of TTypeKind;
```

C++

```
TTypeKind TTypeKinds;
```

File

TypInfo

Description

TTypeKinds defines a set of TTypeKind (see page 4787) values. It is used to select which component property types to use in various routines, such as GetPropInfo (see page 4751).

See Also

GetPropInfo (see page 4751)

TTypeKind (see page 4787)

48.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanIdents (see page 4801)	An array of string representations of true and false.
DotSep (see page 4801)	This is variable TypInfo.DotSep.

48.5.1 TypInfo.BooleanIdents Variable

An array of string representations of true and false.

Pascal

```
BooleanIdents: array [Boolean] of string = ('False', 'True');
```

C++

```
array [Boolean] of string BooleanIdents = ('False', 'True');
```

File

TypInfo

Description

The GetEnumName (see page 4739) routine is part of Delphi's RTTI (Run Time (see page 4338) Type Information). It returns a string representation of an enumerated type value. If that type is true or false, BooleanIdents is used internally.

See Also

GetEnumName (see page 4739)

48.5.2 TypInfo.DotSep Variable

Pascal

```
DotSep: string = '.';
```

C++

```
UnicodeString DotSep = '.';
```

File

TypInfo

Description

This is variable TypInfo.DotSep.

48.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
tkAny (see page 4802)	This is constant TypInfo.tkAny.
tkMethods (see page 4802)	This is constant TypInfo.tkMethods.
tkProperties (see page 4802)	This is constant TypInfo.tkProperties.

48.6.1 TypInfo.tkAny Constant

Pascal

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

C++

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

File

TypInfo

Description

This is constant TypInfo.tkAny.

48.6.2 TypInfo.tkMethods Constant

Pascal

```
tkMethods = [tkMethod];
```

C++

```
tkMethods = [tkMethod];
```

File

TypInfo

Description

This is constant TypInfo.tkMethods.

48.6.3 TypInfo.tkProperties Constant

Pascal

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

C++

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

File

TypInfo

Description

This is constant TypInfo.tkProperties.

49 UDDIHelper Namespace

This is namespace UDDIHelper.

Topics

Name	Description
Functions (see page 4806)	The following table lists functions in this documentation.

Functions

	Name	Description
	GetBindingkeyAccessPoint (see page 4806)	Returns the URL for a Web Service as specified in a UDDI browser.

49.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetBindingkeyAccessPoint (see page 4806)	Returns the URL for a Web Service as specified in a UDDI browser.

49.1.1 UDDIHelper.GetBindingkeyAccessPoint Function

Returns the URL for a Web Service as specified in a UDDI browser.

Pascal

```
function GetBindingkeyAccessPoint(const Operator: String; const key: String): String;
```

C++

```
UnicodeString GetBindingkeyAccessPoint(const UnicodeString Operator, const UnicodeString key);
```

File

UDDIHelper

Description

Use GetBindingkeyAccessPoint to fetch the URL for calling a Web Service from a specified UDDI registry.

Operator is the URL of the UDDI registry from which to fetch the entry. You can obtain this value for a registered invokable interface by calling the invocation registry's GetUDDIInfo method.

BindingKey is the unique identifier for the TModel that describes the Web Service for which you want the URL.

GetBindingkeyAccessPoint returns the URL for the specified Web Service. This can be used to set the URL property of a THTTPRIO component.

See Also

THTTPRIO

TInvokableClassRegistry

50 ValEdit Namespace

This is namespace ValEdit.

Topics

Name	Description
Classes (see page 4808)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4921)	The following table lists structs, records, enums in this documentation.
Types (see page 4923)	The following table lists types in this documentation.

Classes

	Name	Description
	TItemProp (see page 4808)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 4814)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 4911)	This is class ValEdit.TValueListStrings.

Structs, Records, Enums

	Name	Description
	TDisplayOption (see page 4921)	TDisplayOption and TDisplayOptions (see page 4923) control the appearance of a value list editor.
	TKeyOption (see page 4922)	TKeyOption and TKeyOptions (see page 4924) indicate the response of a value list editor to certain keystrokes.

Types

Name	Description
TDisplayOptions (see page 4923)	This is type ValEdit.TDisplayOptions.
TGetPickListEvent (see page 4923)	This is type ValEdit.TGetPickListEvent.
TItemProps (see page 4924)	This is type ValEdit.TItemProps.
TKeyOptions (see page 4924)	This is type ValEdit.TKeyOptions.
TOnValidateEvent (see page 4924)	This is type ValEdit.TOnValidateEvent.

50.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TItemProp (see page 4808)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 4814)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 4911)	This is class ValEdit.TValueListStrings.

50.1.1 TItemProp Class

TItemProp configures how users can edit the item values in a value list editor.

Class Hierarchy

Pascal

```
TItemProp = class(TPersistent);
```

C++

```
class TItemProp : public TPersistent;
```

File

ValEdit

Description

For each item that it allows users to edit, the TValueListEditor (see page 4814) component maintains a TItemProp object. This object's properties govern how users can edit the value in the value list editor.

TItemProp objects are created and owned by the value list editor that uses them to represent properties of item values (such as an edit mask or maximum length) or to supply a fixed pick list of values that users can assign to the item.

See Also

TValueListEditor (see page 4814)

TValueListEditor (see page 4814)

50.1.1.1 TItemProp Members

The following tables list the members exposed by TItemProp.

Protected Methods

	Name	Description
	AssignTo (see page 4810)	Copies the properties of an object to a destination object.
	UpdateEdit (see page 4811)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	Create (see page 4809)	Creates an instance of TItemProp (see page 4808).
	Destroy (see page 4809)	Destroys the TPersistent instance and frees its memory.
	HasPickList (see page 4810)	Indicates whether the PickList (see page 4813) property provides a static set of values for this item.

Published Properties

	Name	Description
	EditMask (see page 4811)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 4812)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 4812)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 4813)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 4813)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 4814)	Indicates whether users can edit the item's value by typing in a new value.

50.1.1.2 TItemProp.Create Constructor

Creates an instance of TItemProp (see page 4808).

Pascal

```
constructor Create(AEditor: TValueListEditor);
```

C++

```
__fastcall TItemProp(TValueListEditor AEditor);
```

Description

Do not call Create directly in an application. TValueListEditor (see page 4814) creates TItemProp (see page 4808) instances as needed when you make assignments to its ItemProps property.

The AEditor parameter is the value list editor that creates the TItemProp (see page 4808) instance to represent the properties of one of its items.

See Also

ItemProps (see page 4851)

50.1.1.3 TItemProp.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TItemProp();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

System::TObject::Destroy (see page 3366)

50.1.1.4 TItemProp Methods

The methods of the TItemProp class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 4810)	Copies the properties of an object to a destination object.
	UpdateEdit (see page 4811)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	HasPickList (see page 4810)	Indicates whether the PickList (see page 4813) property provides a static set of values for this item.

50.1.1.4.1 TItemProp.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3985)

50.1.1.4.2 TItemProp.HasPickList Method

Indicates whether the PickList (see page 4813) property provides a static set of values for this item.

Pascal

```
function HasPickList: Boolean;
```

C++

```
__fastcall Boolean HasPickList();
```

Description

Call PickList (see page 4813) to determine whether this TItemProp (see page 4808) object supplies values using the PickList (see page 4813) property. HasPickList returns true if the PickList (see page 4813) property contains a nonempty list of values.

See Also

PickList (see page 4813)

50.1.1.4.3 TItemProp.UpdateEdit Method

Pascal

```
procedure UpdateEdit;
```

C++

```
__fastcall UpdateEdit();
```

Description

This is UpdateEdit, a member of class TItemProp.

50.1.1.5 TItemProp Properties

The properties of the TItemProp class are listed here.

Published Properties

	Name	Description
	EditMask (see page 4811)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 4812)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 4812)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 4813)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 4813)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 4814)	Indicates whether users can edit the item's value by typing in a new value.

50.1.1.5.1 TItemProp.EditMask Property

Restricts the value that can be entered for the associated item.

Pascal

```
property EditMask: string;
```

C++

```
__property UnicodeString EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character

is not accepted. Validation using the EditMask property is performed on a character-by-character basis.

A mask consists of three fields with semicolons separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are included as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

Setting EditMask to an empty string removes the mask.

For more information on using edit masks, see the TEditMask type.

See Also

Values (see page 4865)

OnSetEditText

50.1.1.5.2 TItemProp.EditStyle Property

Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.

Pascal

```
property EditStyle: TEditStyle;
```

C++

```
__property TEditStyle EditStyle;
```

Description

Set EditStyle to specify whether the cell in the value list editor that contains the associated value should allow users to drop down a pick list or to launch a dialog to edit the value.

When EditStyle is esSimple, the value cell is a simple edit box that lets users edit the associated value directly (unless the ReadOnly (see page 4814) property is true).

When EditStyle is esEllipsis, the value cell contains an ellipsis button that users can click to generate an OnEditButtonClick event. Typically, applications use this event to launch a dialog that edits the value.

When EditStyle is esPickList, the value cell contains a drop-down arrow that users can click to display a list of values from which to select. This value must be supplied by the PickList (see page 4813) property or in an OnGetPickList event handler (or supplied by the PickList (see page 4813) property and modified in an OnGetPickList event handler).

See Also

PickList (see page 4813)

OnGetPickList (see page 4889)

OnEditButtonClick (see page 4879)

ReadOnly (see page 4814)

KeyOptions (see page 4851)

DisplayOptions (see page 4841)

50.1.1.5.3 TItemProp.KeyDesc Property

Specifies the name of the item, which appears in the names column of the Value list editor.

Pascal

```
property KeyDesc: string;
```

C++

```
__property UnicodeString KeyDesc;
```

Description

Set KeyDesc to specify a name for this item. This name appears in the names column of the Value list editor.

If ItemProp is not set, then the name that appears in the key column of the Value list editor is the corresponding value of the Keys property.

See Also

Keys (see page 4851)

50.1.1.5.4 TItemProp.MaxLength Property

Specifies the maximum number of characters that users can enter for the item's value.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be typed when assigning a value to the item. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing value, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered.

See Also

Values (see page 4865)

EditMask (see page 4811)

50.1.1.5.5 TItemProp.PickList Property

Lists the items that appear in the drop-down pick list for the item.

Pascal

```
property PickList: TStrings;
```

C++

```
__property TStrings PickList;
```

Description

Use PickList to supply the values that appear in the drop-down pick list for the associated item. PickList lets you supply a static set of items that appear in the drop-down list for the associated item when EditStyle (see page 4812) is esPickList. These items can be subsequently modified by an OnGetPickList event handler.

Note: PickList is only used when the EditStyle (see page 4812) property is esPickList. If EditStyle (see page 4812) is

esPickList, and PickList does list a set of values, you must supply an OnGetPickList event handler to supply the items.

See Also

EditStyle (see page 4812)

OnGetPickList (see page 4889)

DropDownRows (see page 4846)

50.1.1.5.6 TItemProp.ReadOnly Property

Indicates whether users can edit the item's value by typing in a new value.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent users from typing in a new value for the associated item. For example, to force users to select item values only from a pick list, you can set the EditStyle (see page 4812) to esPickList and set ReadOnly to true. Similarly, to force users to enter values only using a property editor, you can set EditStyle (see page 4812) to esEllipsis and set ReadOnly to true.

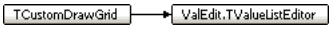
See Also

EditStyle (see page 4812)

50.1.2 TValueListEditor Class

TValueListEditor provides a simple way to edit string lists that contain name/value pairs.

Class Hierarchy



Pascal

```
TValueListEditor = class(TCustomDrawGrid);
```

C++

```
class TValueListEditor : public TCustomDrawGrid;
```

File

ValEdit

Description

Use TValueListEditor to let users edit string lists that consist of strings with the form Name=Value. By default, the names appear in a column headed "Key", and the values in a column headed "Value", although you can customize these labels. You can control whether users can select, add, or edit the strings in the Key column. For each entry in the Value column, you can let users edit the string, supply a drop-down list from which they can select values, or provide a dialog in which they can edit values.

See Also

TStringList

TStrings

Value List Editors (VCL Only)

50.1.2.1 TValueListEditor Members

The following tables list the members exposed by TValueListEditor.

Protected Methods

	Name	Description
	AdjustColWidths (see page 4819)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 4819)	This is AdjustRowCount, a member of class TValueListEditor.
	CanEditModify (see page 4820)	Indicates whether the current cell in the grid can be edited.
	ColWidthsChanged (see page 4820)	Responds when the column widths change.
	CreateEditor (see page 4820)	Creates the inplace editor control.
	CreateWnd (see page 4821)	Creates the underlying window.
	DoExit (see page 4822)	Hides the inplace editor when selection moves to another control in a form.
	DoOnValidate (see page 4822)	This is DoOnValidate, a member of class TValueListEditor.
	DrawCell (see page 4822)	Generates an OnDrawCell event.
	EditListGetItems (see page 4823)	This is EditListGetItems, a member of class TValueListEditor.
	GetCell (see page 4823)	This is GetCell, a member of class TValueListEditor.
	GetColCount (see page 4824)	This is GetColCount, a member of class TValueListEditor.
	GetEditLimit (see page 4824)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 4824)	Generates an OnGetEditMask event.
	GetEditStyle (see page 4824)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 4825)	This is GetEditText, a member of class TValueListEditor.
	GetPickList (see page 4825)	This is GetPickList, a member of class TValueListEditor.
	GetRowCount (see page 4825)	This is GetRowCount, a member of class TValueListEditor.
	IsEmptyRow (see page 4826)	This is IsEmptyRow, a member of class TValueListEditor.
	KeyDown (see page 4826)	Provides special processing when the user presses a key.
	MouseDown (see page 4827)	Provides special processing when the user presses a mouse button.
	Resize (see page 4828)	Respond to control resize.
	RowMoved (see page 4828)	Generates an OnRowMoved event.
	SelectCell (see page 4829)	Determines whether a particular cell in the grid can be selected.
	SetCell (see page 4829)	This is SetCell, a member of class TValueListEditor.
	SetEditText (see page 4829)	This is SetEditText, a member of class TValueListEditor.
	SetStrings (see page 4829)	This is SetStrings, a member of class TValueListEditor.
	StringsChanging (see page 4830)	This is StringsChanging, a member of class TValueListEditor.
	TitleCaptionsStored (see page 4830)	This is TitleCaptionsStored, a member of class TValueListEditor.

Protected Properties

	Name	Description
	EditList (see page 4846)	This is EditList, a member of class TValueListEditor.

Public Methods

	Name	Description
	Create (see page 4817)	Creates and initializes new TValueListEditor (see page 4814) object.
	DeleteRow (see page 4821)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	Destroy (see page 4818)	Frees the memory associated with the TValueListEditor (see page 4814) object.
	FindRow (see page 4823)	Locates the row that corresponds to a specified key name.
	InsertRow (see page 4825)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.

	Refresh (see page 4827)	Updates the display to reflect the current value of the Strings (see page 4863) property.
	RestoreCurrentRow (see page 4828)	Undoes all edits to the current row.

Public Properties

	Name	Description
	Cells (see page 4836)	Lists the strings for each cell in the editor.
R	ColCount (see page 4836)	Indicates the number of columns in the value list editor.
	ItemProps (see page 4851)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 4851)	Lists the strings for each cell in the name (key) column.
R	RowCount (see page 4861)	Indicates the number of rows in the value list editor.
	Values (see page 4865)	Lists the strings for each value from the value column.
	VisibleColCount (see page 4867)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 4867)	Indicates the number of scrollable rows visible in the grid.

Published Events

	Name	Description
	OnClick (see page 4868)	Occurs when the user clicks the control.
	OnContextPopup (see page 4870)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4871)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4871)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4874)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 4878)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 4879)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 4880)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4880)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4881)	Occurs when a control receives the input focus.
	OnExit (see page 4883)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 4886)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 4887)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 4889)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 4889)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4893)	Occurs when a key is pressed.
	OnKeyUp (see page 4894)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4896)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4897)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4899)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4899)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4899)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4902)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 4904)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4904)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 4905)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 4906)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 4908)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 4908)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4909)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnStringsChange (see page 4909)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 4910)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 4910)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 4910)	Occurs when focus shifts away from a cell in the value list editor.

Published Properties

	Name	Description
	Align (see page 4831)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4835)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4835)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4836)	Determines whether a single line border is drawn around the grid.
	Color (see page 4837)	Specifies the background color of the control.
	Constraints (see page 4838)	Specifies the size constraints for the control.
	Ctl3D (see page 4838)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 4839)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 4840)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 4841)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 4841)	Controls aspects of the value list editor's appearance.
	DoubleBuffered (see page 4842)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4843)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4844)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4844)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 4846)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 4847)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 4847)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 4847)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 4848)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 4849)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	KeyOptions (see page 4851)	Controls the way the value list editor responds to user keystrokes.
	Options (see page 4852)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 4853)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4854)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4854)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4855)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4855)	Determines where a control looks for its font information.
	ParentShowHint (see page 4855)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4856)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 4861)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 4862)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 4863)	Provides access to the string list that is being edited.
	TabOrder (see page 4864)	Indicates the position of the control in its parent's tab order.
	TitleCaptions (see page 4864)	Specifies the captions that label the columns of the value list editor.
	Visible (see page 4865)	Determines whether the component appears onscreen.

50.1.2.2 TValueListEditor.Create Constructor

Creates and initializes new TValueListEditor (see page 4814) object.

Pascal

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

C++

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

Description

Call Create to create a value list editor component at runtime. TValueListEditor (see page 4814) components added to forms at design time are created automatically.

The AOwner parameter specifies another component (typically the form) that is responsible for freeing the TValueListEditor (see page 4814) instance.

See Also

Owner

50.1.2.3 TValueListEditor.Destroy Destructor

Frees the memory associated with the TValueListEditor (see page 4814) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListEditor();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the value list editor reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

50.1.2.4 TValueListEditor Methods

The methods of the TValueListEditor class are listed here.

Protected Methods

	Name	Description
	AdjustColWidths (see page 4819)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 4819)	This is AdjustRowCount, a member of class TValueListEditor.
	CanEditModify (see page 4820)	Indicates whether the current cell in the grid can be edited.
	ColWidthsChanged (see page 4820)	Responds when the column widths change.
	CreateEditor (see page 4820)	Creates the inplace editor control.
	CreateWnd (see page 4821)	Creates the underlying window.
	DoExit (see page 4822)	Hides the inplace editor when selection moves to another control in a form.
	DoOnValidate (see page 4822)	This is DoOnValidate, a member of class TValueListEditor.
	DrawCell (see page 4822)	Generates an OnDrawCell event.
	EditListGetItems (see page 4823)	This is EditListGetItems, a member of class TValueListEditor.
	GetCell (see page 4823)	This is GetCell, a member of class TValueListEditor.
	GetColCount (see page 4824)	This is GetColCount, a member of class TValueListEditor.

	GetEditLimit (see page 4824)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 4824)	Generates an OnGetEditMask event.
	GetEditStyle (see page 4824)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 4825)	This is GetEditText, a member of class TValueListEditor.
	GetPickList (see page 4825)	This is GetPickList, a member of class TValueListEditor.
	GetRowCount (see page 4825)	This is GetRowCount, a member of class TValueListEditor.
	IsEmptyRow (see page 4826)	This is IsEmptyRow, a member of class TValueListEditor.
	KeyDown (see page 4826)	Provides special processing when the user presses a key.
	MouseDown (see page 4827)	Provides special processing when the user presses a mouse button.
	Resize (see page 4828)	Respond to control resize.
	RowMoved (see page 4828)	Generates an OnRowMoved event.
	SelectCell (see page 4829)	Determines whether a particular cell in the grid can be selected.
	SetCell (see page 4829)	This is SetCell, a member of class TValueListEditor.
	SetEditText (see page 4829)	This is SetEditText, a member of class TValueListEditor.
	SetStrings (see page 4829)	This is SetStrings, a member of class TValueListEditor.
	StringsChanging (see page 4830)	This is StringsChanging, a member of class TValueListEditor.
	TitleCaptionsStored (see page 4830)	This is TitleCaptionsStored, a member of class TValueListEditor.

Public Methods

	Name	Description
	DeleteRow (see page 4821)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	FindRow (see page 4823)	Locates the row that corresponds to a specified key name.
	InsertRow (see page 4825)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.
	Refresh (see page 4827)	Updates the display to reflect the current value of the Strings (see page 4863) property.
	RestoreCurrentRow (see page 4828)	Undoes all edits to the current row.

50.1.2.4.1 TValueListEditor.AdjustColWidths Method

Pascal

```
procedure AdjustColWidths; virtual;
```

C++

```
virtual __fastcall AdjustColWidths();
```

Description

This is AdjustColWidths, a member of class TValueListEditor.

50.1.2.4.2 TValueListEditor.AdjustRowCount Method

Pascal

```
procedure AdjustRowCount; virtual;
```

C++

```
virtual __fastcall AdjustRowCount();
```

Description

This is AdjustRowCount, a member of class TValueListEditor.

50.1.2.4.3 TValueListEditor.CanEditModify Method

Indicates whether the current cell in the grid can be edited.

Pascal

```
function CanEditModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditModify();
```

Description

Grid components use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify returns true unless the inplace editor has disallowed editing for the cell.

See Also

CanEditShow

InplaceEditor

50.1.2.4.4 TValueListEditor.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; override;
```

C++

```
virtual __fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change. The change can result from setting the ColWidths property, from setting the DefaultColWidth property, from moving one of the columns, or from resizing the columns with the mouse.

ColWidthsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new column size and positions of the columns.

See Also

ColWidths

DefaultColWidth

InplaceEditor

ScrollBars

50.1.2.4.5 TValueListEditor.CreateEditor Method

Creates the inplace editor control.

Pascal

```
function CreateEditor: TInplaceEdit; override;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

Grid controls call CreateEditor internally the first time the grid goes into edit mode. CreateEditor is exposed as a protected method so that descendants of TCustomGrid can substitute a specialized descendant of TInplaceEdit to handle cell editing.

See Also

EditorMode

InplaceEditor

ShowEditor

50.1.2.4.6 TValueListEditor.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

50.1.2.4.7 TValueListEditor.DeleteRow Method

Deletes a row in the value list editor and a corresponding string from the associated TStrings object.

Pascal

```
procedure DeleteRow(ARow: Integer); override;
```

C++

```
virtual __fastcall DeleteRow(int ARow);
```

Description

Call DeleteRow to remove the row specified by the ARow parameter. Delete (see page 3653) removes the corresponding string from the TStrings object being edited and updates the display.

ARow must be the index of a row that displays a name/value pair. That is, it can't be less than 0 or greater than the number of rows. If DisplayOptions (see page 4841) includes doColumnTitles, ARow can't be 0, because that is the title row, not a name/value pair in the associated TStrings object.

Note: DeleteRow deletes the specified row regardless of the value of KeyOptions (see page 4851). The keyDelete member of KeyOptions (see page 4851) controls only whether the user can delete a row by pressing the Delete (see page 3653) key.

See Also

Cells (see page 4836)

Strings (see page 4863)

KeyOptions (see page 4851)

DisplayOptions (see page 4841)

50.1.2.4.8 TValueListEditor.DoExit Method

Hides the inplace editor when selection moves to another control in a form.

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

After calling the inherited method, which generates an OnExit event, DoExit hides the inplace editor unless the Options property includes goAlwaysShowEditor.

See Also

HideEditor

Options

OnExit

50.1.2.4.9 TValueListEditor.DoOnValidate Method

Pascal

```
procedure DoOnValidate; virtual;
```

C++

```
virtual __fastcall DoOnValidate();
```

Description

This is DoOnValidate, a member of class TValueListEditor.

50.1.2.4.10 TValueListEditor.DrawCell Method

Generates an OnDrawCell event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell

are drawn after DrawCell. Descendants of TCustomDrawGrid can override this method to draw the contents of the cell or to block the OnDrawCell event.

See Also

DefaultDrawing

FixedCols

FixedRows

OnDrawCell

50.1.2.4.11 TValueListEditor.EditListGetItems Method

Pascal

```
procedure EditListGetItems(ACol: Integer; ARow: Integer; Items: TStrings);
```

C++

```
__fastcall EditListGetItems(int ACol, int ARow, TStrings Items);
```

Description

This is EditListGetItems, a member of class TValueListEditor.

50.1.2.4.12 TValueListEditor.FindRow Method

Locates the row that corresponds to a specified key name.

Pascal

```
function FindRow(const KeyName: string; var Row: Integer): Boolean;
```

C++

```
__fastcall Boolean FindRow(const UnicodeString KeyName, int Row);
```

Description

Call FindRow to locate the row in the value list editor that has the value specified by KeyName in the first column. This corresponds to a name/value pair in the string list where the name part is KeyName.

If a row in the value list editor has the specified key name, FindRow returns true and the Row parameter returns the index of the first row found with the specified name. If the specified name does not correspond to any of the rows in the value list editor, FindRow returns false.

See Also

Keys (see page 4851)

50.1.2.4.13 TValueListEditor.GetCell Method

Pascal

```
function GetCell(ACol: Integer; ARow: Integer): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetCell(int ACol, int ARow);
```

Description

This is GetCell, a member of class TValueListEditor.

50.1.2.4.14 TValueListEditor.GetColCount Method

Pascal

```
function GetColCount: Integer;
```

C++

```
__fastcall int GetColCount();
```

Description

This is GetColCount, a member of class TValueListEditor.

50.1.2.4.15 TValueListEditor.GetEditLimit Method

Provides the interface for a method that limits the number of characters the user can type into cells.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The inplace editor calls GetEditLimit to determine if a limit should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits. Descendants of TCustomGrid override this method to limit the text for certain cells.

See Also

InplaceEditor

50.1.2.4.16 TValueListEditor.GetEditMask Method

Generates an OnGetEditMask event.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns the Value parameter set by the OnGetEditMask event handler. If there is no OnGetEditMask event handler, GetEditMask returns an empty string, indicating that there is no edit mask.

See Also

OnGetEditMask

50.1.2.4.17 TValueListEditor.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; override;
```

C++

```
virtual __fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomGrid, GetEditStyle always returns esSimple, indicating that there is no ellipsis button or drop-down arrow. Descendant classes can override this methods if they provide a way to add pick lists or ellipsis buttons to cells.

50.1.2.4.18 TValueListEditor.GetEditText Method

Pascal

```
function GetEditText(ACol: Integer; ARow: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditText(int ACol, int ARow);
```

Description

This is GetEditText, a member of class TValueListEditor.

50.1.2.4.19 TValueListEditor.GetPickList Method

Pascal

```
function GetPickList(Values: TStrings; ClearFirst: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean GetPickList(TStrings Values, Boolean ClearFirst = True);
```

Description

This is GetPickList, a member of class TValueListEditor.

50.1.2.4.20 TValueListEditor.GetRowCount Method

Pascal

```
function GetRowCount: Integer;
```

C++

```
__fastcall int GetRowCount();
```

Description

This is GetRowCount, a member of class TValueListEditor.

50.1.2.4.21 TValueListEditor.InsertRow Method

Inserts a row in the value list editor and a corresponding string into the associated TStrings object.

Pascal

```
function InsertRow(const KeyName: string; const Value: string; Append: Boolean): Integer;
```

C++

```
__fastcall int InsertRow(const UnicodeString KeyName, const UnicodeString Value, Boolean
Append);
```

Description

Call InsertRow to add a name/value pair to the value list editor and the TStrings object it is editing.

InsertRow adds a string of the form

`KeyName=Value`

to the Strings (see page 4863) property and updates the display to reflect the change.

If the current row of the value list editor is empty (has no name or value portion), InsertRow assigns the new KeyName and Value to the current row. Otherwise, if the Append (see page 3624) parameter is true, the new string appears after the currently selected row, while if Append (see page 3624) is false, the new string appears before the currently selected row.

Note: InsertRow inserts the specified row regardless of the value of KeyOptions (see page 4851). The keyAdd member of KeyOptions (see page 4851) controls only whether the user can add new rows by pressing the Insert (see page 3698) key or navigating past the last row.

See Also

Cells (see page 4836)

Strings (see page 4863)

KeyOptions (see page 4851)

DisplayOptions (see page 4841)

50.1.2.4.22 TValueListEditor.IsEmptyRow Method

Pascal

```
function IsEmptyRow: Boolean;
```

C++

```
__fastcall Boolean IsEmptyRow();
```

Description

This is IsEmptyRow, a member of class TValueListEditor.

50.1.2.4.23 TValueListEditor.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After generating an OnKeyDown event, KeyDown calls CanGridAcceptKey to screen out problematic key combinations. KeyDown moves to the appropriate cell in response to navigational key combinations, or sets the EditorMode property to true when the user presses F2.

See Also

CanGridAcceptKey

EditorMode

KeyPress

OnKeyDown

50.1.2.4.24 TValueListEditor.MouseDown Method

Provides special processing when the user presses a mouse button.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown hides the editor window so that it can move selection to the cell that receives the mouse click. If the mouse lands in the hot region for beginning a move or resize of a row or column, MouseDown starts the move or resize, and provides visual feedback by changing the cursor. If the editor window was hidden, it is brought back up in its new location. Finally, MouseDown generates an OnMouseDown event.

See Also

HideEditor

InplaceEditor

MouseMove

MouseUp

ShowEditor

OnMouseDown

50.1.2.4.25 TValueListEditor.Refresh Method

Updates the display to reflect the current value of the Strings (see page 4863) property.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to ensure that the value list editor accurately reflects the value of the Strings (see page 4863) property. For example, call Refresh after programmatically editing the value of the Strings (see page 4863) property.

See Also

RestoreCurrentRow (see page 4828)

Repaint

Strings (see page 4863)

50.1.2.4.26 TValueListEditor.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize

50.1.2.4.27 TValueListEditor.RestoreCurrentRow Method

Undoes all edits to the current row.

Pascal

```
function RestoreCurrentRow: Boolean;
```

C++

```
__fastcall Boolean RestoreCurrentRow();
```

Description

Call RestoreCurrentRow to back out edits to the current cell in the value list editor. RestoreCurrentRow backs out edits to the current cell if it is still in edit mode, or deletes a newly inserted row.

The user can accomplish the same effect by pressing the Escape key.

See Also

Cells (see page 4836)

InsertRow (see page 4825)

50.1.2.4.28 TValueListEditor.RowMoved Method

Generates an OnRowMoved event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnRowMoved event.

See Also

OnRowMoved

50.1.2.4.29 TValueListEditor.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an OnSelectCell event. This method then returns true, unless the OnSelectCell event handler indicates that the cell should not be selected. Descendants of TCustomDrawGrid can override this method to check the properties of the information displayed in the cell or block the OnSelectCell event.

See Also

OnSelectCell

50.1.2.4.30 TValueListEditor.SetCell Method

Pascal

```
procedure SetCell(ACol: Integer; ARow: Integer; const Value: string); virtual;
```

C++

```
virtual __fastcall SetCell(int ACol, int ARow, const UnicodeString Value);
```

Description

This is SetCell, a member of class TValueListEditor.

50.1.2.4.31 TValueListEditor.SetEditText Method

Pascal

```
procedure SetEditText(ACol: Integer; ARow: Integer; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(int ACol, int ARow, const UnicodeString Value);
```

Description

This is SetEditText, a member of class TValueListEditor.

50.1.2.4.32 TValueListEditor.SetStrings Method

Pascal

```
procedure SetStrings(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetStrings(const TStrings Value);
```

Description

This is SetStrings, a member of class TValueListEditor.

50.1.2.4.33 TValueListEditor.StringsChanging Method

Pascal

```
procedure StringsChanging; dynamic;
```

C++

```
__fastcall StringsChanging();
```

Description

This is StringsChanging, a member of class TValueListEditor.

50.1.2.4.34 TValueListEditor.TitleCaptionsStored Method

Pascal

```
function TitleCaptionsStored: Boolean;
```

C++

```
__fastcall Boolean TitleCaptionsStored();
```

Description

This is TitleCaptionsStored, a member of class TValueListEditor.

50.1.2.5 TValueListEditor Properties

The properties of the TValueListEditor class are listed here.

Protected Properties

	Name	Description
	EditList (see page 4846)	This is EditList, a member of class TValueListEditor.

Public Properties

	Name	Description
	Cells (see page 4836)	Lists the strings for each cell in the editor.
R	ColCount (see page 4836)	Indicates the number of columns in the value list editor.
	ItemProps (see page 4851)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 4851)	Lists the strings for each cell in the name (key) column.
R	RowCount (see page 4861)	Indicates the number of rows in the value list editor.
	Values (see page 4865)	Lists the strings for each value from the value column.
	VisibleColCount (see page 4867)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 4867)	Indicates the number of scrollable rows visible in the grid.

Published Properties

	Name	Description
	Align (see page 4831)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4835)	Specifies how the control is anchored to its parent.

	BiDiMode (see page 4835)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4836)	Determines whether a single line border is drawn around the grid.
	Color (see page 4837)	Specifies the background color of the control.
	Constraints (see page 4838)	Specifies the size constraints for the control.
	Ctl3D (see page 4838)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 4839)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 4840)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 4841)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 4841)	Controls aspects of the value list editor's appearance.
	DoubleBuffered (see page 4842)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4843)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4844)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4844)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 4846)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 4847)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 4847)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 4847)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 4848)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 4849)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	KeyOptions (see page 4851)	Controls the way the value list editor responds to user keystrokes.
	Options (see page 4852)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 4853)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4854)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4854)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4855)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4855)	Determines where a control looks for its font information.
	ParentShowHint (see page 4855)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4856)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 4861)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 4862)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 4863)	Provides access to the string list that is being edited.
	TabOrder (see page 4864)	Indicates the position of the control in its parent's tab order.
	TitleCaptions (see page 4864)	Specifies the captions that label the columns of the value list editor.
	Visible (see page 4865)	Determines whether the component appears onscreen.

50.1.2.5.1 TValueListEditor.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
```

```
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

50.1.2.5.2 TValueListEditor.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

50.1.2.5.3 TValueListEditor.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data

aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

50.1.2.5.4 TValueListEditor.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

50.1.2.5.5 TValueListEditor.Cells Property

Lists the strings for each cell in the editor.

Pascal

```
property Cells [ACol, ARow: Integer]: string;
```

C++

```
__property UnicodeString Cells[int ACol, ARow];
```

Description

Use Cells to access the string in a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. ACol can be 0 (for the name column) or 1 (for the value column). ARow can range from 0 (for the title row) to the number of strings in the associated TStrings descendant.

To access the strings in the name column, use the Keys (see page 4851) property instead. To access the strings in the value column, use the Values (see page 4865) property instead. To access the underlying string list, use the Strings (see page 4863) property.

See Also

Keys (see page 4851)

Values (see page 4865)

Strings (see page 4863)

50.1.2.5.6 TValueListEditor.ColCount Property

Indicates the number of columns in the value list editor.

Pascal

```
property ColCount;
```

C++

```
__property ColCount;
```

Description

TValueListEditor (see page 4814) redeclares the ColCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of ColCount is always 2 plus the number of fixed columns, because the value list editor is designed to display only the name and value of the strings in the associated string list.

See Also

Cells (see page 4836)

RowCount (see page 4861)

FixedCols

50.1.2.5.7 TValueListEditor.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

50.1.2.5.8 TValueListEditor.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

50.1.2.5.9 TValueListEditor.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

50.1.2.5.10 TValueListEditor.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth;
```

C++

```
__property DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount property, they are created with a width of DefaultColWidth.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}
```

```
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColCount

ColWidths

DefaultRowHeight

50.1.2.5.11 TValueListEditor.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing;
```

C++

```
__property DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell

OnDrawCell

50.1.2.5.12 TValueListEditor.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight;
```

C++

```
__property DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth

RowCount

RowHeights

50.1.2.5.13 TValueListEditor.DisplayOptions Property

Controls aspects of the value list editor's appearance.

Pascal

```
property DisplayOptions: TDisplayOptions;
```

C++

```
__property TDisplayOptions DisplayOptions;
```

Description

Use DisplayOptions to control whether titles appear at the top of the columns in the value list editor and how the control responds when it is resized.

See Also

KeyOptions (see page 4851)

Options

TitleCaptions (see page 4864)

50.1.2.5.14 TValueListEditor.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

50.1.2.5.15 TValueListEditor.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

50.1.2.5.16 TValueListEditor.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

50.1.2.5.17 TValueListEditor.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == dynamic_cast<TObject *>(TreeView1));
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property must be set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

50.1.2.5.18 TValueListEditor.DropDownRows Property

Limits the length of the drop-down list that appears for values that use a pick list.

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

Use DropDownRows to indicate the maximum height of the drop-down list that appears when the user clicks the drop-down button on values that have a pick list. DropDownRows is an upper limit: if the pick list has fewer entries that DropDownRows, the drop-down list includes only a line for every item in the list.

DropDownRows is ignored for values that do not have a pick list. To provide a value in the value list editor with a drop-down button and pick list, you do either of the following:

For the appropriate entry in the ItemProps (see page 4851) property, set the EditStyle property to esPickList and set the PickList property to the list of items.

Write (see page 3789) an OnGetPickList (see page 4889) event handler that supplies the items in the list.

See Also

OnGetPickList (see page 4889)

Values (see page 4865)

ItemProps (see page 4851)

50.1.2.5.19 TValueListEditor.EditList Property

Pascal

```
property EditList: TInplaceEditList;
```

C++

```
__property TInplaceEditList EditList;
```

Description

This is EditList, a member of class TValueListEditor.

50.1.2.5.20 TValueListEditor.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

50.1.2.5.21 TValueListEditor.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor;
```

C++

```
__property FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color

FixedCols

FixedRows

50.1.2.5.22 TValueListEditor.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols;
```

C++

```
__property FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount - 1.

See Also

ColCount

DefaultDrawing

FixedColor

FixedRows

50.1.2.5.23 TValueListEditor.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
```

```
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

50.1.2.5.24 TValueListEditor.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth;
```

C++

```
__property GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
```

```
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols

FixedRows

Options

50.1.2.5.25 TValueListEditor.ItemProps Property

Specifies various constraints on the way users can enter a value for each item in the list.

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

Use ItemProps to indicate, for each item:

- Whether the item is read only.
- The maximum length of strings the user can enter.
- An edit mask that limits the strings the user can enter as a value.
- Whether the item lets users drop down a pick list or launch a dialog to enter values.
- A fixed set of values from which users select (a pick list).

See Also

OnGetPickList (see page 4889)

Values (see page 4865)

50.1.2.5.26 TValueListEditor.KeyOptions Property

Controls the way the value list editor responds to user keystrokes.

Pascal

```
property KeyOptions: TKeyOptions;
```

C++

```
__property TKeyOptions KeyOptions;
```

Description

Use KeyOptions to control how users can edit name/value pairs using the value list editor.

See Also

Keys (see page 4851)

Strings (see page 4863)

50.1.2.5.27 TValueListEditor.Keys Property

Lists the strings for each cell in the name (key) column.

Pascal

```
property Keys [Index: Integer]: string;
```

C++

```
__property UnicodeString Keys[int Index];
```

Description

Use Keys to access the name (or key) value for a particular row. That is, Keys assumes the strings in the associated string list have the form Name=Value, and returns the Name portion.

Index identifies the Row of the value list editor. It takes a value from 0 to the number of rows. When DisplayOptions (see page 4841) includes doColumnTitles, the value of Keys is the title of the first column when Index is 0, and is the Name portion of the string when Index is greater than 0. When DisplayOptions (see page 4841) does not include doColumnTitles, the value of Keys is the Name portion of the string in the specified row.

To access the strings in the value column, use the Values (see page 4865) property instead. To access the underlying string list, use the Strings (see page 4863) property.

See Also

Cells (see page 4836)

Values (see page 4865)

Strings (see page 4863)

DisplayOptions (see page 4841)

Names

50.1.2.5.28 TValueListEditor.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
```

```
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColWidths

EditorMode

GridLineWidth

RowHeights

Selection

TabStops

50.1.2.5.29 TValueListEditor.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

50.1.2.5.30 TValueListEditor.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

50.1.2.5.31 TValueListEditor.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

50.1.2.5.32 TValueListEditor.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

50.1.2.5.33 TValueListEditor.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

50.1.2.5.34 TValueListEditor.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

50.1.2.5.35 TValueListEditor.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
```

```
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

50.1.2.5.36 TValueListEditor.RowCount Property

Indicates the number of rows in the value list editor.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

TValueListEditor (see page 4814) redeclares the RowCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of RowCount is the number of strings in the Strings (see page 4863) property plus the number of fixed rows (typically one, for the column titles).

See Also

Strings (see page 4863)

ColCount (see page 4836)

FixedRows

50.1.2.5.37 TValueListEditor.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars;
```

C++

```
__property ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight

ClientWidth

50.1.2.5.38 TValueListEditor.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
```

```
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

50.1.2.5.39 TValueListEditor.Strings Property

Provides access to the string list that is being edited.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Use Strings to access the string list that the value list editor displays. Strings should be a TStrings descendant that contains strings of the form Name=Value.

At design time, add values to the Strings property using the string list editor. At runtime, you can use the methods of TStrings to

add strings to the list one by one, or assign an existing TStrings descendant to the Strings property. When you set Strings at runtime, TValueListEditor (see page 4814) copies the strings from the TStrings object you assign: it does not take ownership of the object itself.

When Strings is assigned, the value list editor displays the name portion (TStrings . ::Names) of the strings in its key column and the value portion (TStrings . ::Values (see page 4865)) in the value column.

See Also

Names

Values

KeyOptions (see page 4851)

50.1.2.5.40 TValueListEditor.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

50.1.2.5.41 TValueListEditor.TitleCaptions Property

Specifies the captions that label the columns of the value list editor.

Pascal

```
property TitleCaptions: TStrings;
```

C++

```
__property TStrings TitleCaptions;
```

Description

Use TitleCaptions to provide titles for the columns of the value list editor. TitleCaptions is a TStrings descendant that contains two strings: the first string labels the name (key) column, and the second string labels the value column. If TitleCaptions contains less than two strings, missing captions are treated as empty strings. If TitleCaptions contains more than two strings, all strings except the first two are ignored.

Note: TitleCaptions is only used when the DisplayOptions (see page 4841) property includes doColumnTitles.

See Also

DisplayOptions (see page 4841)

50.1.2.5.42 TValueListEditor.Values Property

Lists the strings for each value from the value column.

Pascal

```
property Values [const Key: string]: string;
```

C++

```
__property UnicodeString Values[UnicodeString const Key];
```

Description

Use Values to access the value associated with a specified name from the first (key) column. Key specifies a string that appears in the first column of the value list editor. Values is the corresponding string from the second (value) column.

To access the strings in the key column, use the Keys (see page 4851) property instead. To access the underlying string list, use the Strings (see page 4863) property.

Note: Setting Values with Key set to a name that is not in the list adds a new string to Strings (see page 4863) and a new row to the value list editor.

See Also

Cells (see page 4836)

Keys (see page 4851)

Strings (see page 4863)

Values

50.1.2.5.43 TValueListEditor.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false,

the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

50.1.2.5.44 TValueListEditor.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount;
```

C++

```
__property VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount

FixedCols

VisibleRowCount

50.1.2.5.45 TValueListEditor.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount;
```

C++

```
__property VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows

RowCount

VisibleColCount

50.1.2.6 TValueListEditor Events

The events of the TValueListEditor class are listed here.

Published Events

	Name	Description
	OnClick (see page 4868)	Occurs when the user clicks the control.
	OnContextPopup (see page 4870)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4871)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4871)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4874)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 4878)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 4879)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 4880)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4880)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4881)	Occurs when a control receives the input focus.
	OnExit (see page 4883)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 4886)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 4887)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 4889)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 4889)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4893)	Occurs when a key is pressed.
	OnKeyUp (see page 4894)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4896)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4897)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4899)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4899)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4899)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4902)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 4904)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4904)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 4905)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 4906)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 4908)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 4908)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4909)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStringsChange (see page 4909)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 4910)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 4910)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 4910)	Occurs when focus shifts away from a cell in the value list editor.

50.1.2.6.1 TValueListEditor.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 1970)

Default (see page 1974)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

50.1.2.6.2 TValueListEditor.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

50.1.2.6.3 TValueListEditor.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

50.1.2.6.4 TValueListEditor.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
```

```
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

50.1.2.6.5 TValueListEditor.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
```

```
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

50.1.2.6.6 TValueListEditor.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell;
```

C++

```
__property OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas property. The Rect parameter indicates the location of the cell on the canvas. The Col and Row parameters indicate the column and row indexes of the cell that should be drawn. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
```

```
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

Canvas

DefaultDrawing

50.1.2.6.7 TValueListEditor.OnEditButtonClick Event

Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

Write (see page 3789) an OnEditButtonClick to display a dialog that lets users edit a value in the Values (see page 4865) column. This event only occurs for items for which the ItemProps (see page 4851) property has an EditStyle of esEllipsis.

In the event handler, you can use the Row property to identify the row that the user is editing. The event handler (or the dialog that the event handler displays) can use the Keys (see page 4851) property to identify the current key value, and use that to get and set the corresponding entry in the Values (see page 4865) property that is being edited.

See Also

Values (see page 4865)

ItemProps (see page 4851)

EditStyle (see page 4812)

Row

Keys (see page 4851)

50.1.2.6.8 TValueListEditor.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

50.1.2.6.9 TValueListEditor.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

50.1.2.6.10 TValueListEditor.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

50.1.2.6.11 TValueListEditor.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
```

```
  to contain the empty string, to use the Courier font and to
  use a particular mask, corresponding to IP address coding convetions.
  The mask edit boxes display the IP address, the subnet mask
  and the subnet address correspondingly. When the user exits the
  subnet mask edit box, the subnet address is automatically calculated
  using bit-wise AND operations and displayed in the third mask edit box.
  An additional test is made to assure that the values in the IP address
  and in the subnet mask are in the interval from 0 to 255,
  as this validation test cannot be accomplished
  using the edit mask property alone.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { initialize the maskedit boxes to contain the empty string }
    MaskEdit1.Text := '';
    MaskEdit2.Text := '';
    MaskEdit3.Text := '';

    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
```

```
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

50.1.2.6.12 TValueListEditor.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask;
```

C++

```
__property OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the cell specified by the ACol and

ARow parameters. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options property includes goEditing.

C++ Examples:

```
/*
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
}

void __fastcall TForm1::StringGrid1GetEditMask(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  if (StringGrid1->Cells[ACol][0] == "Phone Number")
    Value =  "!\(999\)000-0000;1";
}
```

Delphi Examples:

```
{
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[2, 0]:= 'Phone Number';
end;

procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  if StringGrid1.Cells[ACol, 0] = 'Phone Number' then
    Value :=  '!\(999\)000-0000;1';
end;
```

See Also

Options

EditMask

50.1.2.6.13 TValueListEditor.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText;
```

C++

```
__property OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the cell specified by the ACol and ARow parameters. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell event handler.

OnGetEditText does not occur unless the Options property includes goEditing.

C++ Examples:

```
/*
The following code lowercases the text for a string grid
cell when the user begins to edit it.
*/
void __fastcall TForm1::StringGrid1GetEditText(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  Value = AnsiLowerCase(Value);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code lowercases the text for a string grid
cell when the user begins to edit it.
}
procedure TForm1.StringGrid1GetEditText(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  Value := AnsiLowerCase(Value);
end;
```

See Also

OnDrawCell

OnSetEditText

50.1.2.6.14 TValueListEditor.OnGetPickList Event

Supplies a pick list for values associated with a particular key.

Pascal

```
property OnGetPickList: TGetPickListEvent;
```

C++

```
__property TGetPickListEvent OnGetPickList;
```

Description

Write (see page 3789) an OnGetPickList event handler to provide a drop-down list of values for specific names from the first (key) column. OnGetPickList is called when the value column gets focus and the name (key) column contains the string specified by KeyName.

If the ItemProps (see page 4851) property has an EditStyle of esPickList for this item, The Values (see page 4865) parameter of the event handler is initialized to the value of its PickList property for this item. Otherwise, Values (see page 4865) is initialized to an empty list. In either case, the event handler can change the list of values that appear in the drop-down pick list.

If the OnGetPickList event handler exits with a non-empty string list as the value of the Values (see page 4865) parameter, a drop-down arrow appears in the value column. The user can click this arrow button to display the pick list you supplied as the value of Values (see page 4865). This makes the Value column behave like a drop-down combo box: the user can select items from the list or type in an entirely different value.

See Also

Keys (see page 4851)

Values (see page 4865)

DropDownRows (see page 4846)

ItemProps (see page 4851)

PickList (see page 4813)

EditStyle (see page 4812)

50.1.2.6.15 TValueListEditor.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse

buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
```

```
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;
```

```
procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
```

```
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

50.1.2.6.16 TValueListEditor.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/
```

```
void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

50.1.2.6.17 TValueListEditor.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
```

```
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

50.1.2.6.18 TValueListEditor.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

50.1.2.6.19 TValueListEditor.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

50.1.2.6.20 TValueListEditor.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

50.1.2.6.21 TValueListEditor.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

50.1.2.6.22 TValueListEditor.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

50.1.2.6.23 TValueListEditor.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

50.1.2.6.24 TValueListEditor.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

50.1.2.6.25 TValueListEditor.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

50.1.2.6.26 TValueListEditor.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved;
```

C++

```
__property OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. OnRowMoved does not occur unless the Options property includes goRowMoving.

C++ Examples:

```
/*
The following code displays the number of rows a row was
moved in the first cell of the row.  To move rows in a
TStringGrid, goRowMoving must be included in the Options property.
*/
void __fastcall TForm1::StringGrid1RowMoved(TObject *Sender, int FromIndex, int ToIndex)

{
  int delta = (FromIndex > ToIndex) ? (FromIndex-ToIndex) : (ToIndex-FromIndex);
  StringGrid1->Cells[0][ToIndex] = IntToStr(delta);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
```

```
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code displays the number of rows a row was
moved in a label.  To move rows in a TStringGrid, goRowMoving
must be included in the Options property.
}
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
  Label1.Caption := IntToStr(Abs(FromIndex-ToIndex));
end;
```

See Also

OnColumnMoved

Options

50.1.2.6.27 TValueListEditor.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell;
```

C++

```
__property OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected. The Col and Row parameters indicate the column and row indexes of the cell that is about to be selected. Set the CanSelect parameter to false to prevent the cell being selected.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
```

```
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code ensures that the user cannot select a cell
unless it is empty.
*/
void __fastcall TForm1::StringGrid1SelectCell(TObject *Sender, int ACol, int ARow,
          bool &CanSelect)
{
  CanSelect = (StringGrid1->Cells[ACol][ARow]== "");
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code ensures that the user cannot select a cell
unless it is empty.
}
procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Longint; var CanSelect:
Boolean);
begin
  CanSelect := (StringGrid1.Cells[ACol, ARow]='')
end;
```

See Also

Col

Row

TabStops

50.1.2.6.28 TValueListEditor.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText;
```

C++

```
__property OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell event handler. OnSetEditText occurs every time the user changes the text.

The Value parameter is the new value for the cell specified by the ACol and ARow parameters.

OnSetEditText does not occur unless the Options property includes goEditing.

See Also

OnDrawCell

OnGetEditText

Options

50.1.2.6.29 TValueListEditor.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

50.1.2.6.30 TValueListEditor.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

50.1.2.6.31 TValueListEditor.OnStringsChange Event

Occurs immediately after the names or values in the string list change.

Pascal

```
property OnStringsChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChange;
```

Description

Write (see page 3789) an OnStringsChange event handler to respond when the strings in the list change. For example, you can use the OnStringsChange event to update your display to reflect the new values of the strings in the value list editor.

OnStringsChange occurs after the change is complete. The properties of the value list editor reflect the new values of the strings.

See Also

OnStringsChanging (see page 4910)

Strings (see page 4863)

OnSetEditText

50.1.2.6.32 TValueListEditor.OnStringsChanging Event

Occurs immediately before the names or values in the string list change.

Pascal

```
property OnStringsChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChanging;
```

Description

Write (see page 3789) an OnStringsChanging event handler to prepare for changes in the list of strings. For example, you can use the OnStringsChanging event to disable repaints until the OnStringsChange (see page 4909) event when the list has finished changing..

OnStringsChanging occurs before the change occurs. The properties of the value list editor reflect the old values of the strings.

See Also

OnStringsChange (see page 4909)

Strings (see page 4863)

OnSetEditText

50.1.2.6.33 TValueListEditor.OnTopLeftChanged Event

Occurs immediately after the TopRow property or the LeftCol property changes.

Pascal

```
property OnTopLeftChanged;
```

C++

```
__property OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol

TopRow

50.1.2.6.34 TValueListEditor.OnValidate Event

Occurs when focus shifts away from a cell in the value list editor.

Pascal

```
property OnValidate: TOnValidateEvent;
```

C++

```
__property TOnValidateEvent OnValidate;
```

Description

Write (see page 3789) an OnValidate event handler to validate any edits the user enters in a cell before focus leaves it. OnValidate gives applications an opportunity to provide more validation than the EditMask property of the corresponding TItemProp (see page 4808) object can supply.

OnValidate only occurs if the user edited the value of the cell that is about to lose focus. The OnValidate event handler can verify the value the user supplied, and if it is not acceptable, raise an exception.

See Also

Keys (see page 4851)

Values (see page 4865)

ItemProps (see page 4851)

EditMask (see page 4811)

50.1.3 TValueListStrings Class

Class Hierarchy

Pascal

```
TValueListStrings = class(TStringList);
```

C++

```
class TValueListStrings : public TStringList;
```

File

ValEdit

Description

This is class ValEdit.TValueListStrings.

50.1.3.1 TValueListStrings Members

The following tables list the members exposed by TValueListStrings.

Protected Methods

	Name	Description
	Changed (see page 4916)	This is Changed, a member of class TValueListStrings.
	Changing (see page 4916)	This is Changing, a member of class TValueListStrings.
	FindItemProp (see page 4918)	This is FindItemProp, a member of class TValueListStrings.
	InsertItem (see page 4918)	This is InsertItem, a member of class TValueListStrings.
	Put (see page 4919)	This is Put, a member of class TValueListStrings.

Protected Properties

	Name	Description
	UpdateCount (see page 4919)	Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Public Methods

	Name	Description
	Assign (see page 4913)	Sets the strings in the list, and possibly associated objects, from a source object.
	Clear (see page 4916)	Deletes all the strings from the list.
	Create (see page 4912)	This is Create, a member of class TValueListStrings.
	CustomSort (see page 4916)	Sorts the strings in the list in a customized order.
	Delete (see page 4917)	Removes the string specified by the Index parameter.
	Destroy (see page 4912)	Destroys an instance of TStringList.
	Exchange (see page 4917)	Swaps the position of two strings in the list.
	KeyIsValid (see page 4918)	This is KeyIsValid, a member of class TValueListStrings.

Public Properties

	Name	Description
	ItemProps (see page 4919)	This is ItemProps, a member of class TValueListStrings.

50.1.3.2 TValueListStrings.Create Constructor

Pascal

```
constructor Create(AEditor: TValueListEditor); reintroduce;
```

C++

```
__fastcall TValueListStrings(TValueListEditor AEditor);
```

Description

This is Create, a member of class TValueListStrings.

50.1.3.3 TValueListStrings.Destroy Destructor

Destroys an instance of TStringList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListStrings();
```

Description

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

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

System::TObject::Free (see page 3382)

50.1.3.4 TValueListStrings Methods

The methods of the TValueListStrings class are listed here.

Protected Methods

	Name	Description
	Changed (see page 4916)	This is Changed, a member of class TValueListStrings.
	Changing (see page 4916)	This is Changing, a member of class TValueListStrings.
	FindItemProp (see page 4918)	This is FindItemProp, a member of class TValueListStrings.
	InsertItem (see page 4918)	This is InsertItem, a member of class TValueListStrings.
	Put (see page 4919)	This is Put, a member of class TValueListStrings.

Public Methods

	Name	Description
	Assign (see page 4913)	Sets the strings in the list, and possibly associated objects, from a source object.
	Clear (see page 4916)	Deletes all the strings from the list.
	CustomSort (see page 4916)	Sorts the strings in the list in a customized order.
	Delete (see page 4917)	Removes the string specified by the Index parameter.
	Exchange (see page 4917)	Swaps the position of two strings in the list.
	KeyIsValid (see page 4918)	This is KeyIsValid, a member of class TValueListStrings.

50.1.3.4.1 TValueListStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TStrings object from another object. If Source is of type TStrings, the list is set to the list of the source TStrings object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TStrings, the inherited Assign will set the value of the list from any object that supports TStrings in its AssignTo method.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
```

```
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
    TabControl1Change(Sender);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TStrings> StringList(new TStringList());
  StringList->Add("This example uses A string List. ");
  StringList->Add("It is the easiest way to add strings");
  StringList->Add("to a combobox's list of strings. ");
  StringList->Add("Always remember TStrings is abstract,");
  StringList->Add("So create a TStringList instead.");
  ComboBox1->Width = 210;
  ComboBox1->Items->Assign(StringList.get());
  ComboBox1->ItemIndex = 0;
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
```

```
    TabControl1Change(Sender);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  StringList: TStringList;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('This example uses A string List.');
      Add('It is the easiest way to add strings');
      Add('to a combobox''s list of strings.');
      Add('Always remember TStrings.Create method');
      Add('is abstract; So use TStringList.Create');
      Add('method instead.');
    end;
    with ComboBox1 do begin
      Width := 210;
      Items.Assign(StringList);
      ItemIndex := 0;
    end;
  finally
    StringList.free;
  end;
end;
```

See Also

AddStrings

Objects

Strings

AssignTo

50.1.3.4.2 TValueListStrings.Changed Method

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TValueListStrings.

50.1.3.4.3 TValueListStrings.Changing Method

Pascal

```
procedure Changing; override;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TValueListStrings.

50.1.3.4.4 TValueListStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete

Objects

Strings

50.1.3.4.5 TValueListStrings.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TStringListSortCompare); override;
```

C++

```
virtual __fastcall CustomSort(TStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The List parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort method.

See Also

Sort

Strings

Sorted

50.1.3.4.6 TValueListStrings.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear

IndexOf

InsertObject

Objects

Strings

50.1.3.4.7 TValueListStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move

Objects

Sorted

Strings

50.1.3.4.8 TValueListStrings.FindItemProp Method

Pascal

```
function FindItemProp(const KeyOrIndex: Variant; Create: Boolean = False): TItemProp;
```

C++

```
__fastcall TItemProp FindItemProp(const Variant KeyOrIndex, Boolean Create = False);
```

Description

This is FindItemProp, a member of class TValueListStrings.

50.1.3.4.9 TValueListStrings.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: string; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertItem(int Index, const UnicodeString S, TObject * AObject);
```

Description

This is InsertItem, a member of class TValueListStrings.

50.1.3.4.10 TValueListStrings.KeyIsValid Method

Pascal

```
function KeyIsValid(const Key: string; RaiseError: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean KeyIsValid(const UnicodeString Key, Boolean RaiseError = True);
```

Description

This is KeyIsValid, a member of class TValueListStrings.

50.1.3.4.11 TValueListStrings.Put Method

Pascal

```
procedure Put(Index: Integer; const S: String); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString S);
```

Description

This is Put, a member of class TValueListStrings.

50.1.3.5 TValueListStrings Properties

The properties of the TValueListStrings class are listed here.

Protected Properties

	Name	Description
	UpdateCount (see page 4919)	Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Public Properties

	Name	Description
	ItemProps (see page 4919)	This is ItemProps, a member of class TValueListStrings.

50.1.3.5.1 TValueListStrings.ItemProps Property

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

This is ItemProps, a member of class TValueListStrings.

50.1.3.5.2 TValueListStrings.UpdateCount Property

Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Pascal

```
property UpdateCount;
```

C++

```
__property UpdateCount;
```

Description

TStrings uses UpdateCount to keep track of calls to the BeginUpdate and EndUpdate methods. Every time a call is made to BeginUpdate, TStrings increments the value of UpdateCount. Every call to EndUpdate causes TStrings to decrement UpdateCount.

When UpdateCount changes from 0 to 1, TStrings calls the SetUpdateState method with a parameter of true. When UpdateCount changes from 1 to 0, TStrings calls the SetUpdateState method with a parameter of false. This allows descendant

classes to perform optimizations when handling multiple updates.

See Also

BeginUpdate

EndUpdate

SetUpdateState

50.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDisplayOption (see page 4921)	TDisplayOption and TDisplayOptions (see page 4923) control the appearance of a value list editor.
	TKeyOption (see page 4922)	TKeyOption and TKeyOptions (see page 4924) indicate the response of a value list editor to certain keystrokes.

50.2.1 ValEdit.TDisplayOption Enumeration

TDisplayOption and TDisplayOptions (see page 4923) control the appearance of a value list editor.

Pascal

```
TDisplayOption = (
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
);
```

C++

```
enum TDisplayOption {
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
};
```

File

ValEdit

Description

TDisplayOption is a set of TDisplayOptions (see page 4923) values. The possible values of TDisplayOptions (see page 4923) are

Option	Meaning
doColumnTitles	The first row of the value list editor is a fixed (nonscrolling) row that displays the captions specified by the TitleCaptions property.
doAutoColResize	When the editor is resized, the column widths automatically adjust so that they completely fill the new width of the editor. If DisplayOptions does not include doKeyColFixed, each column has a width equal to half the width of the editor. Otherwise, the value column takes up any space left over after the fixed (key) column.
doKeyColFixed	The first (key) column of the value list editor is fixed in width. When the value list editor is resized, only the second (value) column resizes. This option is only used when DisplayOptions includes doAutoColResize.

50.2.2 ValEdit.TKeyOption Enumeration

TKeyOption and TKeyOptions (see page 4924) indicate the response of a value list editor to certain keystrokes.

Pascal

```
TKeyOption = (
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
);
```

C++

```
enum TKeyOption {
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
};
```

File

ValEdit

Description

TKeyOption is a set of TKeyOptions (see page 4924) values. The possible values of TKeyOptions (see page 4924) are

Option	Meaning
keyEdit	The user can select and edit values in the first (key) column. That is, the user can specify the name in a name/value pair.
keyAdd	The user can add new name/value pairs. When included, the user can create new entries by pressing the Insert (see page 3698) key or using the down arrow key when positioned on the last name/value pair. keyAdd requires that keyEdit is also included.
keyDelete	The user can delete the selected name/value pair by pressing the Delete (see page 3653) key.
keyUnique	The user can't create name/value pairs where the name portion is the same as the name in another name/value pair. Attempting to add a duplicate name causes the value list editor to raise an exception. keyUnique requires that keyEdit is also included.

50.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDisplayOptions (see page 4923)	This is type ValEdit.TDisplayOptions.
TGetPickListEvent (see page 4923)	This is type ValEdit.TGetPickListEvent.
TItemProps (see page 4924)	This is type ValEdit.TItemProps.
TKeyOptions (see page 4924)	This is type ValEdit.TKeyOptions.
TOnValidateEvent (see page 4924)	This is type ValEdit.TOnValidateEvent.

50.3.1 ValEdit.TDisplayOptions Type

Pascal

```
TDisplayOptions = set of TDisplayOption;
```

C++

```
TDisplayOption TDisplayOptions;
```

File

ValEdit

Description

This is type ValEdit.TDisplayOptions.

50.3.2 ValEdit.TGetPickListEvent Type

Pascal

```
TGetPickListEvent = procedure (Sender: TObject; const KeyName: string; Values: TStrings) of
object;
```

C++

```
(Sender: TObject; const KeyName: string; Values: TStrings) ( TGetPickListEvent)();
```

File

ValEdit

Description

This is type ValEdit.TGetPickListEvent.

50.3.3 ValEdit.TItemProps Type

Pascal

```
TItemProps = array of TItemProp;
```

C++

```
array of TItemProp TItemProps;
```

File

ValEdit

Description

This is type ValEdit.TItemProps.

50.3.4 ValEdit.TKeyOptions Type

Pascal

```
TKeyOptions = set of TKeyOption;
```

C++

```
TKeyOption TKeyOptions;
```

File

ValEdit

Description

This is type ValEdit.TKeyOptions.

50.3.5 ValEdit.TOnValidateEvent Type

Pascal

```
TOnValidateEvent = procedure (Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue:
string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue: string) ( TOnValidateEvent)();
```

File

ValEdit

Description

This is type ValEdit.TOnValidateEvent.

51 VarCmplx Namespace

This is namespace VarCmplx.

Topics

Name	Description
Functions (see page 4927)	The following table lists functions in this documentation.
Variables (see page 4971)	The following table lists variables in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 4928)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a complex number.
	VarComplex (see page 4929)	Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent complex numbers.
	VarComplexAbs (see page 4929)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 4930)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 4931)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 4931)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 4932)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 4933)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 4934)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 4935)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 4937)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 4938)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 4939)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 4940)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 4941)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 4942)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 4943)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 4944)	Returns the conjugate of a complex number.
	VarComplexCos (see page 4945)	Returns the cosine of a complex number.
	VarComplexCosH (see page 4946)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 4947)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 4948)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.

	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCsc (see page 4950)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 4951)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 4952)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 4953)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 4954)	Returns the inverse of a complex number.
	VarComplexLn (see page 4954)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 4955)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 4956)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 4956)	Returns the log base N of a complex number.
	VarComplexPower (see page 4957)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 4958)	Returns the secant of a complex number.
	VarComplexSecH (see page 4959)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 4960)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 4960)	Reduces a Variant (see page 3430) that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 4961)	Returns the sine of a complex number.
	VarComplexSinH (see page 4962)	Returns the hyperbolic sine of a complex number.
	VarComplexSqr (see page 4963)	Returns the square of a complex number.
	VarComplexSqrt (see page 4964)	Returns the square root of a complex number.
	VarComplexTan (see page 4964)	Returns the tangent of a complex number.
	VarComplexTanH (see page 4965)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 4967)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 4967)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 4968)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 4968)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 4969)	Computes the polar coordinates that correspond to a custom Variant (see page 3430) that represents a complex number.
	VarIsComplex (see page 4969)	Indicates whether a Variant (see page 3430) stores its data internally as a complex number.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 4971)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 4971)	This is variable VarCmplx.ComplexNumberSymbol.
ComplexNumberSymbolBeforeImaginary (see page 4972)	This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

51.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 4928)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a complex number.
	VarComplex (see page 4929)	Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent complex numbers.
	VarComplexAbs (see page 4929)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 4930)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 4931)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 4931)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 4932)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 4933)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 4934)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 4935)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 4937)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 4938)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 4939)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 4940)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 4941)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 4942)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 4943)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 4944)	Returns the conjugate of a complex number.
	VarComplexCos (see page 4945)	Returns the cosine of a complex number.
	VarComplexCosH (see page 4946)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 4947)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 4948)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCreate (see page 4949)	Returns a custom Variant (see page 3430) that represents a complex number.
	VarComplexCsc (see page 4950)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 4951)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 4952)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 4953)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 4954)	Returns the inverse of a complex number.
	VarComplexLn (see page 4954)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 4955)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 4956)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 4956)	Returns the log base N of a complex number.
	VarComplexPower (see page 4957)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 4958)	Returns the secant of a complex number.
	VarComplexSecH (see page 4959)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 4960)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 4960)	Reduces a Variant (see page 3430) that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 4961)	Returns the sine of a complex number.
	VarComplexSinH (see page 4962)	Returns the hyperbolic sine of a complex number.

	VarComplexSqr (see page 4963)	Returns the square of a complex number.
	VarComplexSqrt (see page 4964)	Returns the square root of a complex number.
	VarComplexTan (see page 4964)	Returns the tangent of a complex number.
	VarComplexTanH (see page 4965)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 4967)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 4967)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 4968)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 4968)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 4969)	Computes the polar coordinates that correspond to a custom Variant (see page 3430) that represents a complex number.
	VarIsComplex (see page 4969)	Indicates whether a Variant (see page 3430) stores its data internally as a complex number.

51.1.1 VarCmplx.VarAsComplex Function

Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a complex number.

Pascal

```
function VarAsComplex(const AValue: Variant): Variant;
```

C++

```
Variant VarAsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarAsComplex converts the Variant (see page 3430) specified by AValue to a custom Variant (see page 3430) representing the corresponding complex number. It returns the custom Variant (see page 3430).

If VarAsComplex can't perform the conversion, it raises an EInvalidCast exception.

Complex Variants (see page 4997) are custom Variants (see page 4997) that represent complex numbers. The Variant (see page 3430) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3335) values, and boolean values. In addition, the VarCmplx (see page 4925) unit implements a number of global functions for operating on complex Variants (see page 4997).

See Also

EInvalidCast

VarIsComplex (see page 4969)

VarComplexCreate (see page 4949)

VarComplexSimplify (see page 4960)

VarAsType

51.1.2 VarCmplx.VarComplex Function

Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent complex numbers.

Pascal

```
function VarComplex: TVarType;
```

C++

```
TVarType VarComplex();
```

File

VarCmplx

Description

Use VarComplex to obtain the variant type code for custom Variants (see page 4997) that represent complex numbers. The Variant (see page 3430) type codes for custom Variants (see page 4997) are dynamically allocated when the application loads the unit that implements them. VarComplex allows the application to identify the Variant (see page 3430) type code that is currently used for custom Variants (see page 4997) that represent complex numbers. This type code, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3430) types.

Complex Variants (see page 4997) are custom Variants (see page 4997) that represent complex numbers. The Variant (see page 3430) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3335) values, and boolean values. In addition, the VarCmplx (see page 4925) unit implements a number of global functions for operating on complex Variants (see page 4997).

See Also

VarType

51.1.3 VarCmplx.VarComplexAbs Function

Returns the absolute value of a complex number.

Pascal

```
function VarComplexAbs(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbs(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbs to take the absolute value (length) of a complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexAbs returns the absolute value of AValue.

See Also

VarComplexAbsSqr (see page 4930)

VarComplexAngle (see page 4931)

VarComplexConjugate (see page 4944)

VarComplexInverse (see page 4954)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

51.1.4 VarCmplx.VarComplexAbsSqr Function

Returns the square of the absolute value of a complex number.

Pascal

```
function VarComplexAbsSqr(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbsSqr(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbsSqr to obtain the square of a complex number's absolute value.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexAbs (see page 4929) returns the absolute value of AValue, squared.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexConjugate (see page 4944)

VarComplexInverse (see page 4954)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexLn (see page 4954)

VarComplexExp (see page 4952)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.5 VarCmplx.VarComplexAngle Function

Returns the angle made by a complex number.

Pascal

```
function VarComplexAngle(const AValue: Variant): Double;
```

C++

```
Double VarComplexAngle(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAngle to obtain the angle of a complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexAbs (see page 4929) returns the angle that corresponds to AValue.

See Also

VarComplexToPolar (see page 4969)

VarComplexAbs (see page 4929)

51.1.6 VarCmplx.VarComplexArcCos Function

Returns the inverse cosine of a complex number.

Pascal

```
function VarComplexArcCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCos computes the inverse cosine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCos returns a Variant (see page 3430) whose value is the inverse cosine of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)
VarComplexArcSec (see page 4938)
VarComplexArcSecH (see page 4939)
VarComplexArcSin (see page 4940)
VarComplexArcSinH (see page 4941)
VarComplexArcTan (see page 4942)
VarComplexArcTanH (see page 4943)
VarComplexCos (see page 4945)
VarComplexCosH (see page 4946)
VarComplexCot (see page 4947)
VarComplexCotH (see page 4948)
VarComplexCsc (see page 4950)
VarComplexCscH (see page 4951)
VarComplexSec (see page 4958)
VarComplexSecH (see page 4959)
VarComplexSin (see page 4961)
VarComplexSinH (see page 4962)
VarComplexTan (see page 4964)
VarComplexTanH (see page 4965)

51.1.7 VarCmplx.VarComplexArcCosH Function

Returns the hyperbolic inverse cosine of a complex number.

Pascal

```
function VarComplexArcCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCosH computes the hyperbolic inverse cosine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCosH returns a Variant (see page 3430) whose value is the hyperbolic inverse cosine of AValue.

See Also

VarComplexArcCos (see page 4931)

VarComplexArcCot (see page 4933)
VarComplexArcCotH (see page 4934)
VarComplexArcCsc (see page 4935)
VarComplexArcCscH (see page 4937)
VarComplexArcSec (see page 4938)
VarComplexArcSecH (see page 4939)
VarComplexArcSin (see page 4940)
VarComplexArcSinH (see page 4941)
VarComplexArcTan (see page 4942)
VarComplexArcTanH (see page 4943)
VarComplexCos (see page 4945)
VarComplexCosH (see page 4946)
VarComplexCot (see page 4947)
VarComplexCotH (see page 4948)
VarComplexCsc (see page 4950)
VarComplexCscH (see page 4951)
VarComplexSec (see page 4958)
VarComplexSecH (see page 4959)
VarComplexSin (see page 4961)
VarComplexSinH (see page 4962)
VarComplexTan (see page 4964)
VarComplexTanH (see page 4965)

51.1.8 VarCmplx.VarComplexArcCot Function

Returns the inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot computes the inverse cotangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCot returns a Variant (see page 3430) whose value is the inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.9 VarCmplx.VarComplexArcCotH Function

Returns the hyperbolic inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot (see page 4933) computes the hyperbolic inverse cotangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCot (see page 4933) returns a Variant (see page 3430) whose value is the hyperbolic inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCos (see page 4931)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.10 VarCmplx.VarComplexArcCsc Function

Returns the inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc computes the inverse cosecant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCsc returns a Variant (see page 3430) whose value is the inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.11 VarCmplx.VarComplexArcCscH Function

Returns the hyperbolic inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc (see page 4935) computes the hyperbolic inverse cosecant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcCsc (see page 4935) returns a Variant (see page 3430) whose value is the hyperbolic inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCsc (see page 4935)

VarComplexArcCos (see page 4931)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.12 VarCmplx.VarComplexArcSec Function

Returns the inverse secant of a complex number.

Pascal

```
function VarComplexArcSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec computes the inverse secant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcSec returns a Variant (see page 3430) whose value is the inverse secant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCscH (see page 4937)

VarComplexArcCsc (see page 4935)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.13 VarCmplx.VarComplexArcSecH Function

Returns the hyperbolic inverse secant of a complex number.

Pascal

```
function VarComplexArcSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec (see page 4938) computes the hyperbolic inverse secant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcSec (see page 4938) returns a Variant (see page 3430) whose value is the hyperbolic inverse secant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.14 VarCmplx.VarComplexArcSin Function

Returns the inverse sine of a complex number.

Pascal

```
function VarComplexArcSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSin computes the inverse sine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcSin returns a Variant (see page 3430) whose value is the inverse sine of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.15 VarCmplx.VarComplexArcSinH Function

Returns the hyperbolic inverse sine of a complex number.

Pascal

```
function VarComplexArcSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSinH computes the hyperbolic inverse sine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcSinH returns a Variant (see page 3430) whose value is the hyperbolic inverse sine of AValue.

See Also

VarComplexArcSinH

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcCosH (see page 4932)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.16 VarCmplx.VarComplexArcTan Function

Returns the inverse tangent of a complex number.

Pascal

```
function VarComplexArcTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTan computes the inverse tangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcTan returns a Variant (see page 3430) whose value is the inverse tangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcTanH (see page 4943)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.17 VarCmplx.VarComplexArcTanH Function

Returns the hyperbolic inverse tangent of a complex number.

Pascal

```
function VarComplexArcTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTanH computes the hyperbolic inverse tangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexArcTanH returns a Variant (see page 3430) whose value is the hyperbolic inverse tangent of AValue.

See Also

VarComplexArcTan (see page 4942)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcCosH (see page 4932)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.18 VarCmplx.VarComplexConjugate Function

Returns the conjugate of a complex number.

Pascal

```
function VarComplexConjugate(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexConjugate(const Variant AValue);
```

File

VarCmplx

Description

VarComplexConjugate returns the conjugate of a complex number. The conjugate is the value that has the same real part and the inverse of the imaginary part.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexConjugate returns a Variant (see page 3430) whose value is the conjugate of AValue.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexInverse (see page 4954)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexLn (see page 4954)

VarComplexExp (see page 4952)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.19 VarCmplx.VarComplexCos Function

Returns the cosine of a complex number.

Pascal

```
function VarComplexCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCos computes the cosine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCos returns a Variant (see page 3430) whose value is the cosine of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexArcCos (see page 4931)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.20 VarCmplx.VarComplexCosH Function

Returns the hyperbolic cosine of a complex number.

Pascal

```
function VarComplexCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCosH computes the hyperbolic cosine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCosH returns a Variant (see page 3430) whose value is the hyperbolic cosine of AValue.

See Also

VarComplexArcCos (see page 4931)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.21 VarCmplx.VarComplexCot Function

Returns the cotangent of a complex number.

Pascal

```
function VarComplexCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot computes the cotangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCot returns a Variant (see page 3430) whose value is the cotangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)
VarComplexArcCsc (see page 4935)
VarComplexArcCscH (see page 4937)
VarComplexArcSec (see page 4938)
VarComplexArcSecH (see page 4939)
VarComplexArcSin (see page 4940)
VarComplexArcSinH (see page 4941)
VarComplexArcTan (see page 4942)
VarComplexArcTanH (see page 4943)
VarComplexCos (see page 4945)
VarComplexCosH (see page 4946)
VarComplexCot
VarComplexCotH (see page 4948)
VarComplexCsc (see page 4950)
VarComplexCscH (see page 4951)
VarComplexSec (see page 4958)
VarComplexSecH (see page 4959)
VarComplexSin (see page 4961)
VarComplexSinH (see page 4962)
VarComplexTan (see page 4964)
VarComplexTanH (see page 4965)

51.1.22 VarCmplx.VarComplexCotH Function

Returns the hyperbolic cotangent of a complex number.

Pascal

```
function VarComplexCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot (see page 4947) computes the hyperbolic cotangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCot (see page 4947) returns a Variant (see page 3430) whose value is the hyperbolic cotangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCos (see page 4931)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.23 VarCmplx.VarComplexCreate Function

Returns a custom Variant (see page 3430) that represents a complex number.

Pascal

```
function VarComplexCreate: Variant; overload;
function VarComplexCreate(const AReal: Double): Variant; overload;
function VarComplexCreate(const AReal: Double; const AImaginary: Double): Variant; overload;
function VarComplexCreate(const AText: string): Variant; overload;
```

C++

```
Variant VarComplexCreate();
Variant VarComplexCreate(const Double AReal);
Variant VarComplexCreate(const Double AReal, const Double AImaginary);
Variant VarComplexCreate(const UnicodeString AText);
```

File

VarCmplx

Description

Call VarComplexCreate to create a Variant (see page 3430) whose value is a complex number.

AReal is the real part of the complex number.

AImaginary is the imaginary part of the complex number.

Using the first three versions of the constructor, omitting AReal or AImaginary results in a real or imaginary part of 0.

Using the last syntax, AText is parsed using the ComplexNumberSymbol (see page 4971) and ComplexNumberSymbolBeforeImaginary (see page 4972) global variables and, if the string represents a complex value, that becomes the value of the new Variant (see page 3430). If the string can't be parsed into a complex number, VarComplexCreate raises an EConvertError exception.

Complex Variants (see page 4997) are custom Variants (see page 4997) that represent complex numbers. The Variant (see page 3430) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3335) values, and boolean values. In addition, the VarCmplx (see page 4925) unit implements a number of global functions for operating on complex Variants (see page 4997).

See Also

VarIsComplex (see page 4969)

VarAsComplex (see page 4928)

VarComplexSimplify (see page 4960)

VarComplexFromPolar (see page 4953)

ComplexNumberSymbol (see page 4971)

ComplexNumberSymbolBeforeImaginary (see page 4972)

EConvertError

51.1.24 VarCmplx.VarComplexCsc Function

Returns the cosecant of a complex number.

Pascal

```
function VarComplexCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc computes the cosecant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCsc returns a Variant (see page 3430) whose value is the cosecant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.25 VarCmplx.VarComplexCscH Function

Returns the hyperbolic cosecant of a complex number.

Pascal

```
function VarComplexCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc (see page 4950) computes the hyperbolic cosecant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexCsc (see page 4950) returns a Variant (see page 3430) whose value is the hyperbolic cosecant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCsc (see page 4935)

VarComplexArcCos (see page 4931)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.26 VarCmplx.VarComplexExp Function

Returns the exponential of a complex number.

Pascal

```
function VarComplexExp(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexExp(const Variant AValue);
```

File

VarCmplx

Description

VarComplexExp returns value of e raised to the power of a specified complex number, where e is the base of the natural logarithms.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexExp returns a Variant (see page 3430) whose value is the exponential of AValue.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexInverse (see page 4954)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexLn (see page 4954)

VarComplexConjugate (see page 4944)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.27 VarCmplx.VarComplexFromPolar Function

Converts a set of polar coordinates to a complex number.

Pascal

```
function VarComplexFromPolar(const ARadius: Double; const ATheta: Double): Variant;
```

C++

```
Variant VarComplexFromPolar(const Double ARadius, const Double ATheta);
```

File

VarCmplx

Description

VarComplexFromPolar returns the complex number that corresponds to the radius (length) and angle specified by the ARadius and ATheta parameters, respectively. The result is returned as a Variant (see page 3430) whose value is the corresponding complex number.

See Also

VarComplexCreate (see page 4949)

VarComplexToPolar (see page 4969)

51.1.28 VarCmplx.VarComplexInverse Function

Returns the inverse of a complex number.

Pascal

```
function VarComplexInverse(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexInverse(const Variant AValue);
```

File

VarCmplx

Description

VarComplexInverse returns the inverse of a complex number. The inverse is the value that results when you divide 1 by the complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexInverse returns a Variant (see page 3430) whose value is the inverse of AValue.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexConjugate (see page 4944)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexLn (see page 4954)

VarComplexExp (see page 4952)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.29 VarCmplx.VarComplexLn Function

Returns the natural log of a complex number.

Pascal

```
function VarComplexLn(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLn(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLn returns the natural logarithm of a complex number. The natural logarithm is the power to which you must raise e to get the complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexLn returns a Variant (see page 3430) whose value is the natural log of AValue.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexConjugate (see page 4944)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexInverse (see page 4954)

VarComplexExp (see page 4952)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.30 VarCmplx.VarComplexLog10 Function

Returns the log base 10 of a complex number.

Pascal

```
function VarComplexLog10(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog10(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog10 returns the base 10 logarithm of a complex number. The base 10 logarithm is the power to which you must raise 10 to get the complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexLog10 returns a Variant (see page 3430) whose value is the log base 10 of AValue.

See Also

VarComplexLn (see page 4954)

VarComplexLog2 (see page 4956)

VarComplexLogN (see page 4956)

51.1.31 VarCmplx.VarComplexLog2 Function

Returns the log base 2 of a complex number.

Pascal

```
function VarComplexLog2(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog2(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog2 returns the base 2 logarithm of a complex number. The base 2 logarithm is the power to which you must raise 2 to get the complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexLog2 returns a Variant (see page 3430) whose value is the log base 2 of AValue.

See Also

VarComplexLn (see page 4954)

VarComplexLog10 (see page 4955)

VarComplexLogN (see page 4956)

51.1.32 VarCmplx.VarComplexLogN Function

Returns the log base N of a complex number.

Pascal

```
function VarComplexLogN(const AValue: Variant; const X: Double): Variant;
```

C++

```
Variant VarComplexLogN(const Variant AValue, const Double X);
```

File

VarCmplx

Description

VarComplexLogN returns the base N logarithm of a complex number. The base N logarithm is the power to which you must raise an exponent N to get the complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

X is the logarithm exponent.

VarComplexLogN returns a Variant (see page 3430) whose value is the log base N of AValue.

See Also

VarComplexLn (see page 4954)

VarComplexLog2 (see page 4956)

VarComplexLog10 (see page 4955)

51.1.33 VarCmplx.VarComplexPower Function

Returns the value of a complex number raised to a specified power.

Pascal

```
function VarComplexPower(const AValue: Variant; const APower: Variant): Variant;
```

C++

```
Variant VarComplexPower(const Variant AValue, const Variant APower);
```

File

VarCmplx

Description

VarComplexPower returns the result of raising AValue to the power APower.

AValue is a Variant (see page 3430) whose value represents the complex number.

APower is a Variant (see page 3430) that specifies the power to which you want to raise AValue. It can also be a complex number.

VarComplexPower returns a Variant (see page 3430) whose value is AValue raised to the APower power.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexConjugate (see page 4944)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexInverse (see page 4954)

VarComplexExp (see page 4952)

VarComplexSqr (see page 4963)

VarComplexSqrt (see page 4964)

VarComplexLn (see page 4954)

51.1.34 VarCmplx.VarComplexSec Function

Returns the secant of a complex number.

Pascal

```
function VarComplexSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec computes the secant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexSec returns a Variant (see page 3430) whose value is the secant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCscH (see page 4937)

VarComplexArcCsc (see page 4935)

VarComplexArcSecH (see page 4939)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexSec

VarComplexSecH (see page 4959)

VarComplexSec

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.35 VarCmplx.VarComplexSecH Function

Returns the hyperbolic secant of a complex number.

Pascal

```
function VarComplexSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec (see page 4958) computes the hyperbolic secant of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexSec (see page 4958) returns a Variant (see page 3430) whose value is the hyperbolic secant of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCos (see page 4931)

VarComplexArcCotH (see page 4934)

VarComplexArcCot (see page 4933)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexSec (see page 4958)

VarComplexSecH

VarComplexSec (see page 4958)

VarComplexSecH

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.36 VarCmplx.VarComplexSign Function

Pascal

```
function VarComplexSign(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSign(const Variant AValue);
```

File

VarCmplx

Description

This is function VarCmplx.VarComplexSign.

51.1.37 VarCmplx.VarComplexSimplify Function

Reduces a Variant (see page 3430) that represents a complex number to one that represents a real value, if possible.

Pascal

```
function VarComplexSimplify(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSimplify(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexSimplify to convert a complex number to a real value when possible. VarComplexSimplify checks whether the Variant (see page 3430) specified by AValue represents a complex number that has no imaginary part. If so, it returns a Variant (see page 3430) whose value is simply a real number. If AValue does not represent a complex number, or if AValue represents a complex number with a nonzero imaginary part, VarComplexSimplify returns AValue.

See Also

VarIsComplex (see page 4969)

VarComplexCreate (see page 4949)

VarAsComplex (see page 4928)

VarAsType

51.1.38 VarCmplx.VarComplexSin Function

Returns the sine of a complex number.

Pascal

```
function VarComplexSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSin computes the sine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexSin returns a Variant (see page 3430) whose value is the sine of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcSinH (see page 4941)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.39 VarCmplx.VarComplexSinH Function

Returns the hyperbolic sine of a complex number.

Pascal

```
function VarComplexSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSinH computes the hyperbolic sine of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexSinH returns a Variant (see page 3430) whose value is the hyperbolic sine of AValue.

See Also

VarComplexArcSinH (see page 4941)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcCosH (see page 4932)

VarComplexArcTan (see page 4942)

VarComplexArcTanH (see page 4943)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH

VarComplexTan (see page 4964)

VarComplexTanH (see page 4965)

51.1.40 VarCmplx.VarComplexSqr Function

Returns the square of a complex number.

Pascal

```
function VarComplexSqr(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqr(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqr returns the square of AValue.

AValue is a Variant (see page 3430) whose value represents the complex number.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqr (see page 4930)

VarComplexConjugate (see page 4944)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexInverse (see page 4954)

VarComplexExp (see page 4952)

VarComplexLn (see page 4954)

VarComplexSqrt (see page 4964)

VarComplexPower (see page 4957)

51.1.41 VarCmplx.VarComplexSqrt Function

Returns the square root of a complex number.

Pascal

```
function VarComplexSqrt(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqrt(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqrt returns the square root of AValue.

AValue is a Variant (see page 3430) whose value represents the complex number.

See Also

VarComplexAbs (see page 4929)

VarComplexAngle (see page 4931)

VarComplexAbsSqrt

VarComplexConjugate (see page 4944)

VarComplexTimesPosI (see page 4968)

VarComplexTimesNegI (see page 4967)

VarComplexInverse (see page 4954)

VarComplexExp (see page 4952)

VarComplexLn (see page 4954)

VarComplexSqr (see page 4963)

VarComplexPower (see page 4957)

51.1.42 VarCmplx.VarComplexTan Function

Returns the tangent of a complex number.

Pascal

```
function VarComplexTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTan computes the tangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexTan returns a Variant (see page 3430) whose value is the tangent of AValue.

See Also

VarComplexArcCosH (see page 4932)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcTanH (see page 4943)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan

VarComplexTanH (see page 4965)

51.1.43 VarCmplx.VarComplexTanH Function

Returns the hyperbolic tangent of a complex number.

Pascal

```
function VarComplexTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTanH computes the hyperbolic tangent of a complex number and returns the result.

AValue is a Variant (see page 3430) whose value represents the complex number.

VarComplexTanH returns a Variant (see page 3430) whose value is the hyperbolic tangent of AValue.

See Also

VarComplexArcTan (see page 4942)

VarComplexArcCot (see page 4933)

VarComplexArcCotH (see page 4934)

VarComplexArcCsc (see page 4935)

VarComplexArcCscH (see page 4937)

VarComplexArcSec (see page 4938)

VarComplexArcSecH (see page 4939)

VarComplexArcCos (see page 4931)

VarComplexArcCosH (see page 4932)

VarComplexArcSin (see page 4940)

VarComplexArcSinH (see page 4941)

VarComplexCos (see page 4945)

VarComplexCosH (see page 4946)

VarComplexCot (see page 4947)

VarComplexCotH (see page 4948)

VarComplexCsc (see page 4950)

VarComplexCscH (see page 4951)

VarComplexSec (see page 4958)

VarComplexSecH (see page 4959)

VarComplexSin (see page 4961)

VarComplexSinH (see page 4962)

VarComplexTan (see page 4964)

VarComplexTanH

51.1.44 VarCmplx.VarComplexTimesImaginary Function

Returns the value of a complex number multiplied by an imaginary number.

Pascal

```
function VarComplexTimesImaginary(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesImaginary(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesImaginary returns a complex value times an imaginary number.

AValue is a Variant (see page 3430) whose value represents the complex number.

AFactor is a real number that represents the imaginary number. That is, the imaginary number is AFactor times i.

VarComplexTimesImaginary returns a variant whose value is AValue times AFactor times i.

See Also

VarComplexTimesNegI (see page 4967)

VarComplexTimesPosI (see page 4968)

VarComplexTimesReal (see page 4968)

51.1.45 VarCmplx.VarComplexTimesNegI Function

Returns the value of a complex number multiplied by –i.

Pascal

```
function VarComplexTimesNegI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesNegI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesNegI returns AValue times –i.

AValue is a Variant (see page 3430) whose value represents the complex number.

See Also

VarComplexTimesImaginary (see page 4967)

VarComplexTimesPosI (see page 4968)

VarComplexTimesReal (see page 4968)

51.1.46 VarCmplx.VarComplexTimesPosI Function

Returns the value of a complex number multiplied by i.

Pascal

```
function VarComplexTimesPosI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesPosI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesPosI returns AValue times i.

AValue is a Variant (see page 3430) whose value represents the complex number.

See Also

VarComplexTimesNegI (see page 4967)

VarComplexTimesImaginary (see page 4967)

VarComplexTimesReal (see page 4968)

51.1.47 VarCmplx.VarComplexTimesReal Function

Returns the value of a complex number multiplied by a real number.

Pascal

```
function VarComplexTimesReal(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesReal(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesReal returns a complex value times a real number.

AValue is a Variant (see page 3430) whose value represents the complex number.

AFactor is a real number.

VarComplexTimesReal returns a variant whose value is AValue times AFactor.

See Also

VarComplexTimesNegI (see page 4967)

VarComplexTimesPosI (see page 4968)

VarComplexTimesReal

51.1.48 VarCmplx.VarComplexToPolar Function

Computes the polar coordinates that correspond to a custom Variant (see page 3430) that represents a complex number.

Pascal

```
procedure VarComplexToPolar(const AValue: Variant; var ARadius: Double; var ATheta: Double;
AFixTheta: Boolean = True);
```

C++

```
VarComplexToPolar(const Variant AValue, Double ARadius, Double ATheta, Boolean AFixTheta =
True);
```

File

VarCmplx

Description

Call VarComplexToPolar to obtain the polar coordinates for a complex number.

AValue is a Variant (see page 3430) whose value represents the complex number.

ARadius returns the radius of the corresponding polar coordinates.

ATheta returns the angle (Theta) of the corresponding polar coordinates.

AFixTheta indicates whether VarComplexToPolar should normalize the angle (ATheta). When AFixTheta is true, VarComplexToPolar always returns an angle between – pi and pi.

See Also

VarComplexAngle (see page 4931)

VarComplexFromPolar (see page 4953)

VarComplexAbs (see page 4929)

51.1.49 VarCmplx.VarIsComplex Function

Indicates whether a Variant (see page 3430) stores its data internally as a complex number.

Pascal

```
function VarIsComplex(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarIsComplex returns true if AValue represents a complex number (that is, if it has a VarType (see page 5101) of VarComplex (see page 4929)).

Complex Variants (see page 4997) are custom Variants (see page 4997) that represent complex numbers. The Variant (see page 3430) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division),

and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3335) values, and boolean values. In addition, the VarCmplx (see page 4925) unit implements a number of global functions for operating on complex Variants (see page 4997).

See Also

VarAsComplex (see page 4928)

VarComplexCreate (see page 4949)

VarComplexSimplify (see page 4960)

VarComplex (see page 4929)

VarType

51.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 4971)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 4971)	This is variable VarCmplx.ComplexNumberSymbol.
ComplexNumberSymbolBeforeImaginary (see page 4972)	This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

51.2.1 VarCmplx.ComplexNumberDefuzzAtZero Variable

Pascal

```
ComplexNumberDefuzzAtZero: Boolean = True;
```

C++

```
Boolean ComplexNumberDefuzzAtZero = True;
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberDefuzzAtZero.

51.2.2 VarCmplx.ComplexNumberSymbol Variable

Pascal

```
ComplexNumberSymbol: string = 'i';
```

C++

```
UnicodeString ComplexNumberSymbol = 'i';
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberSymbol.

51.2.3 VarCmplx.ComplexNumberSymbolBeforeImaginary Variable

Pascal

```
ComplexNumberSymbolBeforeImaginary: Boolean = False;
```

C++

```
Boolean ComplexNumberSymbolBeforeImaginary = False;
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

52 VarConv Namespace

This is namespace VarConv.

Topics

Name	Description
Functions (see page 4974)	The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 4974)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a measurement.
	VarAsConvert (see page 4974)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a measurement.
	VarConvert (see page 4975)	Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent measurements.
	VarConvertCreate (see page 4976)	Returns a custom Variant (see page 3430) that represents a measurement.
	VarConvertCreate (see page 4976)	Returns a custom Variant (see page 3430) that represents a measurement.
	VarIsConvert (see page 4977)	Indicates whether a Variant (see page 3430) is a custom Variant (see page 3430) that represents a measurement.

52.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 4974)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a measurement.
	VarAsConvert (see page 4974)	Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a measurement.
	VarConvert (see page 4975)	Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent measurements.
	VarConvertCreate (see page 4976)	Returns a custom Variant (see page 3430) that represents a measurement.
	VarConvertCreate (see page 4976)	Returns a custom Variant (see page 3430) that represents a measurement.
	VarIsConvert (see page 4977)	Indicates whether a Variant (see page 3430) is a custom Variant (see page 3430) that represents a measurement.

52.1.1 VarConv.VarAsConvert Function

Casts an arbitrary Variant (see page 3430) so that it is a custom Variant (see page 3430) representing a measurement.

Pascal

```
function VarAsConvert(const AValue: Variant): Variant; overload;
function VarAsConvert(const AValue: Variant; const AType: TConvType): Variant; overload;
```

C++

```
Variant VarAsConvert(const Variant AValue);
Variant VarAsConvert(const Variant AValue, const TConvType AType);
```

File

VarConv

Description

VarAsConvert converts the Variant (see page 3430) specified by AValue to a VarConvert (see page 4975) custom Variant (see page 3430).

Using the first syntax, AValue is interpreted as the numeric portion (the number of units), and AType identifies the units.

Using the second syntax, AValue should be a Variant (see page 3430) that can be parsed into a VarConv (see page 4973) custom Variant (see page 3430) (for example, a variant whose value is a string such as '1.5 miles').

If VarAsConvert can't perform the conversion, it raises an EInvalidCast (see page 4000) exception.

VarConvert (see page 4975) custom Variants (see page 4997) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4973) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4973) Variant (see page 3430) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4973) Variants (see page 4997). VarConv (see page 4973) custom Variants (see page 4997) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3430) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

EInvalidCast

VarIsConvert (see page 4977)

VarConvertCreate (see page 4976)

VarAsType

TVarData

52.1.2 VarConv.VarConvert Function

Returns the Variant (see page 3430) type code for custom Variants (see page 4997) that represent measurements.

Pascal

```
function VarConvert: TVarType;
```

C++

```
TVarType VarConvert();
```

File

VarConv

Description

Use VarConvert to obtain the variant type code for custom Variants (see page 4997) that represent measurements. The Variant (see page 3430) type codes for custom Variants (see page 4997) are dynamically allocated when the application loads the unit that implements them. VarConvert allows the application to identify the Variant (see page 3430) type code that is currently used for custom Variants (see page 4997) that represent VarConvert custom Variants (see page 4997). This type code, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3430) types.

VarConvert custom Variants (see page 4997) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4973) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4973) Variant (see page 3430) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4973) Variants (see page 4997). VarConv (see page 4973) custom Variants (see page 4997) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3430) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarType

52.1.3 VarConv.VarConvertCreate Function

Returns a custom Variant (see page 3430) that represents a measurement.

Pascal

```
function VarConvertCreate(const AValue: Double; const AType: TConvType): Variant; overload;
function VarConvertCreate(const AValue: string): Variant; overload;
```

C++

```
Variant VarConvertCreate(const Double AValue, const TConvType AType);
Variant VarConvertCreate(const UnicodeString AValue);
```

File

VarConv

Description

Call VarConvertCreate to create a Variant (see page 3430) whose value is a measurement.

Using the first syntax, AType indicates the measurement units, and AValue is the number of those units. AType must represent a registered conversion unit.

Using the second syntax, AValue is a string that can be parsed by the TryStrToConvUnit function. (For example '1.5 liters'.)

If the string can't be parsed into a measurement, VarConvertCreate raises an EInvalidCast (see page 4000) exception.

VarConvert (see page 4975) custom Variants (see page 4997) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4973) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4973) Variant (see page 3430) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4973) Variants (see page 4997). VarConv (see page 4973) custom Variants (see page 4997) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3430) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarIsConvert (see page 4977)

VarAsConvert (see page 4974)

EInvalidCast

52.1.4 VarConv.VarIsConvert Function

Indicates whether a Variant (see page 3430) is a custom Variant (see page 3430) that represents a measurement.

Pascal

```
function VarIsConvert(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsConvert(const Variant AValue);
```

File

VarConv

Description

VarIsConvert returns true if AValue represents a measurement (that is, if it has a VarType (see page 5101) of VarConvert (see page 4975)).

VarConvert (see page 4975) custom Variants (see page 4997) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4973) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4973) Variant (see page 3430) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4973) Variants (see page 4997). VarConv (see page 4973) custom Variants (see page 4997) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3430) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarAsConvert (see page 4974)

VarConvertCreate (see page 4976)

VarConvert (see page 4975)

VarType

53 VarHlpr Namespace

This is namespace VarHlpr.

Topics

Name	Description
Functions (see page 4981)	The following table lists functions in this documentation.

Functions

	Name	Description
	VariantAdd (see page 4981)	Adds two Variants (see page 4997).
	VariantAdd2 (see page 4982)	Adds two Variants (see page 4997).
	VariantAnd (see page 4982)	Performs an **and** operation between two Variant (see page 3430) values.
	VariantAnd2 (see page 4983)	Performs an **and** operation between two Variants (see page 4997).
	VariantArrayRedim (see page 4983)	Resizes a Variant (see page 3430) array.
	VariantCast (see page 4984)	Performs a type-cast on a Variant (see page 3430).
	VariantClear (see page 4984)	Empties a Variant (see page 3430), so that it is unassigned.
	VariantCpy (see page 4985)	Copies a Variant (see page 3430).
	VariantDiv (see page 4985)	Divides two Variants (see page 4997).
	VariantDiv2 (see page 4986)	Divides two Variants (see page 4997).
	VariantFromUnicodeString (see page 4986)	Assigns an Unicode string to a Variant (see page 3430).
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantMod (see page 4988)	Returns the remainder on division of two Variants (see page 4997).
	VariantMod2 (see page 4988)	Returns the remainder on division of two Variants (see page 4997).
	VariantMul (see page 4989)	Multiplies two Variants (see page 4997).
	VariantMul2 (see page 4989)	Multiplies two Variants (see page 4997).
	VariantNeg (see page 4990)	Inverts the sign of a Variant (see page 3430).
	VariantNot (see page 4990)	Performs a **not** operation on a Variant (see page 3430).
	VariantOr (see page 4990)	Performs an **or** operation between two Variants (see page 4997).
	VariantOr2 (see page 4991)	Performs an **or** operation between two Variants (see page 4997).
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.

	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantShl (see page 4992)	This is function VarHlpr.VariantShl.
	VariantShl2 (see page 4993)	Performs an **shl** (shift left) operation between two Variants (see page 4997).
	VariantShr (see page 4993)	Performs an **shr** (shift right) operation between two Variants (see page 4997).
	VariantShr2 (see page 4994)	Performs an **shr** (shift right) operation between two Variants (see page 4997).
	VariantSub (see page 4994)	Subtracts two Variants (see page 4997).
	VariantSub2 (see page 4995)	Subtracts two Variants (see page 4997).
	VariantToUnicodeString (see page 4995)	Converts a Variant (see page 3430) to an Unicode string.
	VariantXor (see page 4996)	Performs a **xor** (exclusive or) operation between two Variants (see page 4997).
	VariantXor2 (see page 4996)	Performs a **xor** (exclusive or) operation between two Variants (see page 4997).

53.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VariantAdd (see page 4981)	Adds two Variants (see page 4997).
	VariantAdd2 (see page 4982)	Adds two Variants (see page 4997).
	VariantAnd (see page 4982)	Performs an **and** operation between two Variant (see page 3430) values.
	VariantAnd2 (see page 4983)	Performs an **and** operation between two Variants (see page 4997).
	VariantArrayRedim (see page 4983)	Resizes a Variant (see page 3430) array.
	VariantCast (see page 4984)	Performs a type-cast on a Variant (see page 3430).
	VariantClear (see page 4984)	Empties a Variant (see page 3430), so that it is unassigned.
	VariantCpy (see page 4985)	Copies a Variant (see page 3430).
	VariantDiv (see page 4985)	Divides two Variants (see page 4997).
	VariantDiv2 (see page 4986)	Divides two Variants (see page 4997).
	VariantFromUnicodeString (see page 4986)	Assigns an Unicode string to a Variant (see page 3430).
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantGetElement (see page 4987)	Returns an element from a Variant (see page 3430) array.
	VariantMod (see page 4988)	Returns the remainder on division of two Variants (see page 4997).
	VariantMod2 (see page 4988)	Returns the remainder on division of two Variants (see page 4997).
	VariantMul (see page 4989)	Multiplies two Variants (see page 4997).
	VariantMul2 (see page 4989)	Multiplies two Variants (see page 4997).
	VariantNeg (see page 4990)	Inverts the sign of a Variant (see page 3430).
	VariantNot (see page 4990)	Performs a **not** operation on a Variant (see page 3430).
	VariantOr (see page 4990)	Performs an **or** operation between two Variants (see page 4997).
	VariantOr2 (see page 4991)	Performs an **or** operation between two Variants (see page 4997).
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantPutElement (see page 4992)	Sets the value of an element in a Variant (see page 3430) array.
	VariantShl (see page 4992)	This is function VarHlpr.VariantShl.
	VariantShl2 (see page 4993)	Performs an **shl** (shift left) operation between two Variants (see page 4997).
	VariantShr (see page 4993)	Performs an **shr** (shift right) operation between two Variants (see page 4997).
	VariantShr2 (see page 4994)	Performs an **shr** (shift right) operation between two Variants (see page 4997).
	VariantSub (see page 4994)	Subtracts two Variants (see page 4997).
	VariantSub2 (see page 4995)	Subtracts two Variants (see page 4997).
	VariantToUnicodeString (see page 4995)	Converts a Variant (see page 3430) to an Unicode string.
	VariantXor (see page 4996)	Performs a **xor** (exclusive or) operation between two Variants (see page 4997).
	VariantXor2 (see page 4996)	Performs a **xor** (exclusive or) operation between two Variants (see page 4997).

53.1.1 VarHlpr.VariantAdd Function

Adds two Variants (see page 4997).

Pascal

```
procedure VariantAdd(const src: Variant; var dst: Variant);
```

C++

```
VariantAdd(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantAdd adds **src** to **dst** and returns the result as **dst**.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAdd2 (see page 4982)

VariantSub (see page 4994)

VariantSub2 (see page 4995)

53.1.2 VarHlpr.VariantAdd2 Function

Adds two Variants (see page 4997).

Pascal

```
function VariantAdd2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantAdd2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantAdd2 adds **V1** to **V2** and returns a new Variant (see page 3430) holding the result.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAdd (see page 4981)

VariantSub (see page 4994)

VariantSub2 (see page 4995)

53.1.3 VarHlpr.VariantAnd Function

Performs an **and** operation between two Variant (see page 3430) values.

Pascal

```
procedure VariantAnd(const src: Variant; var dst: Variant);
```

C++

```
VariantAnd(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantAnd performs a bit-wise or a logical **and** operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAnd2 (see page 4983)

VariantOr (see page 4990)

VariantOr2 (see page 4991)

53.1.4 VarHlpr.VariantAnd2 Function

Performs an **and** operation between two Variants (see page 4997).

Pascal

```
function VariantAnd2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantAnd2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantAnd2 performs a bit-wise or a logical **and** operation between **V1** and **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAnd (see page 4982)

VariantOr (see page 4990)

VariantOr2 (see page 4991)

53.1.5 VarHlpr.VariantArrayRedim Function

Resizes a Variant (see page 3430) array.

Pascal

```
procedure VariantArrayRedim(var V: Variant; High: Integer);
```

C++

```
VariantArrayRedim(Variant V, int High);
```

File

VarHlpr

Description

VariantArrayRedim resizes the given Variant (see page 3430) array by changing the high bound of the rightmost dimension to the given value.

The input value must be a Variant (see page 3430) array; otherwise an EVariantInvalidArgError exception will be raised.

See Also

VariantGetElement (see page 4987)

VariantPutElement (see page 4992)

53.1.6 VarHlpr.VariantCast Function

Performs a type-cast on a Variant (see page 3430).

Pascal

```
procedure VariantCast(const src: Variant; var dst: Variant; vt: Integer);
```

C++

```
VariantCast(const Variant src, Variant dst, int vt);
```

File

VarHlpr

Description

VariantCast type-casts **src** to the desired type and returns the resulted Variant (see page 3430) as **dst**. **vt** is a value which identifies the type of the destination Variant (see page 3430).

An EVariantTypeCastError exception will be raised if the source cannot be type-casted to the desired type.

See Also

VariantClear (see page 4984)

VarCast

53.1.7 VarHlpr.VariantClear Function

Empties a Variant (see page 3430), so that it is unassigned.

Pascal

```
procedure VariantClear(var V: Variant);
```

C++

```
VariantClear(Variant V);
```

File

VarHlpr

Description

Calling VariantClear is equivalent to assigning the Unassigned constant to the Variant (see page 3430). After calling VariantClear, the VarIsEmpty function returns true, and the VarType function returns varEmpty (see page 3933). Using an unassigned variant in an expression causes an exception to be thrown. Likewise, if you attempt to convert an unassigned Variant (see page 3430) to another type (using VariantCast (see page 4984)), an exception is thrown.

Note: Do not confuse clearing a Variant (see page 3430), which leaves it unassigned, with assigning a Null value. A Null Variant (see page 3430) is still assigned, but has the value Null. Unlike unassigned Variants (see page 4997), Null Variants (see page 4997) can be used in expressions and can be converted to other types of Variants (see page 4997).

See Also

VariantCast (see page 4984)

VarClear (see page 3786)

53.1.8 VarHlpr.VariantCpy Function

Copies a Variant (see page 3430).

Pascal

```
procedure VariantCpy(const src: Variant; var dst: Variant);
```

C++

```
VariantCpy(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantCpy copies the Variant (see page 3430) value contained in **src** to **dst**.

See Also

VariantClear (see page 4984)

VariantCast (see page 4984)

53.1.9 VarHlpr.VariantDiv Function

Divides two Variants (see page 4997).

Pascal

```
procedure VariantDiv(const src: Variant; var dst: Variant);
```

C++

```
VariantDiv(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantDiv divides **src** to **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantDiv2 (see page 4986)

VariantMul (see page 4989)

VariantMul2 (see page 4989)

53.1.10 VarHlpr.VariantDiv2 Function

Divides two Variants (see page 4997).

Pascal

```
function VariantDiv2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantDiv2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantDiv2 divides **V1** to **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantDiv (see page 4985)

VariantMul (see page 4989)

VariantMul2 (see page 4989)

53.1.11 VarHlpr.VariantFromUnicodeString Function

Assigns an Unicode string to a Variant (see page 3430).

Pascal

```
procedure VariantFromUnicodeString(var V: Variant; const Str: UnicodeString);
```

C++

```
VariantFromUnicodeString(Variant V, const UnicodeString Str);
```

File

VarHlpr

Description

VariantFromUnicodeString assigns an Unicode string to a given Variant (see page 3430) parameter. The old value, if there was one, of the Variant (see page 3430) will be lost.

See Also

VariantToUnicodeString (see page 4995)

VariantClear (see page 4984)

VariantCast (see page 4984)

53.1.12 VarHlpr.VariantGetElement Function

Returns an element from a Variant (see page 3430) array.

Pascal

```
function VariantGetElement(const V: Variant; i1: integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer): Variant;
overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer; i4:
integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer; i4:
integer; i5: integer): Variant; overload;
```

C++

```
Variant VariantGetElement(const Variant V, int i1);
Variant VariantGetElement(const Variant V, int i1, int i2);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3, int i4);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3, int i4, int i5);
```

File

VarHlpr

Description

VariantGetElement returns the Variant (see page 3430) value located at the specified indices in the Variant (see page 3430) array **V**.

This function comes in five forms. Each form accepts a Variant (see page 3430) array and a number of indices in that array, one for each dimension.

See Also

VariantPutElement (see page 4992)

VarArrayGet (see page 5079)

VarArrayPut (see page 5081)

53.1.13 VarHlpr.VariantMod Function

Returns the remainder on division of two Variants (see page 4997).

Pascal

```
procedure VariantMod(const src: Variant; var dst: Variant);
```

C++

```
VariantMod(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantMod calculates the remainder on division of **src** by **dst** and returns the result as **dst**.

The input values must be convertible to integer numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMod2 (see page 4988)

VariantDiv (see page 4985)

VariantDiv2 (see page 4986)

53.1.14 VarHlpr.VariantMod2 Function

Returns the remainder on division of two Variants (see page 4997).

Pascal

```
function VariantMod2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantMod2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantMod2 calculates the remainder on division of **V1** by **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to integer numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMod (see page 4988)

VariantDiv (see page 4985)

VariantDiv2 (see page 4986)

53.1.15 VarHlpr.VariantMul Function

Multiplies two Variants (see page 4997).

Pascal

```
procedure VariantMul(const src: Variant; var dst: Variant);
```

C++

```
VariantMul(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantMul multiplies **src** to **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMul2 (see page 4989)

VariantDiv (see page 4985)

VariantDiv2 (see page 4986)

53.1.16 VarHlpr.VariantMul2 Function

Multiplies two Variants (see page 4997).

Pascal

```
function VariantMul2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantMul2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantMul2 multiplies **V1** to **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMul (see page 4989)

VariantDiv (see page 4985)

VariantDiv2 (see page 4986)

53.1.17 VarHlpr.VariantNeg Function

Inverts the sign of a Variant (see page 3430).

Pascal

```
function VariantNeg(const V1: Variant): Variant;
```

C++

```
Variant VariantNeg(const Variant V1);
```

File

VarHlpr

Description

VariantNeg will invert the sign of **V1** from positive to negative and vice-versa.

The input parameter must be convertible to a numerical type; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantNot (see page 4990)

53.1.18 VarHlpr.VariantNot Function

Performs a **not** operation on a Variant (see page 3430).

Pascal

```
function VariantNot(const V1: Variant): Variant;
```

C++

```
Variant VariantNot(const Variant V1);
```

File

VarHlpr

Description

VariantNot performs a bit-wise or a logical **not** operation on **V1** and returns a new Variant (see page 3430) holding the result.

The input value must be convertible to a numerical or boolean type; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantNeg (see page 4990)

53.1.19 VarHlpr.VariantOr Function

Performs an **or** operation between two Variants (see page 4997).

Pascal

```
procedure VariantOr(const src: Variant; var dst: Variant);
```

C++

```
VariantOr(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantOr performs a bit-wise or a logical **or** operation between **src** and **dst** and returns the result as **dst**.

Given input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantOr2 (see page 4991)

VariantAnd (see page 4982)

VariantAnd2 (see page 4983)

53.1.20 VarHlpr.VariantOr2 Function

Performs an **or** operation between two Variants (see page 4997).

Pascal

```
function VariantOr2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantOr2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantOr2 performs a bit-wise or a logical **or** operation between **V1** and **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantOr (see page 4990)

VariantAnd (see page 4982)

VariantAnd2 (see page 4983)

53.1.21 VarHlpr.VariantPutElement Function

Sets the value of an element in a Variant (see page 3430) array.

Pascal

```
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer);
overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer; i4: integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer; i4: integer; i5: integer); overload;
```

C++

```
VariantPutElement(Variant V, const Variant data, int i1);
VariantPutElement(Variant V, const Variant data, int i1, int i2);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3, int i4);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3, int i4, int i5);
```

File

VarHlpr

Description

VariantPutElement sets the Variant (see page 3430) value located at the specified indices in the Variant (see page 3430) array **V**.

This function comes in five forms. Each form accepts a Variant (see page 3430) array, a Variant (see page 3430) value to assign, and a number of indices in that array, one for each dimension.

See Also

VariantGetElement (see page 4987)

VarArrayGet (see page 5079)

VarArrayPut (see page 5081)

53.1.22 VarHlpr.VariantShl Function

Pascal

```
procedure VariantShl(const src: Variant; var dst: Variant);
```

C++

```
VariantShl(const Variant src, Variant dst);
```

File

VarHlpr

Description

This is function VarHlpr.VariantShl.

53.1.23 VarHlpr.VariantShl2 Function

Performs an **shl** (shift left) operation between two Variants (see page 4997).

Pascal

```
function VariantShl2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantShl2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantShl2 performs a bit-wise **shl** (shift left) operation between **V1** and **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShl (see page 4992)

VariantShr (see page 4993)

VariantShr2 (see page 4994)

53.1.24 VarHlpr.VariantShr Function

Performs an **shr** (shift right) operation between two Variants (see page 4997).

Pascal

```
procedure VariantShr(const src: Variant; var dst: Variant);
```

C++

```
VariantShr(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantShr performs a bit-wise **shr** (shift right) operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShr2 (see page 4994)

VariantShl (see page 4992)

VariantShl2 (see page 4993)

53.1.25 VarHlpr.VariantShr2 Function

Performs an **shr** (shift right) operation between two Variants (see page 4997).

Pascal

```
function VariantShr2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantShr2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantShr2 performs a bit-wise **shr** (shift right) operation between **V1** and **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShr (see page 4993)

VariantShl (see page 4992)

VariantShl2 (see page 4993)

53.1.26 VarHlpr.VariantSub Function

Subtracts two Variants (see page 4997).

Pascal

```
procedure VariantSub(const src: Variant; var dst: Variant);
```

C++

```
VariantSub(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantSub subtracts **src** from **dst** and returns the result as **dst**.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantSub2 (see page 4995)

VariantAdd (see page 4981)

VariantAdd2 (see page 4982)

53.1.27 VarHlpr.VariantSub2 Function

Subtracts two Variants (see page 4997).

Pascal

```
function VariantSub2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantSub2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantSub2 subtracts **V1** from **V2** and returns a new Variant (see page 3430) holding the result.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantSub (see page 4994)

VariantAdd (see page 4981)

VariantAdd2 (see page 4982)

53.1.28 VarHlpr.VariantToUnicodeString Function

Converts a Variant (see page 3430) to an Unicode string.

Pascal

```
procedure VariantToUnicodeString(const V: Variant; var Str: UnicodeString);
```

C++

```
VariantToUnicodeString(const Variant V, UnicodeString Str);
```

File

VarHlpr

Description

VariantToUnicodeString converts the parameter **V** to an Unicode string and places the result into **Str (see page 3764)**.

See Also

VariantFromUnicodeString (see page 4986)

VariantClear (see page 4984)

VariantCast (see page 4984)

53.1.29 VarHlpr.VariantXor Function

Performs a **xor** (exclusive or) operation between two Variants (see page 4997).

Pascal

```
procedure VariantXor(const src: Variant; var dst: Variant);
```

C++

```
VariantXor(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantXor performs a bit-wise **xor** (exclusive or) operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantXor2 (see page 4996)

VariantOr (see page 4990)

VariantOr2 (see page 4991)

53.1.30 VarHlpr.VariantXor2 Function

Performs a **xor** (exclusive or) operation between two Variants (see page 4997).

Pascal

```
function VariantXor2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantXor2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantXor2 performs a bit-wise **xor** (exclusive or) operation between **V1** and **V2** and returns a new Variant (see page 3430) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantXor (see page 4996)

VariantOr (see page 4990)

VariantOr2 (see page 4991)

54 Variants Namespace

This is namespace Variants.

Topics

Name	Description
Classes (see page 5002)	The following table lists classes in this documentation.
Interfaces (see page 5045)	The following table lists interfaces in this documentation.
Functions (see page 5048)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5105)	The following table lists structs, records, enums in this documentation.
Types (see page 5108)	The following table lists types in this documentation.
Variables (see page 5110)	The following table lists variables in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 5002)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 5003)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 5004)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 5004)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 5005)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 5005)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 5006)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 5075) variant operations.
	EVariantInvalidOpError (see page 5007)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 5007)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 5008)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 5008)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 5009)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 5010)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 5010)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 5011)	EVariantUnexpectedError is the exception class for unexpected variant errors.

	TCustomVariantType (see page 5011)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 5037)	TInvokeableVariantType is the base class for custom Variant (see page 3430) types that implement custom Variants with properties and methods.

Functions

	Name	Description
	_DispInvoke (see page 5050)	This is function Variants._DispInvoke.
	_OleVarFromInt (see page 5051)	This is function Variants._OleVarFromInt.
	_OleVarFromInt64 (see page 5051)	This is function Variants._OleVarFromInt64.
	_OleVarFromLStr (see page 5051)	This is function Variants._OleVarFromLStr.
	_OleVarFromPStr (see page 5052)	This is function Variants._OleVarFromPStr.
	_OleVarFromUInt64 (see page 5052)	This is function Variants._OleVarFromUInt64.
	_OleVarFromUStr (see page 5052)	This is function Variants._OleVarFromUStr.
	_OleVarFromVar (see page 5053)	This is function Variants._OleVarFromVar.
	_VarAdd (see page 5053)	This is function Variants._VarAdd.
	_VarAddRef (see page 5053)	This is function Variants._VarAddRef.
	_VarAnd (see page 5054)	This is function Variants._VarAnd.
	_VarArrayGet (see page 5054)	This is function Variants._VarArrayGet.
	_VarArrayPut (see page 5054)	This is function Variants._VarArrayPut.
	_VarArrayRedim (see page 5055)	This is function Variants._VarArrayRedim.
	_VarCast (see page 5055)	This is function Variants._VarCast.
	_VarCastOle (see page 5055)	This is function Variants._VarCastOle.
	_VarClear (see page 5056)	This is function Variants._VarClear.
	_VarClr (see page 5056)	This is function Variants._VarClr.
	_VarCmpEQ (see page 5056)	This is function Variants._VarCmpEQ.
	_VarCmpGE (see page 5057)	This is function Variants._VarCmpGE.
	_VarCmpGT (see page 5057)	This is function Variants._VarCmpGT.
	_VarCmpLE (see page 5057)	This is function Variants._VarCmpLE.
	_VarCmpLT (see page 5058)	This is function Variants._VarCmpLT.
	_VarCmpNE (see page 5058)	This is function Variants._VarCmpNE.
	_VarCopy (see page 5058)	This is function Variants._VarCopy.
	_VarCopyNoInd (see page 5059)	This is function Variants._VarCopyNoInd.
	_VarDiv (see page 5059)	This is function Variants._VarDiv.
	_VarFromBool (see page 5059)	This is function Variants._VarFromBool.
	_VarFromCurr (see page 5060)	This is function Variants._VarFromCurr.
	_VarFromDisp (see page 5060)	This is function Variants._VarFromDisp.
	_VarFromDynArray (see page 5060)	This is function Variants._VarFromDynArray.
	_VarFromInt (see page 5061)	This is function Variants._VarFromInt.
	_VarFromInt64 (see page 5061)	This is function Variants._VarFromInt64.
	_VarFromIntf (see page 5061)	This is function Variants._VarFromIntf.
	_VarFromLStr (see page 5062)	This is function Variants._VarFromLStr.
	_VarFromPStr (see page 5062)	This is function Variants._VarFromPStr.
	_VarFromReal (see page 5062)	This is function Variants._VarFromReal.
	_VarFromTDateTime (see page 5063)	This is function Variants._VarFromTDateTime.
	_VarFromUInt64 (see page 5063)	This is function Variants._VarFromUInt64.
	_VarFromUStr (see page 5063)	This is function Variants._VarFromUStr.
	_VarFromWChar (see page 5064)	This is function Variants._VarFromWChar.
	_VarFromWStr (see page 5064)	This is function Variants._VarFromWStr.
	_VarMod (see page 5064)	This is function Variants._VarMod.
	_VarMul (see page 5065)	This is function Variants._VarMul.
	_VarNeg (see page 5065)	This is function Variants._VarNeg.
	_VarNot (see page 5065)	This is function Variants._VarNot.
	_VarOr (see page 5066)	This is function Variants._VarOr.
	_VarRDiv (see page 5066)	This is function Variants._VarRDiv.
	_VarShl (see page 5066)	This is function Variants._VarShl.

	_VarShr (see page 5067)	This is function Variants._VarShr.
	_VarSub (see page 5067)	This is function Variants._VarSub.
	_VarToBool (see page 5067)	This is function Variants._VarToBool.
	_VarToCurrency (see page 5068)	This is function Variants._VarToCurrency.
	_VarToDisp (see page 5068)	This is function Variants._VarToDisp.
	_VarToDynArray (see page 5068)	This is function Variants._VarToDynArray.
	_VarToInt64 (see page 5069)	This is function Variants._VarToInt64.
	_VarToInteger (see page 5069)	This is function Variants._VarToInteger.
	_VarToIntf (see page 5069)	This is function Variants._VarToIntf.
	_VarToLStr (see page 5070)	This is function Variants._VarToLStr.
	_VarToPStr (see page 5070)	This is function Variants._VarToPStr.
	_VarToReal (see page 5070)	This is function Variants._VarToReal.
	_VarToUInt64 (see page 5071)	This is function Variants._VarToUInt64.
	_VarToUStr (see page 5071)	This is function Variants._VarToUStr.
	_VarToWStr (see page 5071)	This is function Variants._VarToWStr.
	_VarXor (see page 5072)	This is function Variants._VarXor.
	DynArrayFromVariant (see page 5072)	Creates a dynamic array from a Variant (see page 3430).
	DynArrayToVariant (see page 5072)	Creates a Variant (see page 3430) array from a dynamic array.
	FindCustomVariantType (see page 5073)	Retrieves the object that implements a custom Variant (see page 3430) type.
	FindCustomVariantType (see page 5073)	Retrieves the object that implements a custom Variant (see page 3430) type.
	FindVarData (see page 5074)	Returns pointer to variant's data.
	HandleConversionException (see page 5074)	Handles exceptions during variant type conversions.
	Null (see page 5075)	Returns a Null variant.
	Unassigned (see page 5076)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 5077)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 5077)	Creates a variant array.
	VarArrayCreateError (see page 5078)	Indicates an array creation error.
	VarArrayDimCount (see page 5078)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 5079)	Returns a Variant (see page 3430) that represents a single value from a multi-dimensional Variant (see page 3430) array.
	VarArrayHighBound (see page 5079)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 5080)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 5080)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 5081)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 5081)	Sets the value of a single cell in a multi-dimensional Variant (see page 3430) array.
	VarArrayRef (see page 5082)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 5083)	Unlocks a variant array.
	VarAsError (see page 5083)	Converts a HRESULT (see page 3826) into an Error (see page 3662) variant.
	VarAsType (see page 5083)	Converts a variant to specified type.
	VarCastError (see page 5084)	Raises an EVariantTypeCastError (see page 5010) exception.
	VarCastError (see page 5084)	Raises an EVariantTypeCastError (see page 5010) exception.
	VarCheckEmpty (see page 5084)	Raises an exception if a specified variant's value is Unassigned (see page 5076).
	VarCompareValue (see page 5085)	Returns the relationship between Variants.
	VarCopyNoInd (see page 5086)	Copies a Variant (see page 3430).
	VarEnsureRange (see page 5086)	Returns a Variant (see page 3430) with a value in a specified range.
	VarFromDateTime (see page 5087)	Returns a variant containing a specified data and time.
	VarInRange (see page 5087)	Indicates whether a Variant (see page 3430)'s value is in a specified range.
	VarInvalidNullOp (see page 5088)	Indicate variant operation on null.
	VarInvalidOp (see page 5088)	Indicate invalid variant operation.
	VarIsArray (see page 5089)	Indicates whether the specified variant is an array.
	VarIsArray (see page 5089)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 5089)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 5090)	Indicates whether the specified variant has an undefined value.

	VarIsCustom (see page 5090)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 5091)	Indicates whether the specified variant is unassigned.
	VarIsEmptyParam (see page 5092)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 5092)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3826).
	VarIsError (see page 5092)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3826).
	VarIsFloat (see page 5093)	Indicates whether the specified variant represents a floating-point value.
	VarIsNull (see page 5093)	Indicates whether the specified variant is Null (see page 5075).
	VarIsNumeric (see page 5094)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 5094)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 5095)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 5095)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 5095)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 5096)	Raises an EVariantOverflowError (see page 5009) exception.
	VarRangeCheckError (see page 5096)	Raises an EVariantRangeCheckError (see page 5010) exception.
	VarResultCheck (see page 5097)	Raises variant exception based on HRESULT (see page 3826) value.
	VarResultCheck (see page 5097)	Raises variant exception based on HRESULT (see page 3826) value.
	VarSameValue (see page 5097)	Indicates whether two Variants have equivalent values.
	VarSupports (see page 5098)	Indicates whether the Variant (see page 3430)'s value is an interface that supports a specified interface.
	VarSupports (see page 5098)	Indicates whether the Variant (see page 3430)'s value is an interface that supports a specified interface.
	VarToDateTime (see page 5098)	Converts a specified variant to a TDateTime (see page 3335) value.
	VarToStr (see page 5099)	Converts a variant's value to a string.
	VarToStrDef (see page 5099)	Converts a variant's value to a string.
	VarToWideStr (see page 5100)	Converts a variant's value to a WideString (see page 3476).
	VarToWideStrDef (see page 5100)	Converts a variant's value to a WideString (see page 3476).
	VarType (see page 5101)	Returns the type code of a specified variant.
	VarTypeAsText (see page 5102)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 5103)	Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.
	VarTypeIsValidElementType (see page 5103)	Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.

Interfaces

	Name	Description
	IVarInstanceReference (see page 5045)	This is class Variants.IVarInstanceReference.
	IVarInvokeable (see page 5046)	This is class Variants.IVarInvokeable.

Structs, Records, Enums

	Name	Description
	TBooleanToStringRule (see page 5105)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5110).
	TNullCompareRule (see page 5105)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 5106)	This is record Variants.TVarCompareResult.
	TVariantRelationship (see page 5106)	TVariantRelationship describes the results of comparing two Variants.

Types

Name	Description
TAnyProc (see page 5108)	This is type Variants.TAnyProc.
TCustomVariantTypeClass (see page 5108)	This is type Variants.TCustomVariantTypeClass.
TVarDataArray (see page 5109)	This is type Variants.TVarDataArray.
TVarDispProc (see page 5109)	This is type Variants.TVarDispProc.

Variables

Name	Description
BooleanToStringRule (see page 5110)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 5111)	This is variable Variants.ChangeAnyProc.
ClearAnyProc (see page 5111)	This is variable Variants.ClearAnyProc.
EmptyParam (see page 5111)	Contains an OleVariant (see page 3281) that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 5112)	Specifies string value of Null (see page 5075) variants.
NullEqualityRule (see page 5112)	Specifies rule for Null (see page 5075) variant equality comparison.
NullMagnitudeRule (see page 5113)	Specifies rule for Null (see page 5075) variant magnitude comparison.
NullStrictConvert (see page 5114)	Specifies whether Null (see page 5075) conversion is an error.
OleVariantInt64AsDouble (see page 5114)	This is variable Variants.OleVariantInt64AsDouble.
PackVarCreation (see page 5115)	Specifies whether space-efficient Variant (see page 3430) Types (see page 4683) are preferred.
RefAnyProc (see page 5115)	This is variable Variants.RefAnyProc.
VarDispProc (see page 5115)	Provides the support for IDispatch (see page 3498)-based Automation.

54.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 5002)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 5003)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 5004)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 5004)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 5005)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 5005)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 5006)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 5075) variant operations.
	EVariantInvalidOpError (see page 5007)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 5007)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 5008)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 5008)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 5009)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 5010)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 5010)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 5011)	EVariantUnexpectedError is the exception class for unexpected variant errors.
	TCustomVariantType (see page 5011)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 5037)	TInvokeableVariantType is the base class for custom Variant (see page 3430) types that implement custom Variants (see page 4997) with properties and methods.

54.1.1 EVariantArrayCreateError Class

EVariantArrayCreateError is the exception class for failure to create or resize a variant array.

Class Hierarchy

Pascal

```
EVariantArrayCreateError = class(EVariantError);
```

C++

```
class EVariantArrayCreateError : public EVariantError;
```

File

Variants

Description

EVariantArrayCreateError is raised when an attempt to create or resize a variant array fails. To raise this exception programmatically, call VarArrayCreateError (see page 5078)

See Also

VarArrayCreateError (see page 5078)

TVarData (see page 3813)

VarArrayCreate (see page 5077)

VarArrayOf (see page 5081)

VarArrayRedim

TCustomVariantType (see page 5011)

VarArrayCreateError (see page 5078)

54.1.2 EVariantArrayLockedError Class

EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

Class Hierarchy

Pascal

```
EVariantArrayLockedError = class(EVariantError);
```

C++

```
class EVariantArrayLockedError : public EVariantError;
```

File

Variants

Description

EVariantArrayLockedError is raised when an operation on a variant array fails because the array is locked or fixed.

See Also

TVarData (see page 3813)

TVariantManager

VarAsType (see page 5083)

VarArrayLock (see page 5080)

VarArrayUnlock (see page 5083)

VarArrayOf (see page 5081)

VarArrayRedim

54.1.3 EVariantBadIndexError Class

EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.

Class Hierarchy

Pascal

```
EVariantBadIndexError = class(EVariantError);
```

C++

```
class EVariantBadIndexError : public EVariantError;
```

File

Variants

Description

EVariantBadIndexError is raised when an operation on a variant array fails because an index value is out of bounds.

See Also

TVarData (see page 3813)

TVariantManager

VarAsType (see page 5083)

VarArrayCreate (see page 5077)

VarArrayHighBound (see page 5079)

VarArrayLowBound (see page 5080)

VarArrayOf (see page 5081)

VarArrayRedim

TCustomVariantType (see page 5011)

54.1.4 EVariantBadVarTypeError Class

EVariantBadVarTypeError is the exception class for variant type errors.

Class Hierarchy

Pascal

```
EVariantBadVarTypeError = class(EVariantError);
```

C++

```
class EVariantBadVarTypeError : public EVariantError;
```

File

Variants

Description

EVariantBadVarTypeError is raised when a variant operation fails because of a value that isn't a valid variant type.

See Also

TVarData (see page 3813)

TVariantManager

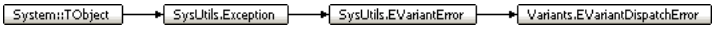
VarAsType (see page 5083)

TCustomVariantType (see page 5011)

54.1.5 EVariantDispatchError Class

EVariantDispatchError is the exception class for failed dispatch attempts.

Class Hierarchy



Pascal

```
EVariantDispatchError = class(EVariantError);
```

C++

```
class EVariantDispatchError : public EVariantError;
```

File

Variants

Description

EVariantDispatchError is raised when a variant type is unable to dispatch a method call. Do not directly instantiate EVariantDispatchError. Instead call the RaiseDispError method of TCustomVariant.

See Also

TVarData (see page 3813)

TVariantManager

TCustomVariantType (see page 5011)

TCustomVariantType (see page 5011)

54.1.6 EVariantInvalidArgError Class

EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.

Class Hierarchy

Pascal

```
EVariantInvalidArgError = class(EVariantError);
```

C++

```
class EVariantInvalidArgError : public EVariantError;
```

File

Variants

Description

EVariantInvalidArgError is raised when a variant operation fails because it was passed an inappropriate value.

See Also

TVarData (see page 3813)

TVariantManager

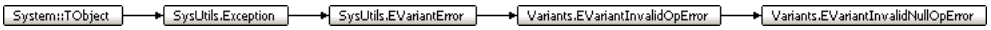
VarAsType (see page 5083)

TCustomVariantType (see page 5011)

54.1.7 EVariantInvalidNullOpError Class

EVariantInvalidNullOpError is the exception class for invalid Null (see page 5075) variant operations.

Class Hierarchy



Pascal

```
EVariantInvalidNullOpError = class(EVariantInvalidOpError);
```

C++

```
class EVariantInvalidNullOpError : public EVariantInvalidOpError;
```

File

Variants

Description

EVariantInvalidNullOpError is raised when a variant operation fails because the operation is not valid for Null (see page 5075) variants. To raise this exception programmatically, call VarInvalidNullOp (see page 5088).

Note: Do not raise EVariantInvalidNullOpError unless NullEqualityRule (see page 5112) or NullMagnitudeRule (see page 5113) is ncrError, and a corresponding operation has been attempted.

See Also

TVarData (see page 3813)

TVariantManager

TCustomVariantType (see page 5011)

VarInvalidNullOp (see page 5088)

Null (see page 5075)

NullEqualityRule (see page 5112)

NullMagnitudeRule (see page 5113)

54.1.8 EVariantInvalidOpError Class

EVariantInvalidOpError is the exception class for invalid variant operations.

Class Hierarchy

Pascal

```
EVariantInvalidOpError = class(EVariantError);
```

C++

```
class EVariantInvalidOpError : public EVariantError;
```

File

Variants

Description

EVariantInvalidOpError is raised when a variant operation fails because the operands are incompatible with the specific operation. To raise this exception programmatically, call VarInvalidOp (see page 5088).

See Also

TVarData (see page 3813)

TVariantManager

TCustomVariantType (see page 5011)

VarInvalidOp (see page 5088)

54.1.9 EVariantNotAnArrayError Class

EVariantNotAnArrayError is the exception class inappropriate variant array operations.

Class Hierarchy

Pascal

```
EVariantNotAnArrayError = class(EVariantError);
```

C++

```
class EVariantNotAnArrayError : public EVariantError;
```

File

Variants

Description

EVariantNotAnArrayError is raised when an array operation is attempted on a variant that does not contain an array.

See Also

VarArrayCreateError (see page 5078)

TVarData (see page 3813)

VarArrayCreate (see page 5077)

VarArrayOf (see page 5081)

VarArrayRedim

TCustomVariantType (see page 5011)

VarArrayCreateError (see page 5078)

54.1.10 EVariantNotImplError Class

EVariantNotImplError is the exception class for unimplemented variant operations.

Class Hierarchy

Pascal

```
EVariantNotImplError = class(EVariantError);
```

C++

```
class EVariantNotImplError : public EVariantError;
```

File

Variants

Description

EVariantNotImplError is raised when an operation is attempted on a variant type that does not implement that operation.

See Also

TVarData (see page 3813)

TVariantManager

TCustomVariantType (see page 5011)

54.1.11 EVariantOutOfMemoryError Class

EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.

Class Hierarchy

Pascal

```
EVariantOutOfMemoryError = class(EVariantError);
```

C++

```
class EVariantOutOfMemoryError : public EVariantError;
```

File

Variants

Description

EVariantOutOfMemoryError is raised when a variant operation fails because of a failure to allocate necessary memory.

See Also

TVarData (see page 3813)

TVariantManager

VarArrayCreate (see page 5077)

VarArrayOf (see page 5081)

VarArrayRedim

VarToDateTime (see page 5098)

TCustomVariantType (see page 5011)

EOutOfMemory (see page 4008)

54.1.12 EVariantOverflowError Class

EVariantOverflowError is the exception class for overflow in a variant operation.

Class Hierarchy

Pascal

```
EVariantOverflowError = class(EVariantError);
```

C++

```
class EVariantOverflowError : public EVariantError;
```

File

Variants

Description

EVariantOverflowError is raised when a variant operation fails because the result was too large. To raise this exception programmatically, call VarOverflowError (see page 5096).

See Also

TVarData (see page 3813)

TVariantManager

VarAsType (see page 5083)

VarToDateTime (see page 5098)

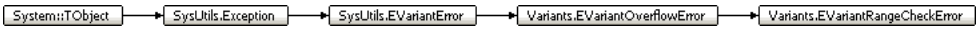
TCustomVariantType (see page 5011)

VarOverflowError (see page 5096)

54.1.13 EVariantRangeCheckError Class

EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.

Class Hierarchy



Pascal

```
EVariantRangeCheckError = class(EVariantOverflowError);
```

C++

```
class EVariantRangeCheckError : public EVariantOverflowError;
```

File

Variants

Description

EVariantRangeCheckError is raised when a variant operation fails because the result is out-of-range for the destination variant type. To raise this exception programmatically, call VarRangeCheckError (see page 5096).

See Also

TVarData (see page 3813)

TVariantManager

VarAsType (see page 5083)

VarToDateTime (see page 5098)

TCustomVariantType (see page 5011)

VarRangeCheckError (see page 5096)

54.1.14 EVariantTypeCastError Class

EVariantTypeCastError is the exception class for variant type cast errors.

Class Hierarchy

Pascal

```
EVariantTypeCastError = class(EVariantError);
```

C++

```
class EVariantTypeCastError : public EVariantError;
```

File

Variants

Description

EVariantTypeCastError is raised when a variant operation fails because a value could not be cast to the destination variant type. To raise this exception programmatically, call VarCastError (see page 5084).

See Also

TVarData (see page 3813)

TVariantManager

VarAsType (see page 5083)

VarToDateTime (see page 5098)

TCustomVariantType (see page 5011)

VarOverflowError (see page 5096)

54.1.15 EVariantUnexpectedError Class

EVariantUnexpectedError is the exception class for unexpected variant errors.

Class Hierarchy

Pascal

```
EVariantUnexpectedError = class(EVariantError);
```

C++

```
class EVariantUnexpectedError : public EVariantError;
```

File

Variants

Description

EVariantUnexpectedError is raised when a variant operation generates an unexpected error.

See Also

TVarData (see page 3813)

TVariantManager

TCustomVariantType (see page 5011)

54.1.16 TCustomVariantType Class

TCustomVariantType is the base class for Delphi custom variant types.

Class Hierarchy

Pascal

```
TCustomVariantType = class(TObject, IInterface);
```

C++

```
class TCustomVariantType : public TObject, public IInterface;
```

File

Variants

Description

Use TCustomVariantType to define new data types for Delphi Variants (see page 4997). TCustomVariantType descendants do not represent the Variants (see page 4997) themselves. Rather, they act as utility classes that the custom Variants (see page 4997) call on to perform their operations.

To create a Variant (see page 3430) type,

1.Map the storage of the Variant (see page 3430)'s data on to the TVarData (see page 3813) record.

2.Write (see page 3789) a method that creates instances of your custom Variant (see page 3430). This fills in the Variant (see page 3430)'s data as defined in step 1.

3.Declare a class that descends from TCustomVariantType.

4.Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically BinaryOp (see page 5018), UnaryOp (see page 5029), CompareOp (see page 5021) or Compare (see page 5020), Cast (see page 5018), CastTo (see page 5019), Clear (see page 5019), Copy (see page 5022), IsClear (see page 5023), LeftPromotion (see page 5023), and RightPromotion (see page 5027)).

5.In the initialization section of the unit that defines your TCustomVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant (see page 3430)-handling system so that the new Variant (see page 3430) type is enabled.

6.In the finalization section of the unit that defines your TCustomVariantType descendant, free the instance of your class. This automatically unregisters the Variant (see page 3430) type.

Note: C++ can use Custom Variant (see page 3430) types, but TCustomVariantType descendants must be implemented in Delphi.

For an example, see the FMTBcd unit, which implements a custom Variant (see page 3430) type for binary-coded decimal.

Tip: To create a custom Variant (see page 3430) that allows users to call methods or procedures and to get or set property values, use TInvokeableVariantType (see page 5037) as a base class instead.

See Also

IDH

TInvokeableVariantType (see page 5037)

Creating a Class to Enable the Custom Variant Type

54.1.16.1 TCustomVariantType Members

The following tables list the members exposed by TCustomVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 5017)	Increments the reference count for this interface.

	_Release (see page 5017)	Decrements the reference count for this interface.
V	DispInvoke (see page 5022)	Invokes a method call when the Variant (see page 3430) supports such operations.
V	LeftPromotion (see page 5023)	Indicates whether the left argument to a binary operation should be coerced to a different type.
V	OlePromotion (see page 5024)	Indicates the type to which a Variant (see page 3430) value must be cast to make it Automation-compatible.
V	QueryInterface (see page 5025)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5025)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5026)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5026)	Generates a system error indicating an invalid operation.
V	RightPromotion (see page 5027)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5028)	Clears a TVarData (see page 3813) record by re-initializing it.
	SimplisticCopy (see page 5028)	Performs a simple block copy of one TVarData (see page 3813) record to another.
	VarDataCast (see page 5029)	Performs a cast from one Variant (see page 3430) type to another.
	VarDataCastTo (see page 5030)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5030)	Performs a cast to a Variant (see page 3430) type of varOleStr (see page 3936).
	VarDataClear (see page 5031)	Clears the data from a built-in Variant (see page 3430) type.
	VarDataCopy (see page 5031)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataCopyNoInd (see page 5032)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataFromOleStr (see page 5032)	Creates a TVarData (see page 3813) record to represent a Unicode string.
	VarDataFromStr (see page 5033)	Creates a TVarData (see page 3813) record to represent a string.
	VarDataInit (see page 5033)	Initializes a TVarData (see page 3813) record.
	VarDataIsArray (see page 5033)	Indicates whether a TVarData (see page 3813) record represents the data from a Variant (see page 3430) array.
	VarDataIsByRef (see page 5034)	Indicates whether a TVarData (see page 3813) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5034)	Indicates whether a TVarData (see page 3813) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5034)	Indicates whether a TVarData (see page 3813) record represents a floating-point value.
	VarDataIsNumeric (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a numeric value.
	VarDataIsOrdinal (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a ordinal value.
	VarDataIsStr (see page 5036)	Indicates whether a TVarData (see page 3813) record represents a string (normal or Unicode).
	VarDataToStr (see page 5036)	Returns a string that represents the value from a TVarData (see page 3813) record.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
V	BinaryOp (see page 5018)	Implements any binary operations of which the custom Variant (see page 3430) type is capable.
V	Cast (see page 5018)	Converts the data from another Variant (see page 3430) type so that it represents this custom type's data format.
V	CastTo (see page 5019)	This is CastTo, a member of class TCustomVariantType.
V	CastToOle (see page 5019)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 5019)	Sets the data for this custom Variant (see page 3430) type to a blank or unassigned value.
	Compare (see page 5020)	Implements comparisons that involve the custom Variant (see page 3430) type.
	CompareOp (see page 5021)	Implements any comparison operations of which the custom Variant (see page 3430) type is capable.
	Copy (see page 5022)	Copies the data from a Variant (see page 3430) of this custom type.
	Create (see page 5014)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 5015)	Frees an instance of TCustomVariantType (see page 5011).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 5023)	Indicates whether the data from a custom Variant (see page 3430) represents a blank or unassigned value.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnaryOp (see page 5029)	Implements any unary operations of which the custom Variant (see page 3430) type is capable.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	VarType (see page 5037)	Indicates the Variant (see page 3430) type code for the custom Variant (see page 3430) type.

54.1.16.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TCustomVariantType.Create () (see page 5015)	Creates an instance of TCustomVariantType (see page 5011).
	TCustomVariantType.Create (TVarType) (see page 5015)	Creates an instance of TCustomVariantType (see page 5011).

54.1.16.2.1 TCustomVariantType.Create Constructor ()

Creates an instance of TCustomVariantType (see page 5011).

Pascal

```
constructor Create; overload;
constructor Create(RequestedVarType: TVarType); overload;
```

C++

```
__fastcall TCustomVariantType();
__fastcall TCustomVariantType(TVarType RequestedVarType);
```

Description

Call Create in the initialization section of the unit that defines your TCustomVariantType (see page 5011) descendant to instantiate a single instance of the custom Variant (see page 3430) type. This single instance automatically registers itself with the Variant (see page 3430) system so that the custom Variant (see page 3430) type is enabled in the application.

The RequestedVarType parameter specifies a type code for the new custom Variant (see page 3430) type. It must be in the range from $010F to $0FFF.

When called with no parameters, the constructor generates a new type code for the custom Variant (see page 3430) type.

The TCustomVariantType (see page 5011) constructor raises an exception if there is a problem with the new Variant (see page 3430) type code. Such problems include

The RequestedVarType parameter specifies a type code that is already used by another Variant (see page 3430) type.

The RequestedVarType parameter specifies a type code that is not in the valid range for custom Variants (see page 4997).

There are too many custom Variants (see page 4997) already defined to allow another one.

See Also

VarType (see page 5037)

54.1.16.3 TCustomVariantType.Destroy Destructor

Frees an instance of TCustomVariantType (see page 5011).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomVariantType();
```

Description

To free a TCustomVariantType (see page 5011) instance, use the Free method instead. Free checks that the instance is not already nil before calling Destroy.

Free the single instance of a TCustomVariantType (see page 5011) descendant from the finalization section of the unit that implements the class.

See Also

Create (see page 5015)

System::TObject::Free (see page 3382)

54.1.16.4 TCustomVariantType Methods

The methods of the TCustomVariantType class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5017)	Increments the reference count for this interface.
	_Release (see page 5017)	Decrements the reference count for this interface.
V	DispInvoke (see page 5022)	Invokes a method call when the Variant (see page 3430) supports such operations.
V	LeftPromotion (see page 5023)	Indicates whether the left argument to a binary operation should be coerced to a different type.
V	OlePromotion (see page 5024)	Indicates the type to which a Variant (see page 3430) value must be cast to make it Automation-compatible.
V	QueryInterface (see page 5025)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5025)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5026)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5026)	Generates a system error indicating an invalid operation.
V	RightPromotion (see page 5027)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5028)	Clears a TVarData (see page 3813) record by re-initializing it.
	SimplisticCopy (see page 5028)	Performs a simple block copy of one TVarData (see page 3813) record to another.
	VarDataCast (see page 5029)	Performs a cast from one Variant (see page 3430) type to another.
	VarDataCastTo (see page 5030)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5030)	Performs a cast to a Variant (see page 3430) type of varOleStr (see page 3936).
	VarDataClear (see page 5031)	Clears the data from a built-in Variant (see page 3430) type.
	VarDataCopy (see page 5031)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataCopyNoInd (see page 5032)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataFromOleStr (see page 5032)	Creates a TVarData (see page 3813) record to represent a Unicode string.
	VarDataFromStr (see page 5033)	Creates a TVarData (see page 3813) record to represent a string.
	VarDataInit (see page 5033)	Initializes a TVarData (see page 3813) record.
	VarDataIsArray (see page 5033)	Indicates whether a TVarData (see page 3813) record represents the data from a Variant (see page 3430) array.
	VarDataIsByRef (see page 5034)	Indicates whether a TVarData (see page 3813) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5034)	Indicates whether a TVarData (see page 3813) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5034)	Indicates whether a TVarData (see page 3813) record represents a floating-point value.
	VarDataIsNumeric (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a numeric value.
	VarDataIsOrdinal (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a ordinal value.
	VarDataIsStr (see page 5036)	Indicates whether a TVarData (see page 3813) record represents a string (normal or Unicode).
	VarDataToStr (see page 5036)	Returns a string that represents the value from a TVarData (see page 3813) record.

Public Methods

	Name	Description
V	BinaryOp (see page 5018)	Implements any binary operations of which the custom Variant (see page 3430) type is capable.
V	Cast (see page 5018)	Converts the data from another Variant (see page 3430) type so that it represents this custom type's data format.
V	CastTo (see page 5019)	This is CastTo, a member of class TCustomVariantType.
V	CastToOle (see page 5019)	This is CastToOle, a member of class TCustomVariantType.

	Clear (see page 5019)	Sets the data for this custom Variant (see page 3430) type to a blank or unassigned value.
	Compare (see page 5020)	Implements comparisons that involve the custom Variant (see page 3430) type.
	CompareOp (see page 5021)	Implements any comparison operations of which the custom Variant (see page 3430) type is capable.
	Copy (see page 5022)	Copies the data from a Variant (see page 3430) of this custom type.
	IsClear (see page 5023)	Indicates whether the data from a custom Variant (see page 3430) represents a blank or unassigned value.
	UnaryOp (see page 5029)	Implements any unary operations of which the custom Variant (see page 3430) type is capable.

54.1.16.4.1 TCustomVariantType._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 3504)

54.1.16.4.2 TCustomVariantType._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 3504)

54.1.16.4.3 TCustomVariantType.BinaryOp Method

Implements any binary operations of which the custom Variant (see page 3430) type is capable.

Pascal

```
procedure BinaryOp(var Left: TVarData; const Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall BinaryOp(TVarData Left, const TVarData Right, const TVarOp Operator);
```

Description

Override BinaryOp to implement any binary operators the custom Variant (see page 3430) type supports.

Left is the value that appears to the left of the operator. BinaryOp changes this value to indicate the result of the operation.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opAdd (see page 3917)	addition
opSubtract (see page 3926)	subtraction
opMultiply (see page 3923)	multiplication
opDivide (see page 3921)	floating-point division
opIntDivide (see page 3922)	integer division
opModulus (see page 3922)	remainder
opShiftLeft (see page 3925)	bitwise left shift
opShiftRight (see page 3925)	bitwise right shift
opAnd (see page 3917)	bitwise and
opOr (see page 3924)	bitwise or
opXor (see page 3926)	bitwise exclusive or

As implemented in TCustomVariantType (see page 5011), BinaryOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 5029)

CompareOp (see page 5021)

LeftPromotion (see page 5023)

RightPromotion (see page 5027)

Implementing Binary Operations

54.1.16.4.4 TCustomVariantType.Cast Method

Converts the data from another Variant (see page 3430) type so that it represents this custom type's data format.

Pascal

```
procedure Cast(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall Cast(TVarData Dest, const TVarData Source);
```

Description

Override Cast to enable other types of Variants (see page 4997) to be cast to this custom Variant (see page 3430) type.

Dest is a TVarData (see page 3813) record that returns the Variant (see page 3430)'s data stored in the format used by this custom Variant (see page 3430) type. The Cast method fills out this record from the information in Source.

Source is a TVarData (see page 3813) record that contains the data from another Variant (see page 3430) type. Check the VType field of Source to identify how it currently stores its data. Then use the other fields to fetch the data and convert it to the format used by this Variant (see page 3430) type.

As implemented in TCustomVariantType (see page 5011), Cast checks whether Source is another custom Variant (see page 3430) type. If so, it calls its CastTo (see page 5019) method, to convert it to this Variant (see page 3430) type. If not, it raises a system error indicating an invalid cast.

See Also

castto (see page 5019)

VarDataCast (see page 5029)

Enabling Casting

54.1.16.4.5 TCustomVariantType.CastTo Method

Pascal

```
procedure CastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
virtual;
```

C++

```
virtual __fastcall CastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
```

Description

This is CastTo, a member of class TCustomVariantType.

54.1.16.4.6 TCustomVariantType.CastToOle Method

Pascal

```
procedure CastToOle(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall CastToOle(TVarData Dest, const TVarData Source);
```

Description

This is CastToOle, a member of class TCustomVariantType.

54.1.16.4.7 TCustomVariantType.Clear Method

Sets the data for this custom Variant (see page 3430) type to a blank or unassigned value.

Pascal

```
procedure Clear(var V: TVarData); virtual; abstract;
```

C++

```
virtual __fastcall Clear(TVarData V) = 0;
```

Description

Override Clear to implement the way this custom Variant (see page 3430) type represents a blank or unassigned value.

V is a TVarData (see page 3813) record that represents the data from a Variant (see page 3430) of this custom type that needs to be cleared.

In TCustomVariantType (see page 5011), Clear is an abstract method. Descendant classes must override this method to provide an implementation. If the custom Variant (see page 3430) type stores its data in a manner that includes memory that needs to be freed, Clear should free that memory. Once any memory is freed, Clear can call the SimplisticClear (see page 5028) method, which re-initializes the TVarData (see page 3813) record, or set the VType field to varEmpty (see page 3933).

See Also

Unassigned (see page 5076)

IsClear (see page 5023)

SimplisticClear (see page 5028)

VarDataClear (see page 5031)

Copying and Clearing Custom Variants

54.1.16.4.8 TCustomVariantType.Compare Method

Implements comparisons that involve the custom Variant (see page 3430) type.

Pascal

```
procedure Compare(const Left: TVarData; const Right: TVarData; var Relationship:
TVarCompareResult); virtual;
```

C++

```
virtual __fastcall Compare(const TVarData Left, const TVarData Right, TVarCompareResult
Relationship);
```

Description

Override Compare to implement any comparison operators the custom Variant (see page 3430) type supports. Descendants can implement comparison operators by overriding either the Compare or the CompareOp (see page 5021) method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Relationship returns a value indicating the relationship between Left and Right, as indicated in the following table:

Value	Operation
crLessThan	Left is less than Right.
crEqual	Left is equal to Right.
crGreaterThan	Left is greater than Right.

As implemented in TCustomVariantType (see page 5011), CompareOp (see page 5021) raises a system error indicating an invalid operation.

Tip: For custom Variants (see page 4997) that support a test of inequality, but not ordering, it makes more sense to override the CompareOp (see page 5021) method instead.

See Also

CompareOp (see page 5021)

CompareValue

Implementing Comparison Operations

54.1.16.4.9 TCustomVariantType.CompareOp Method

Implements any comparison operations of which the custom Variant (see page 3430) type is capable.

Pascal

```
function CompareOp(const Left: TVarData; const Right: TVarData; const Operator: TVarOp):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CompareOp(const TVarData Left, const TVarData Right, const TVarOp
Operator);
```

Description

Override CompareOp to implement any comparison operators the custom Variant (see page 3430) type supports. Descendants can implement comparison operators by overriding either the Compare (see page 5020) or the CompareOp method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opCmpEQ (see page 3918)	equality comparison
opCmpNE (see page 3920)	inequality comparison
opCmpLT (see page 3920)	less than
opCmpLE (see page 3919)	less than or equal
opCmpGT (see page 3919)	greater than
opCmpGE (see page 3918)	greater than or equal

CompareOp returns true if the comparison is true as specified. It returns false if the specified comparison does not hold true.

As implemented in TCustomVariantType (see page 5011), CompareOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 5029)

BinaryOp (see page 5018)

LeftPromotion (see page 5023)

RightPromotion (see page 5027)

Compare (see page 5020)

Implementing Comparison Operations

54.1.16.4.10 TCustomVariantType.Copy Method

Copies the data from a Variant (see page 3430) of this custom type.

Pascal

```
procedure Copy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean); virtual;
abstract;
```

C++

```
virtual __fastcall Copy(TVarData Dest, const TVarData Source, const Boolean Indirect) = 0;
```

Description

Override Copy to enable the custom Variant (see page 3430) type to be copied (assigned) to another Variant (see page 3430).

Dest is the TVarData (see page 3813) record from the Variant (see page 3430) whose value is to be a copy of the source Variant (see page 3430).

Source is the TVarData (see page 3813) record from a Variant (see page 3430) of this custom type that is to be copied to Dest.

Indirect indicates whether the copy is direct or indirect. When Indirect is true, the source TVarData (see page 3813) record contains an indirect reference to its data (the VarDataIsByRef (see page 5034) method should return true for Source). When making an indirect copy, it is not necessary to reallocate records or objects, because the Variant (see page 3430) does not "own" the data, it merely points to it. When making a direct copy, Copy must allocate new memory for any allocated records or objects, so that Dest can "own" its own copy of the data.

In TCustomVariantType (see page 5011), Copy is an abstract method. Descendant classes must override this method to provide an implementation.

See Also

VarDataCopy (see page 5031)

VarDataCopyNoInd (see page 5032)

SimplisticCopy (see page 5028)

VarDataIsByRef (see page 5034)

Copying and Clearing Custom Variants

54.1.16.4.11 TCustomVariantType.DispInvoke Method

Invokes a method call when the Variant (see page 3430) supports such operations.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); virtual;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a Variant (see page 3430) of the custom Variant (see page 3430) type, the system automatically generates a call to DispInvoke. TCustomVariantType (see page 5011) responds by generating a system error indicating an invalid dispatch attempt.

To create a custom Variant (see page 3430) type that allows applications to call methods or use properties, use TInvokeableVariantType (see page 5037) as a base class instead of TCustomVariantType (see page 5011). TInvokeableVariantType (see page 5037) introduces methods for executing method calls and implementing properties, and overrides the DispInvoke method so that it dispatches method calls to the appropriate methods.

Dest is a TVarData (see page 3813) record that receives the return value of any function calls.

Source is the TVarData (see page 3813) record that holds the data from the custom Variant (see page 3430) whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

TInvokeableVariantType (see page 5037)

54.1.16.4.12 TCustomVariantType.IsClear Method

Indicates whether the data from a custom Variant (see page 3430) represents a blank or unassigned value.

Pascal

```
function IsClear(const V: TVarData): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsClear(const TVarData V);
```

Description

Override IsClear to identify when data from a Variant (see page 3430) of the custom Variant (see page 3430) type represents a blank or unassigned value. The global VarIsClear (see page 5090) method calls IsClear when given a custom Variant (see page 3430) of this type.

As Implemented in TCustomVariantType (see page 5011), IsClear always returns false. Override this method to return true for invalid values or special values set by the Clear (see page 5019) method.

See Also

Clear (see page 5019)

VarIsClear (see page 5090)

Copying and Clearing Custom Variants

54.1.16.4.13 TCustomVariantType.LeftPromotion Method

Indicates whether the left argument to a binary operation should be coerced to a different type.

Pascal

```
function LeftPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean LeftPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override LeftPromotion to indicate when the left argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. LeftPromotion is called when the right-hand Variant (see page 3430) in a binary

or comparison operation is of this Variant (see page 3430) type and

The left-hand argument is a built-in Variant (see page 3430) type.

The left-hand argument is a custom Variant (see page 3430) type that does not indicate any problem with this Variant (see page 3430) type for the right-hand argument.

V is the TVarData (see page 3813) record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd (see page 3917)	addition
opSubtract (see page 3926)	subtraction
opMultiply (see page 3923)	multiplication
opDivide (see page 3921)	floating-point division
opIntDivide (see page 3922)	integer division
opModulus (see page 3922)	remainder
opShiftLeft (see page 3925)	left shift
opShiftRight (see page 3925)	right shift
opAnd (see page 3917)	bitwise and
opOr (see page 3924)	bitwise or
opXor (see page 3926)	bitwise exclusive or
opCompare (see page 3921)	any comparison operation

RequiredVarType returns the Variant (see page 3430) type code for the type to which the left-hand argument should be cast before performing the operation. If the operation can proceed with the left-hand argument left as-is, RequiredVarType returns the VType field of V.

LeftPromotion returns true if the TCustomVariantType (see page 5011) descendant can perform the operation (using the BinaryOp (see page 5018), CompareOp (see page 5021), or Compare (see page 5020) method) assuming the left-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 5011) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 5011), LeftPromotion returns the VarType (see page 5037) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 5018)

CompareOp (see page 5021)

Compare (see page 5020)

VarType (see page 5037)

RightPromotion (see page 5027)

Implementing Binary Operations

54.1.16.4.14 TCustomVariantType.OlePromotion Method

Indicates the type to which a Variant (see page 3430) value must be cast to make it Automation-compatible.

Pascal

```
function OlePromotion(const V: TVarData; out RequiredVarType: TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OlePromotion(const TVarData V, TVarType RequiredVarType);
```

Description

Override OlePromotion to indicate when a Variant (see page 3430) of this custom Variant (see page 3430) type must be cast to another type before it can be auto-marshaled using an Automation interface.

V is the TVarData (see page 3813) record for the custom Variant (see page 3430) that the application wants to marshal.

RequiredVarType returns the type to which V must be cast before it is Automation-compatible.

OlePromotion returns true if V can be cast to an Automation-compatible type, false if there is no type to which V can be cast.

As implemented in TCustomVariantType (see page 5011), OlePromotion sets RequiredVarType to varOleStr (see page 3936) and returns true if V does not already represent a Unicode string.

See Also

casttoole (see page 5019)

54.1.16.4.15 TCustomVariantType.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TCustomVariantType (see page 5011) object supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_addref (see page 5017)

_release (see page 5017)

54.1.16.4.16 TCustomVariantType.RaiseCastError Method

Generates a system error that indicates an invalid cast.

Pascal

```
procedure RaiseCastError;
```

C++

```
__fastcall RaiseCastError();
```

Description

Call RaiseCastError from the implementation of a TCustomVariantType (see page 5011) descendant when an operation fails because a Variant (see page 3430) value could not be cast to a necessary type.

See Also

Cast (see page 5018)

castto (see page 5019)

54.1.16.4.17 TCustomVariantType.RaiseDispError Method

Generates a system error indicating an invalid dispatch attempt.

Pascal

```
procedure RaiseDispError;
```

C++

```
__fastcall RaiseDispError();
```

Description

Call RaiseDispError from the implementation of a TCustomVariantType (see page 5011) descendant when an operation fails because the variant can't execute a method call. This raises EVariantDispatchError (see page 5005).

See Also

DispInvoke (see page 5022)

EVariantDispatchError (see page 5005)

54.1.16.4.18 TCustomVariantType.RaiseInvalidOp Method

Generates a system error indicating an invalid operation.

Pascal

```
procedure RaiseInvalidOp;
```

C++

```
__fastcall RaiseInvalidOp();
```

Description

Call RaiseInvalidOp from the implementation of a TCustomVariantType (see page 5011) descendant when it can't perform an operation.

See Also

BinaryOp (see page 5018)

UnaryOp (see page 5029)

CompareOp (see page 5021)

Compare (see page 5020)

54.1.16.4.19 TCustomVariantType.RightPromotion Method

Indicates whether the right argument to a binary operation should be coerced to a different type.

Pascal

```
function RightPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RightPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override RightPromotion to indicate when the right-hand argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. RightPromotion is called when the right-hand Variant (see page 3430) in a binary or comparison operation is of this Variant (see page 3430) type.

Note: In operations that involve two custom Variants (see page 4997), the left-hand argument's RightPromotion method is called first. The right-hand argument's LeftPromotion (see page 5023) method is then called only if the left-hand argument's RightPromotion method returns false.

V is the TVarData (see page 3813) record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd (see page 3917)	addition
opSubtract (see page 3926)	subtraction
opMultiply (see page 3923)	multiplication
opDivide (see page 3921)	floating-point division
opIntDivide (see page 3922)	integer division
opModulus (see page 3922)	remainder
opShiftRight (see page 3925)	right shift
opShiftRight (see page 3925)	right shift
opAnd (see page 3917)	bitwise and
opOr (see page 3924)	bitwise or
opXor (see page 3926)	bitwise exclusive or
opCompare (see page 3921)	any comparison operation

RequiredVarType returns the Variant (see page 3430) type code for the type to which the right-hand argument should be cast before performing the operation. If the operation can proceed with the right-hand argument left as-is, RequiredVarType returns the VType field of V.

RightPromotion returns true if the TCustomVariantType (see page 5011) descendant can perform the operation (using the BinaryOp (see page 5018), CompareOp (see page 5021), or Compare (see page 5020) method) assuming the right-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 5011) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 5011), RightPromotion returns the VarType (see page 5037) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 5018)

CompareOp (see page 5021)

Compare (see page 5020)

VarType (see page 5037)

LeftPromotion (see page 5023)

Implementing Binary Operations

54.1.16.4.20 TCustomVariantType.SimplisticClear Method

Clears a TVarData (see page 3813) record by re-initializing it.

Pascal

```
procedure SimplisticClear(var V: TVarData);
```

C++

```
__fastcall SimplisticClear(TVarData V);
```

Description

Call SimplisticClear from the implementation of a TCustomVariantType (see page 5011) descendant to clear the TVarData (see page 3813) record of a Variant (see page 3430) of the custom type. This method should only be called if the TVarData (see page 3813) record does not contain any members that must be freed.

SimplisticClear simply calls the VarDataInit (see page 5033) method, which reinitializes the TVarData (see page 3813) record passed as V.

See Also

Clear (see page 5019)

VarDataClear (see page 5031)

VarDataInit (see page 5033)

54.1.16.4.21 TCustomVariantType.SimplisticCopy Method

Performs a simple block copy of one TVarData (see page 3813) record to another.

Pascal

```
procedure SimplisticCopy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean =
False);
```

C++

```
__fastcall SimplisticCopy(TVarData Dest, const TVarData Source, const Boolean Indirect =
False);
```

Description

Call SimplisticCopy from the implementation of a TCustomVariantType (see page 5011) descendant to block copy one TVarData (see page 3813) record to another.

Dest is the TVarData (see page 3813) record that receives the copy.

Source is the TVarData (see page 3813) record to copy.

Indirect indicates whether Source represents the data (false) or merely a reference to the data (true).

See Also

Copy (see page 5022)

VarDataCopy (see page 5031)

VarDataCopyNoInd (see page 5032)

Copying and Clearing Custom Variants

54.1.16.4.22 TCustomVariantType.UnaryOp Method

Implements any unary operations of which the custom Variant (see page 3430) type is capable.

Pascal

```
procedure UnaryOp(var Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall UnaryOp(TVarData Right, const TVarOp Operator);
```

Description

Override UnaryOp to implement any unary operators the custom Variant (see page 3430) type supports.

Right is the value that appears to the right of the operator. UnaryOp changes this value to indicate the result of the operation.

Operator identifies the operator that appears before Right. It can be any of the following values:

Value	Operation
opNegate (see page 3923)	unary minus
opNot (see page 3924)	logical negation

As implemented in TCustomVariantType (see page 5011), UnaryOp raises a system error indicating an invalid operation.

See Also

BinaryOp (see page 5018)

CompareOp (see page 5021)

Implementing Unary Operations

54.1.16.4.23 TCustomVariantType.VarDataCast Method

Performs a cast from one Variant (see page 3430) type to another.

Pascal

```
procedure VarDataCast(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCast(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCast from the implementation of a TCustomVariantType (see page 5011) descendant to perform casts on the data from another Variant (see page 3430) type to the type specified by the VarType (see page 5037) property. Do not use VarDataCast as the entire implementation of the Cast (see page 5018) method, as this will lead to a circular reference.

Dest returns the result of the cast.

Source is the TVarData (see page 3813) record of another Variant (see page 3430) type that is to be converted.

See Also

Cast (see page 5018)

castto (see page 5019)

VarDataCastTo (see page 5030)

VarType (see page 5037)

54.1.16.4.24 VarDataCastTo Method

This is the overview for the VarDataCastTo method overload.

Overload List

	Name	Description
	TCustomVariantType.VarDataCastTo (TVarData, TVarData, TVarType) (see page 5030)	Performs a cast from one Variant (see page 3430) type to another.
	TCustomVariantType.VarDataCastTo (TVarData, TVarType) (see page 5030)	Performs a cast from one Variant (see page 3430) type to another.

54.1.16.4.24.1 TCustomVariantType.VarDataCastTo Method (TVarData, TVarData, TVarType)

Performs a cast from one Variant (see page 3430) type to another.

Pascal

```
procedure VarDataCastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
overload;
procedure VarDataCastTo(var Dest: TVarData; const AVarType: TVarType); overload;
```

C++

```
__fastcall VarDataCastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
__fastcall VarDataCastTo(TVarData Dest, const TVarType AVarType);
```

Description

Call VarDataCastTo from the implementation of a TCustomVariantType (see page 5011) descendant to perform casts on the data from one Variant (see page 3430) type to another. Do not use VarDataCastTo as the entire implementation of the CastTo (see page 5019) method, as this will lead to a circular reference.

Dest returns the result of the cast. If VarDataCastTo is called with no Source parameter, it also provides the source data to be cast.

Source is the TVarData (see page 3813) record to be cast.

VarType (see page 5037) is the type code of the Variant (see page 3430) type to which the data should be cast.

See Also

castto (see page 5019)

54.1.16.4.25 TCustomVariantType.VarDataCastToOleStr Method

Performs a cast to a Variant (see page 3430) type of varOleStr (see page 3936).

Pascal

```
procedure VarDataCastToOleStr(var Dest: TVarData);
```

C++

```
__fastcall VarDataCastToOleStr(TVarData Dest);
```

Description

Call VarDataCastToOleStr from the implementation of a TCustomVariantType (see page 5011) descendant to perform casts to a TVarData (see page 3813) record with Variant (see page 3430) type code of varOleStr (see page 3936) (Unicode string).

Dest is the TVarData (see page 3813) record of a Variant (see page 3430), and returns the result of the cast.

See Also

VarDataFromOleStr (see page 5032)

VarDataToStr (see page 5036)

54.1.16.4.26 TCustomVariantType.VarDataClear Method

Clears the data from a built-in Variant (see page 3430) type.

Pascal

```
procedure VarDataClear(var Dest: TVarData);
```

C++

```
__fastcall VarDataClear(TVarData Dest);
```

Description

Call VarDataClear from the implementation of a TCustomVariantType (see page 5011) descendant to clear the data in the TVarData (see page 3813) record of a Variant (see page 3430). Do not use VarDataClear as the entire implementation of the Clear (see page 5019) method, as this will lead to a circular reference.

Dest is the TVarData (see page 3813) record of a Variant (see page 3430) type, and returns the result after clearing the data.

See Also

SimplisticClear (see page 5028)

Clear (see page 5019)

54.1.16.4.27 TCustomVariantType.VarDataCopy Method

Copies the data from one built-in Variant (see page 3430) type to another.

Pascal

```
procedure VarDataCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopy(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopy from the implementation of a TCustomVariantType (see page 5011) descendant to clear the data from a destination Variant (see page 3430) and then copy the TVarData (see page 3813) record from another Variant (see page 3430). VarDataCopy can handle data that is stored indirectly (when the VType field includes varByRef (see page 3930)). Do not use VarDataCopy as the entire implementation of the Copy (see page 5022) method, as this will lead to a circular

reference.

Dest is the TVarData (see page 3813) record that receives the copy.

Source is the TVarData (see page 3813) record to be copied.

See Also

SimplisticCopy (see page 5028)

VarDataCopyNoInd (see page 5032)

Copy (see page 5022)

54.1.16.4.28 TCustomVariantType.VarDataCopyNoInd Method

Copies the data from one built-in Variant (see page 3430) type to another.

Pascal

```
procedure VarDataCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopyNoInd(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopyNoInd from the implementation of a TCustomVariantType (see page 5011) descendant to clear the data from a destination Variant (see page 3430) and then copy the TVarData (see page 3813) record from another Variant (see page 3430). Unlike the VarDataCopy (see page 5031) method, it can't handle data that is stored indirectly (when the VType field includes varByRef (see page 3930)). However, it is often more efficient than VarDataCopy (see page 5031).

Dest is the TVarData (see page 3813) record that receives the copy.

Source is the TVarData (see page 3813) record to be copied.

See Also

VarDataCopy (see page 5031)

Copy (see page 5022)

54.1.16.4.29 TCustomVariantType.VarDataFromOleStr Method

Creates a TVarData (see page 3813) record to represent a Unicode string.

Pascal

```
procedure VarDataFromOleStr(var V: TVarData; const Value: WideString);
```

C++

```
__fastcall VarDataFromOleStr(TVarData V, const BSTR Value);
```

Description

Call VarDataCopy (see page 5031) from the implementation of a TCustomVariantType (see page 5011) descendant to fill out a TVarData (see page 3813) record so that it represents a specified Unicode string.

V is the TVarData (see page 3813) record to be filled out.

Value is a Unicode string that becomes the value of V.

See Also

VarDataCastToOleStr (see page 5030)

VarDataFromStr (see page 5033)

54.1.16.4.30 TCustomVariantType.VarDataFromStr Method

Creates a TVarData (see page 3813) record to represent a string.

Pascal

```
procedure VarDataFromStr(var V: TVarData; const Value: string);
```

C++

```
__fastcall VarDataFromStr(TVarData V, const UnicodeString Value);
```

Description

Call VarDataCopy (see page 5031) from the implementation of a TCustomVariantType (see page 5011) descendant to fill out a TVarData (see page 3813) record so that it represents a specified string.

V is the TVarData (see page 3813) record to be filled out.

Value is a string that becomes the value of V.

See Also

VarDataFromOleStr (see page 5032)

VarDataIsStr (see page 5036)

VarDataToStr (see page 5036)

54.1.16.4.31 TCustomVariantType.VarDataInit Method

Initializes a TVarData (see page 3813) record.

Pascal

```
procedure VarDataInit(var Dest: TVarData);
```

C++

```
__fastcall VarDataInit(TVarData Dest);
```

Description

Call VarDataInit from the implementation of a TCustomVariantType (see page 5011) descendant to initialize a newly-allocated TVarData (see page 3813) record.

See Also

VarDataClear (see page 5031)

54.1.16.4.32 TCustomVariantType.VarDataIsArray Method

Indicates whether a TVarData (see page 3813) record represents the data from a Variant (see page 3430) array.

Pascal

```
function VarDataIsArray(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsArray(const TVarData V);
```

Description

VarDataIsArray returns true if the VType field of V contains the varArray (see page 3929) bit.

See Also

VarDataIsByRef (see page 5034)

VarType (see page 5101)

54.1.16.4.33 TCustomVariantType.VarDataIsByRef Method

Indicates whether a TVarData (see page 3813) record stores its data indirectly.

Pascal

```
function VarDataIsByRef(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsByRef(const TVarData V);
```

Description

VarDataIsArray (see page 5033) returns true if the VType field of V contains the varByRef (see page 3930) bit.

See Also

VarDataIsArray (see page 5033)

VarType (see page 5101)

54.1.16.4.34 TCustomVariantType.VarDataIsEmptyParam Method

Indicates whether a TVarData (see page 3813) record represents an unassigned optional parameter.

Pascal

```
function VarDataIsEmptyParam(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsEmptyParam(const TVarData V);
```

Description

VarDataIsEmptyParam returns true if the given TVarData (see page 3813) record represents an unassigned optional parameter.

See Also

VarDataIsArray (see page 5033)

VarType (see page 5101)

54.1.16.4.35 TCustomVariantType.VarDataIsFloat Method

Indicates whether a TVarData (see page 3813) record represents a floating-point value.

Pascal

```
function VarDataIsFloat(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsFloat(const TVarData V);
```

Description

VarDataIsFloat returns true if the given TVarData (see page 3813) record comes from a built-in Variant (see page 3430) type that represents a floating-point value (Single, Double, or Currency (see page 3242)). VarDataIsFloat does not return true for data from Custom Variant (see page 3430) types.

See Also

VarDataIsNumeric (see page 5035)

VarType (see page 5101)

VarDataIsOrdinal (see page 5035)

VarDataIsStr (see page 5036)

54.1.16.4.36 TCustomVariantType.VarDataIsNumeric Method

Indicates whether a TVarData (see page 3813) record represents a numeric value.

Pascal

```
function VarDataIsNumeric(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsNumeric(const TVarData V);
```

Description

VarDataIsNumeric returns true if the given TVarData (see page 3813) record represents a numeric value of some sort. This can be either a floating-point value or an ordinal value. VarDataIsNumeric does not return true for data from Custom Variant (see page 3430) types.

See Also

VarDataIsFloat (see page 5034)

VarType (see page 5101)

VarDataIsOrdinal (see page 5035)

VarDataIsStr (see page 5036)

54.1.16.4.37 TCustomVariantType.VarDataIsOrdinal Method

Indicates whether a TVarData (see page 3813) record represents a ordinal value.

Pascal

```
function VarDataIsOrdinal(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsOrdinal(const TVarData V);
```

Description

VarDataIsOrdinal returns true if the given TVarData (see page 3813) record comes from a built-in Variant (see page 3430) type that represents a ordinal value (such as an integer or Boolean type). VarDataIsOrdinal does not return true for data from Custom Variant (see page 3430) types.

See Also

VarDataIsNumeric (see page 5035)

VarType (see page 5101)

VarDataIsFloat (see page 5034)

VarDataIsStr (see page 5036)

54.1.16.4.38 TCustomVariantType.VarDataIsStr Method

Indicates whether a TVarData (see page 3813) record represents a string (normal or Unicode).

Pascal

```
function VarDataIsStr(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsStr(const TVarData V);
```

Description

VarDataIsStr returns true if the given TVarData (see page 3813) record has a VType field of varString (see page 3938) or varOleStr (see page 3936).

See Also

VarDataIsNumeric (see page 5035)

VarType (see page 5101)

VarDataIsFloat (see page 5034)

VarDataIsStr

54.1.16.4.39 TCustomVariantType.VarDataToStr Method

Returns a string that represents the value from a TVarData (see page 3813) record.

Pascal

```
function VarDataToStr(const V: TVarData): string;
```

C++

```
__fastcall UnicodeString VarDataToStr(const TVarData V);
```

Description

Call VarDataToStr from the implementation of a TCustomVariantType (see page 5011) descendant to extract a string that represents the value of a Variant (see page 3430).

V is the TVarData (see page 3813) record that represents the value of the Variant (see page 3430).

See Also

VarDataCastToOleStr (see page 5030)

VarDataFromStr (see page 5033)

54.1.16.5 TCustomVariantType Properties

The properties of the TCustomVariantType class are listed here.

Public Properties

	Name	Description
	VarType (see page 5037)	Indicates the Variant (see page 3430) type code for the custom Variant (see page 3430) type.

54.1.16.5.1 TCustomVariantType.VarType Property

Indicates the Variant (see page 3430) type code for the custom Variant (see page 3430) type.

Pascal

```
property VarType: TVarType;
```

C++

```
__property TVarType VarType;
```

Description

The VarType property provides read-only access to the type code that identifies a custom Variant (see page 3430) data type.

The value of VarType is set in the constructor. This can be an automatically generated value, or it can be a value explicitly passed to the constructor. Note that if the VarType is automatically-generated, it may not remain the same for every application that uses the custom Variant (see page 3430) type: its value will depend on the other custom Variants (see page 4997) also used in the application.

See Also

TVarData (see page 3813)

Create (see page 5015)

VarType (see page 5101)

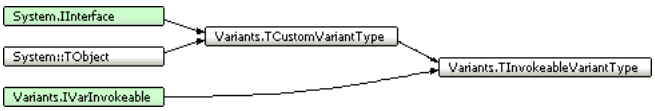
FindCustomVariantType (see page 5073)

Enabling Casting

54.1.17 TInvokeableVariantType Class

TInvokeableVariantType is the base class for custom Variant (see page 3430) types that implement custom Variants (see page 4997) with properties and methods.

Class Hierarchy



Pascal

```
TInvokeableVariantType = class(TCustomVariantType, IVarInvokeable);
```

C++

```
class TInvokeableVariantType : public TCustomVariantType, public IVarInvokeable;
```

File

Variants

Description

Use TInvokeableVariantType to define new data types for Delphi Variants (see page 4997) with properties and methods. TInvokeableVariantType descendants do not represent the Variants (see page 4997) themselves. Rather, they act as utility classes that the custom Variants (see page 4997) call on to perform their operations.

Note: Do not define TInvokeableVariantType descendants in C++. The logic for custom Variant (see page 3430) types must be defined in Delphi code.

To create a Variant (see page 3430) type that supports properties and methods,

1. Map the storage of the Variant (see page 3430)'s data on to the TVarData (see page 3813) record.
2. Write (see page 3789) a method that creates instances of your custom Variant (see page 3430). This fills in the Variant (see page 3430)'s data as defined in step 1.
3. Declare a class that descends from TInvokeableVariantType.
4. Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction (see page 5041), DoProcedure (see page 5042), GetProperty (see page 5043), SetProperty (see page 5044), BinaryOp (see page 5018), UnaryOp (see page 5029), CompareOp (see page 5021) or Compare (see page 5020), Cast (see page 5018), CastTo (see page 5019), Clear (see page 5019), Copy (see page 5022), IsClear (see page 5023), LeftPromotion (see page 5023), and RightPromotion (see page 5027)).
5. In the initialization section of the unit that defines your TInvokeableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant (see page 3430)-handling system so that the new Variant (see page 3430) type is enabled.
6. In the finalization section of the unit that defines your TInvokeableVariantType descendant, free the instance of your class. This automatically unregisters the Variant (see page 3430) type.

For an example, see the VarConv (see page 4973) unit, which implements a custom Variant (see page 3430) type for measurements.

Tip: If the custom Variant (see page 3430) type uses an object instance to store its data, you can automatically implement the published properties of that object by deriving your custom Variant (see page 3430) type from TPublishableVariantType instead.

See Also

IDH

TPublishableVariantType (see page 4728)

TVarData (see page 3813)

Supporting Properties and Methods in Custom Variants

54.1.17.1 TInvokeableVariantType Members

The following tables list the members exposed by TInvokeableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 5017)	Increments the reference count for this interface.
	_Release (see page 5017)	Decrements the reference count for this interface.
	DispInvoke (see page 5041)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5037) method.
	FixupIdent (see page 5043)	Fixes up dispatch identifier.

	LeftPromotion (see page 5023)	Indicates whether the left argument to a binary operation should be coerced to a different type.
	OlePromotion (see page 5024)	Indicates the type to which a Variant (see page 3430) value must be cast to make it Automation-compatible.
	QueryInterface (see page 5025)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5025)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5026)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5026)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 5027)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5028)	Clears a TVarData (see page 3813) record by re-initializing it.
	SimplisticCopy (see page 5028)	Performs a simple block copy of one TVarData (see page 3813) record to another.
	VarDataCast (see page 5029)	Performs a cast from one Variant (see page 3430) type to another.
	VarDataCastTo (see page 5030)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5030)	Performs a cast to a Variant (see page 3430) type of varOleStr (see page 3936).
	VarDataClear (see page 5031)	Clears the data from a built-in Variant (see page 3430) type.
	VarDataCopy (see page 5031)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataCopyNoInd (see page 5032)	Copies the data from one built-in Variant (see page 3430) type to another.
	VarDataFromOleStr (see page 5032)	Creates a TVarData (see page 3813) record to represent a Unicode string.
	VarDataFromStr (see page 5033)	Creates a TVarData (see page 3813) record to represent a string.
	VarDataInit (see page 5033)	Initializes a TVarData (see page 3813) record.
	VarDataIsArray (see page 5033)	Indicates whether a TVarData (see page 3813) record represents the data from a Variant (see page 3430) array.
	VarDataIsByRef (see page 5034)	Indicates whether a TVarData (see page 3813) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5034)	Indicates whether a TVarData (see page 3813) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5034)	Indicates whether a TVarData (see page 3813) record represents a floating-point value.
	VarDataIsNumeric (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a numeric value.
	VarDataIsOrdinal (see page 5035)	Indicates whether a TVarData (see page 3813) record represents a ordinal value.
	VarDataIsStr (see page 5036)	Indicates whether a TVarData (see page 3813) record represents a string (normal or Unicode).
	VarDataToStr (see page 5036)	Returns a string that represents the value from a TVarData (see page 3813) record.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	BinaryOp (see page 5018)	Implements any binary operations of which the custom Variant (see page 3430) type is capable.
	Cast (see page 5018)	Converts the data from another Variant (see page 3430) type so that it represents this custom type's data format.
	CastTo (see page 5019)	This is CastTo, a member of class TCustomVariantType.
	CastToOle (see page 5019)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 5019)	Sets the data for this custom Variant (see page 3430) type to a blank or unassigned value.
	Compare (see page 5020)	Implements comparisons that involve the custom Variant (see page 3430) type.
	CompareOp (see page 5021)	Implements any comparison operations of which the custom Variant (see page 3430) type is capable.
	Copy (see page 5022)	Copies the data from a Variant (see page 3430) of this custom type.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 5014)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 5015)	Frees an instance of TCustomVariantType (see page 5011).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoFunction (see page 5041)	Implements all methods that return values.
	DoProcedure (see page 5042)	Implements all methods that do not return values.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetProperty (see page 5043)	Returns the value for any property this Variant (see page 3430) type supports.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 5023)	Indicates whether the data from a custom Variant (see page 3430) represents a blank or unassigned value.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetProperty (see page 5044)	Sets the value for any property this Variant (see page 3430) type supports.
	ToString (see page 3391)	Returns a string containing the class name.
	UnaryOp (see page 5029)	Implements any unary operations of which the custom Variant (see page 3430) type is capable.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	VarType (see page 5037)	Indicates the Variant (see page 3430) type code for the custom Variant (see page 3430) type.

54.1.17.2 TInvokeableVariantType Methods

The methods of the TInvokeableVariantType class are listed here.

Protected Methods

	Name	Description
	DispInvoke (see page 5041)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5037) method.

	FixupIdent (see page 5043)	Fixes up dispatch identifier.

Public Methods

	Name	Description
	DoFunction (see page 5041)	Implements all methods that return values.
	DoProcedure (see page 5042)	Implements all methods that do not return values.
	GetProperty (see page 5043)	Returns the value for any property this Variant (see page 3430) type supports.
	SetProperty (see page 5044)	Sets the value for any property this Variant (see page 3430) type supports.

54.1.17.2.1 TInvokeableVariantType.DispInvoke Method

Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5037) method.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); override;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a Variant (see page 3430) of the custom Variant (see page 3430) type, the system automatically generates a call to DispInvoke. TInvokeableVariantType (see page 5037) responds by generating a call to the GetProperty (see page 5043), SetProperty (see page 5044), DoProcedure (see page 5042), or DoFunction (see page 5041) method, as appropriate.

Dest is a TVarData (see page 3813) record that receives the return value of any function calls.

Source is the TVarData (see page 3813) record that holds the data from the custom Variant (see page 3430) whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

GetProperty (see page 5043)

SetProperty (see page 5044)

DoProcedure (see page 5042)

DoFunction (see page 5041)

54.1.17.2.2 TInvokeableVariantType.DoFunction Method

Implements all methods that return values.

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
Arguments: TVarDataArray): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const UnicodeString
Name, const TVarDataArray Arguments);
```

Description

Override DoFunction to provide the implementation for method calls that return values. When the application invokes a Variant (see page 3430) member function (Delphi) or calls the Variant (see page 3430)'s Exec method (C++) a call is automatically generated to DoFunction.

Dest is a TVarData (see page 3813) record that receives the value that the function returns.

V is the TVarData (see page 3813) record from the Variant (see page 3430) instance that was used to call the function.

Name is the name of the function.

Arguments is a dynamic array of TVarData (see page 3813) records, where each one represents the value of an argument passed to the function.

DoFunction returns true if it executes the method call, false if it does not implement the specified function.

As implemented in TInvokeableVariant, DoFunction always returns false, because it does not implement any function calls.

See Also

GetProperty (see page 5043)

DoProcedure (see page 5042)

SetProperty (see page 5044)

Exec

Supporting Properties and Methods in Custom Variants

54.1.17.2.3 TInvokeableVariantType.DoProcedure Method

Implements all methods that do not return values.

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoProcedure(const TVarData V, const UnicodeString Name, const
TVarDataArray Arguments);
```

Description

Override DoProcedure to provide the implementation for methods calls that do not return values. When the application invokes a Variant (see page 3430) member procedure (Delphi) or calls the Variant (see page 3430)'s Exec method (C++) a call is automatically generated to DoProcedure that allows the TInvokeableVariant descendant to implement the call.

V is the TVarData (see page 3813) record from the Variant (see page 3430) instance that was used to call the procedure.

Name is the name of the procedure.

Arguments is a dynamic array of TVarData (see page 3813) records, where each one represents the value of an argument passed to the procedure.

DoProcedure returns true if it executes the procedure call, false if it does not implement the specified procedure.

As implemented in TInvokeableVariant, DoProcedure always returns false, because it does not implement any procedure calls.

See Also

GetProperty (see page 5043)

DoFunction (see page 5041)

SetProperty (see page 5044)

Exec

Supporting Properties and Methods in Custom Variants

54.1.17.2.4 TInvokeableVariantType.FixupIdent Method

Fixes up dispatch identifier.

Pascal

```
function FixupIdent(const AText: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString FixupIdent(const UnicodeString AText);
```

Description

FixupIdent is called internally to determine the final fixup for the dispatch identifier. As defined in TInvokeableVariantType (see page 5037), this fixup consists of converting the identifier to all upper case.

54.1.17.2.5 TInvokeableVariantType.GetProperty Method

Returns the value for any property this Variant (see page 3430) type supports.

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString
Name);
```

Description

Override GetProperty to provide the read implementations for properties of instances of the Variant (see page 3430). When the application reads a Variant (see page 3430) property (Delphi) or calls the Variant (see page 3430)'s Exec method (C++) a call is automatically generated to GetProperty that allows the TInvokeableVariant descendant to implement the call.

Dest is a TVarData (see page 3813) record that receives the property value.

V is the TVarData (see page 3813) record from the Variant (see page 3430) instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, false if it does not implement the specified property.

As implemented in TInvokeableVariant, GetProperty always returns false, because it does not implement any properties.

See Also

DoFunction (see page 5041)

DoProcedure (see page 5042)

SetProperty (see page 5044)

Exec

Supporting Properties and Methods in Custom Variants

54.1.17.2.6 TInvokeableVariantType.SetProperty Method

Sets the value for any property this Variant (see page 3430) type supports.

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const
TVarData Value);
```

Description

Override SetProperty to provide the write implementation for the properties of instances of the invokeable Variant (see page 3430) type. When the application writes to a Variant (see page 3430) property (Delphi) or calls the Variant (see page 3430)'s Exec method (C++) a call is automatically generated to SetProperty that allows the TInvokeableVariant descendant to implement the call.

Name is the name of the property.

Value is a TVarData (see page 3813) record that contains the new property value.

GetProperty (see page 5043) returns true if it sets a property value, false if it does not implement the specified property or if the specified property is read-only.

As implemented in TInvokeableVariant, GetProperty (see page 5043) always returns false, because it does not implement any properties.

See Also

DoFunction (see page 5041)

DoProcedure (see page 5042)

GetProperty (see page 5043)

Exec

Supporting Properties and Methods in Custom Variants

54.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IVarInstanceReference (see page 5045)	This is class Variants.IVarInstanceReference.
	IVarInvokeable (see page 5046)	This is class Variants.IVarInvokeable.

54.2.1 IVarInstanceReference Interface

Class Hierarchy

Pascal

```
IVarInstanceReference = interface;
```

C++

```
__interface IVarInstanceReference;
```

File

Variants

Description

This is class Variants.IVarInstanceReference.

54.2.1.1 IVarInstanceReference Members

The following tables list the members exposed by IVarInstanceReference.

Public Methods

	Name	Description
	GetInstance (see page 5046)	This is GetInstance, a member of class IVarInstanceReference.

54.2.1.2 IVarInstanceReference Methods

The methods of the IVarInstanceReference class are listed here.

Public Methods

	Name	Description
	GetInstance (see page 5046)	This is GetInstance, a member of class IVarInstanceReference.

54.2.1.2.1 IVarInstanceReference.GetInstance Method

Pascal

```
function GetInstance(const V: TVarData): TObject;
```

C++

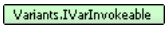
```
__fastcall TObject * GetInstance(const TVarData V);
```

Description

This is GetInstance, a member of class IVarInstanceReference.

54.2.2 IVarInvokeable Interface

Class Hierarchy



Pascal

```
IVarInvokeable = interface;
```

C++

```
__interface IVarInvokeable;
```

File

Variants

Description

This is class Variants.IVarInvokeable.

54.2.2.1 IVarInvokeable Members

The following tables list the members exposed by IVarInvokeable.

Public Methods

	Name	Description
	DoFunction (see page 5047)	This is DoFunction, a member of class IVarInvokeable.
	DoProcedure (see page 5047)	This is DoProcedure, a member of class IVarInvokeable.
	GetProperty (see page 5047)	This is GetProperty, a member of class IVarInvokeable.
	SetProperty (see page 5047)	This is SetProperty, a member of class IVarInvokeable.

54.2.2.2 IVarInvokeable Methods

The methods of the IVarInvokeable class are listed here.

Public Methods

	Name	Description
	DoFunction (see page 5047)	This is DoFunction, a member of class IVarInvokeable.
	DoProcedure (see page 5047)	This is DoProcedure, a member of class IVarInvokeable.
	GetProperty (see page 5047)	This is GetProperty, a member of class IVarInvokeable.

	SetProperty (see page 5047)	This is SetProperty, a member of class IVarInvokeable.

54.2.2.2.1 IVarInvokeable.DoFunction Method

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
Arguments: TVarDataArray): Boolean;
```

C++

```
__fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const UnicodeString Name, const
TVarDataArray Arguments);
```

Description

This is DoFunction, a member of class IVarInvokeable.

54.2.2.2.2 IVarInvokeable.DoProcedure Method

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean;
```

C++

```
__fastcall Boolean DoProcedure(const TVarData V, const UnicodeString Name, const TVarDataArray
Arguments);
```

Description

This is DoProcedure, a member of class IVarInvokeable.

54.2.2.2.3 IVarInvokeable.GetProperty Method

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
```

C++

```
__fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString Name);
```

Description

This is GetProperty, a member of class IVarInvokeable.

54.2.2.2.4 IVarInvokeable.SetProperty Method

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
```

C++

```
__fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const TVarData
Value);
```

Description

This is SetProperty, a member of class IVarInvokeable.

54.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	_DispInvoke (see page 5050)	This is function Variants._DispInvoke.
	_OleVarFromInt (see page 5051)	This is function Variants._OleVarFromInt.
	_OleVarFromInt64 (see page 5051)	This is function Variants._OleVarFromInt64.
	_OleVarFromLStr (see page 5051)	This is function Variants._OleVarFromLStr.
	_OleVarFromPStr (see page 5052)	This is function Variants._OleVarFromPStr.
	_OleVarFromUInt64 (see page 5052)	This is function Variants._OleVarFromUInt64.
	_OleVarFromUStr (see page 5052)	This is function Variants._OleVarFromUStr.
	_OleVarFromVar (see page 5053)	This is function Variants._OleVarFromVar.
	_VarAdd (see page 5053)	This is function Variants._VarAdd.
	_VarAddRef (see page 5053)	This is function Variants._VarAddRef.
	_VarAnd (see page 5054)	This is function Variants._VarAnd.
	_VarArrayGet (see page 5054)	This is function Variants._VarArrayGet.
	_VarArrayPut (see page 5054)	This is function Variants._VarArrayPut.
	_VarArrayRedim (see page 5055)	This is function Variants._VarArrayRedim.
	_VarCast (see page 5055)	This is function Variants._VarCast.
	_VarCastOle (see page 5055)	This is function Variants._VarCastOle.
	_VarClear (see page 5056)	This is function Variants._VarClear.
	_VarClr (see page 5056)	This is function Variants._VarClr.
	_VarCmpEQ (see page 5056)	This is function Variants._VarCmpEQ.
	_VarCmpGE (see page 5057)	This is function Variants._VarCmpGE.
	_VarCmpGT (see page 5057)	This is function Variants._VarCmpGT.
	_VarCmpLE (see page 5057)	This is function Variants._VarCmpLE.
	_VarCmpLT (see page 5058)	This is function Variants._VarCmpLT.
	_VarCmpNE (see page 5058)	This is function Variants._VarCmpNE.
	_VarCopy (see page 5058)	This is function Variants._VarCopy.
	_VarCopyNoInd (see page 5059)	This is function Variants._VarCopyNoInd.
	_VarDiv (see page 5059)	This is function Variants._VarDiv.
	_VarFromBool (see page 5059)	This is function Variants._VarFromBool.
	_VarFromCurr (see page 5060)	This is function Variants._VarFromCurr.
	_VarFromDisp (see page 5060)	This is function Variants._VarFromDisp.
	_VarFromDynArray (see page 5060)	This is function Variants._VarFromDynArray.
	_VarFromInt (see page 5061)	This is function Variants._VarFromInt.
	_VarFromInt64 (see page 5061)	This is function Variants._VarFromInt64.
	_VarFromIntf (see page 5061)	This is function Variants._VarFromIntf.
	_VarFromLStr (see page 5062)	This is function Variants._VarFromLStr.
	_VarFromPStr (see page 5062)	This is function Variants._VarFromPStr.
	_VarFromReal (see page 5062)	This is function Variants._VarFromReal.
	_VarFromTDateTime (see page 5063)	This is function Variants._VarFromTDateTime.
	_VarFromUInt64 (see page 5063)	This is function Variants._VarFromUInt64.
	_VarFromUStr (see page 5063)	This is function Variants._VarFromUStr.
	_VarFromWChar (see page 5064)	This is function Variants._VarFromWChar.
	_VarFromWStr (see page 5064)	This is function Variants._VarFromWStr.
	_VarMod (see page 5064)	This is function Variants._VarMod.
	_VarMul (see page 5065)	This is function Variants._VarMul.

	_VarNeg (see page 5065)	This is function Variants._VarNeg.
	_VarNot (see page 5065)	This is function Variants._VarNot.
	_VarOr (see page 5066)	This is function Variants._VarOr.
	_VarRDiv (see page 5066)	This is function Variants._VarRDiv.
	_VarShl (see page 5066)	This is function Variants._VarShl.
	_VarShr (see page 5067)	This is function Variants._VarShr.
	_VarSub (see page 5067)	This is function Variants._VarSub.
	_VarToBool (see page 5067)	This is function Variants._VarToBool.
	_VarToCurrency (see page 5068)	This is function Variants._VarToCurrency.
	_VarToDisp (see page 5068)	This is function Variants._VarToDisp.
	_VarToDynArray (see page 5068)	This is function Variants._VarToDynArray.
	_VarToInt64 (see page 5069)	This is function Variants._VarToInt64.
	_VarToInteger (see page 5069)	This is function Variants._VarToInteger.
	_VarToIntf (see page 5069)	This is function Variants._VarToIntf.
	_VarToLStr (see page 5070)	This is function Variants._VarToLStr.
	_VarToPStr (see page 5070)	This is function Variants._VarToPStr.
	_VarToReal (see page 5070)	This is function Variants._VarToReal.
	_VarToUInt64 (see page 5071)	This is function Variants._VarToUInt64.
	_VarToUStr (see page 5071)	This is function Variants._VarToUStr.
	_VarToWStr (see page 5071)	This is function Variants._VarToWStr.
	_VarXor (see page 5072)	This is function Variants._VarXor.
	DynArrayFromVariant (see page 5072)	Creates a dynamic array from a Variant (see page 3430).
	DynArrayToVariant (see page 5072)	Creates a Variant (see page 3430) array from a dynamic array.
	FindCustomVariantType (see page 5073)	Retrieves the object that implements a custom Variant (see page 3430) type.
	FindCustomVariantType (see page 5073)	Retrieves the object that implements a custom Variant (see page 3430) type.
	FindVarData (see page 5074)	Returns pointer to variant's data.
	HandleConversionException (see page 5074)	Handles exceptions during variant type conversions.
	Null (see page 5075)	Returns a Null variant.
	Unassigned (see page 5076)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 5077)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 5077)	Creates a variant array.
	VarArrayCreateError (see page 5078)	Indicates an array creation error.
	VarArrayDimCount (see page 5078)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 5079)	Returns a Variant (see page 3430) that represents a single value from a multi-dimensional Variant (see page 3430) array.
	VarArrayHighBound (see page 5079)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 5080)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 5080)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 5081)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 5081)	Sets the value of a single cell in a multi-dimensional Variant (see page 3430) array.
	VarArrayRef (see page 5082)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 5083)	Unlocks a variant array.
	VarAsError (see page 5083)	Converts a HRESULT (see page 3826) into an Error (see page 3662) variant.
	VarAsType (see page 5083)	Converts a variant to specified type.
	VarCastError (see page 5084)	Raises an EVariantTypeCastError (see page 5010) exception.
	VarCastError (see page 5084)	Raises an EVariantTypeCastError (see page 5010) exception.
	VarCheckEmpty (see page 5084)	Raises an exception if a specified variant's value is Unassigned (see page 5076).
	VarCompareValue (see page 5085)	Returns the relationship between Variants (see page 4997).
	VarCopyNoInd (see page 5086)	Copies a Variant (see page 3430).
	VarEnsureRange (see page 5086)	Returns a Variant (see page 3430) with a value in a specified range.
	VarFromDateTime (see page 5087)	Returns a variant containing a specified data and time.
	VarInRange (see page 5087)	Indicates whether a Variant (see page 3430)'s value is in a specified range.
	VarInvalidNullOp (see page 5088)	Indicate variant operation on null.

	VarInvalidOp (see page 5088)	Indicate invalid variant operation.
	VarIsArray (see page 5089)	Indicates whether the specified variant is an array.
	VarIsArray (see page 5089)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 5089)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 5090)	Indicates whether the specified variant has an undefined value.
	VarIsCustom (see page 5090)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 5091)	Indicates whether the specified variant is unassigned.
	VarIsEmptyParam (see page 5092)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 5092)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3826).
	VarIsError (see page 5092)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3826).
	VarIsFloat (see page 5093)	Indicates whether the specified variant represents a floating-point value.
	VarIsNull (see page 5093)	Indicates whether the specified variant is Null (see page 5075).
	VarIsNumeric (see page 5094)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 5094)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 5095)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 5095)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 5095)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 5096)	Raises an EVariantOverflowError (see page 5009) exception.
	VarRangeCheckError (see page 5096)	Raises an EVariantRangeCheckError (see page 5010) exception.
	VarResultCheck (see page 5097)	Raises variant exception based on HRESULT (see page 3826) value.
	VarResultCheck (see page 5097)	Raises variant exception based on HRESULT (see page 3826) value.
	VarSameValue (see page 5097)	Indicates whether two Variants (see page 4997) have equivalent values.
	VarSupports (see page 5098)	Indicates whether the Variant (see page 3430)'s value is an interface that supports a specified interface.
	VarSupports (see page 5098)	Indicates whether the Variant (see page 3430)'s value is an interface that supports a specified interface.
	VarToDateTime (see page 5098)	Converts a specified variant to a TDateTime (see page 3335) value.
	VarToStr (see page 5099)	Converts a variant's value to a string.
	VarToStrDef (see page 5099)	Converts a variant's value to a string.
	VarToWideStr (see page 5100)	Converts a variant's value to a WideString (see page 3476).
	VarToWideStrDef (see page 5100)	Converts a variant's value to a WideString (see page 3476).
	VarType (see page 5101)	Returns the type code of a specified variant.
	VarTypeAsText (see page 5102)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 5103)	Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.
	VarTypeIsValidElementType (see page 5103)	Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.

54.3.1 Variants._DispInvoke Function

Pascal

```
procedure _DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); cdecl;
```

C++

```
__cdecl _DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void * Params);
```

File

Variants

Description

This is function Variants._DispInvoke.

54.3.2 Variants._OleVarFromInt Function

Pascal

```
procedure _OleVarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_OleVarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

This is function Variants._OleVarFromInt.

54.3.3 Variants._OleVarFromInt64 Function

Pascal

```
procedure _OleVarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_OleVarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

This is function Variants._OleVarFromInt64.

54.3.4 Variants._OleVarFromLStr Function

Pascal

```
procedure _OleVarFromLStr(var V: TVarData; const Value: AnsiString);
```

C++

```
_OleVarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

This is function Variants._OleVarFromLStr.

54.3.5 Variants._OleVarFromPStr Function

Pascal

```
procedure _OleVarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_OleVarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

This is function Variants._OleVarFromPStr.

54.3.6 Variants._OleVarFromUInt64 Function

Pascal

```
procedure _OleVarFromUInt64(var V: TVarData; const Value: UInt64);
```

C++

```
_OleVarFromUInt64(TVarData V, const UInt64 Value);
```

File

Variants

Description

This is function Variants._OleVarFromUInt64.

54.3.7 Variants._OleVarFromUStr Function

Pascal

```
procedure _OleVarFromUStr(var V: TVarData; const Value: UnicodeString);
```

C++

```
_OleVarFromUStr(TVarData V, const UnicodeString Value);
```

File

Variants

Description

This is function Variants._OleVarFromUStr.

54.3.8 Variants._OleVarFromVar Function

Pascal

```
procedure _OleVarFromVar(var Dest: TVarData; const Source: TVarData);
```

C++

```
_OleVarFromVar(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._OleVarFromVar.

54.3.9 Variants._VarAdd Function

Pascal

```
procedure _VarAdd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAdd(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarAdd.

54.3.10 Variants._VarAddRef Function

Pascal

```
procedure _VarAddRef(var V: TVarData);
```

C++

```
_VarAddRef(TVarData V);
```

File

Variants

Description

This is function Variants._VarAddRef.

54.3.11 Variants._VarAnd Function

Pascal

```
procedure _VarAnd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAnd(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarAnd.

54.3.12 Variants._VarArrayGet Function

Pascal

```
function _VarArrayGet(var A: TVarData; IndexCount: Integer; const Indices:
TVarArrayCoorArray): TVarData; cdecl;
```

C++

```
__cdecl TVarData _VarArrayGet(TVarData A, int IndexCount, const TVarArrayCoorArray Indices);
```

File

Variants

Description

This is function Variants._VarArrayGet.

54.3.13 Variants._VarArrayPut Function

Pascal

```
procedure _VarArrayPut(var A: TVarData; const Value: TVarData; IndexCount: Integer; const
Indices: TVarArrayCoorArray); cdecl;
```

C++

```
__cdecl _VarArrayPut(TVarData A, const TVarData Value, int IndexCount, const
TVarArrayCoorArray Indices);
```

File

Variants

Description

This is function Variants._VarArrayPut.

54.3.14 Variants._VarArrayRedim Function

Pascal

```
procedure _VarArrayRedim(var A: TVarData; HighBound: Integer);
```

C++

```
_VarArrayRedim(TVarData A, int HighBound);
```

File

Variants

Description

This is function Variants._VarArrayRedim.

54.3.15 Variants._VarCast Function

Pascal

```
procedure _VarCast(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCast(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

This is function Variants._VarCast.

54.3.16 Variants._VarCastOle Function

Pascal

```
procedure _VarCastOle(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCastOle(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

This is function Variants._VarCastOle.

54.3.17 Variants._VarClear Function

Pascal

```
procedure _VarClear(var V: TVarData);
```

C++

```
_VarClear(TVarData V);
```

File

Variants

Description

This is function Variants._VarClear.

54.3.18 Variants._VarClr Function

Pascal

```
procedure _VarClr(var V: TVarData);
```

C++

```
_VarClr(TVarData V);
```

File

Variants

Description

This is function Variants._VarClr.

54.3.19 Variants._VarCmpEQ Function

Pascal

```
procedure _VarCmpEQ(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpEQ(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpEQ.

54.3.20 Variants._VarCmpGE Function

Pascal

```
procedure _VarCmpGE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpGE.

54.3.21 Variants._VarCmpGT Function

Pascal

```
procedure _VarCmpGT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpGT.

54.3.22 Variants._VarCmpLE Function

Pascal

```
procedure _VarCmpLE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpLE.

54.3.23 Variants._VarCmpLT Function

Pascal

```
procedure _VarCmpLT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpLT.

54.3.24 Variants._VarCmpNE Function

Pascal

```
procedure _VarCmpNE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpNE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpNE.

54.3.25 Variants._VarCopy Function

Pascal

```
procedure _VarCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopy(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._VarCopy.

54.3.26 Variants._VarCopyNoInd Function

Pascal

```
procedure _VarCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopyNoInd(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._VarCopyNoInd.

54.3.27 Variants._VarDiv Function

Pascal

```
procedure _VarDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarDiv.

54.3.28 Variants._VarFromBool Function

Pascal

```
procedure _VarFromBool(var V: TVarData; const Value: Boolean);
```

C++

```
_VarFromBool(TVarData V, const Boolean Value);
```

File

Variants

Description

This is function Variants._VarFromBool.

54.3.29 Variants._VarFromCurr Function

Pascal

```
procedure _VarFromCurr;
```

C++

```
_VarFromCurr();
```

File

Variants

Description

This is function Variants._VarFromCurr.

54.3.30 Variants._VarFromDisp Function

Pascal

```
procedure _VarFromDisp(var V: TVarData; const Value: IDispatch);
```

C++

```
_VarFromDisp(TVarData V, const IDispatch Value);
```

File

Variants

Description

This is function Variants._VarFromDisp.

54.3.31 Variants._VarFromDynArray Function

Pascal

```
procedure _VarFromDynArray(var V: TVarData; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
_VarFromDynArray(TVarData V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

This is function Variants._VarFromDynArray.

54.3.32 Variants._VarFromInt Function

Pascal

```
procedure _VarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_VarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

This is function Variants._VarFromInt.

54.3.33 Variants._VarFromInt64 Function

Pascal

```
procedure _VarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_VarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

This is function Variants._VarFromInt64.

54.3.34 Variants._VarFromIntf Function

Pascal

```
procedure _VarFromIntf(var V: TVarData; const Value: IInterface);
```

C++

```
_VarFromIntf(TVarData V, const IInterface Value);
```

File

Variants

Description

This is function Variants._VarFromIntf.

54.3.35 Variants._VarFromLStr Function

Pascal

```
procedure _VarFromLStr(var V: TVarData; const Value: AnsiString);
```

C++

```
_VarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

This is function Variants._VarFromLStr.

54.3.36 Variants._VarFromPStr Function

Pascal

```
procedure _VarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_VarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

This is function Variants._VarFromPStr.

54.3.37 Variants._VarFromReal Function

Pascal

```
procedure _VarFromReal;
```

C++

```
_VarFromReal();
```

File

Variants

Description

This is function Variants._VarFromReal.

54.3.38 Variants._VarFromTDateTime Function

Pascal

```
procedure _VarFromTDateTime;
```

C++

```
_VarFromTDateTime();
```

File

Variants

Description

This is function Variants._VarFromTDateTime.

54.3.39 Variants._VarFromUInt64 Function

Pascal

```
procedure _VarFromUInt64(var V: TVarData; const Value: UInt64);
```

C++

```
_VarFromUInt64(TVarData V, const UInt64 Value);
```

File

Variants

Description

This is function Variants._VarFromUInt64.

54.3.40 Variants._VarFromUStr Function

Pascal

```
procedure _VarFromUStr(var V: TVarData; const Value: UnicodeString);
```

C++

```
_VarFromUStr(TVarData V, const UnicodeString Value);
```

File

Variants

Description

This is function Variants._VarFromUStr.

54.3.41 Variants._VarFromWChar Function

Pascal

```
procedure _VarFromWChar(var V: TVarData; const Value: WideChar);
```

C++

```
_VarFromWChar(TVarData V, const WideChar Value);
```

File

Variants

Description

This is function Variants._VarFromWChar.

54.3.42 Variants._VarFromWStr Function

Pascal

```
procedure _VarFromWStr(var V: TVarData; const Value: WideString);
```

C++

```
_VarFromWStr(TVarData V, const BSTR Value);
```

File

Variants

Description

This is function Variants._VarFromWStr.

54.3.43 Variants._VarMod Function

Pascal

```
procedure _VarMod(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMod(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarMod.

54.3.44 Variants._VarMul Function

Pascal

```
procedure _VarMul(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMul(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarMul.

54.3.45 Variants._VarNeg Function

Pascal

```
procedure _VarNeg(var Dest: TVarData);
```

C++

```
_VarNeg(TVarData Dest);
```

File

Variants

Description

This is function Variants._VarNeg.

54.3.46 Variants._VarNot Function

Pascal

```
procedure _VarNot(var Dest: TVarData);
```

C++

```
_VarNot(TVarData Dest);
```

File

Variants

Description

This is function Variants._VarNot.

54.3.47 Variants._VarOr Function

Pascal

```
procedure _VarOr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarOr(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarOr.

54.3.48 Variants._VarRDiv Function

Pascal

```
procedure _VarRDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarRDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarRDiv.

54.3.49 Variants._VarShl Function

Pascal

```
procedure _VarShl(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShl(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarShl.

54.3.50 Variants._VarShr Function

Pascal

```
procedure _VarShr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShr(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarShr.

54.3.51 Variants._VarSub Function

Pascal

```
procedure _VarSub(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarSub(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarSub.

54.3.52 Variants._VarToBool Function

Pascal

```
function _VarToBool(const V: TVarData): LongBool;
```

C++

```
LongBool _VarToBool(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToBool.

54.3.53 Variants._VarToCurrency Function

Pascal

```
function _VarToCurrency(const V: TVarData): Currency;
```

C++

```
Currency _VarToCurrency(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToCurrency.

54.3.54 Variants._VarToDisp Function

Pascal

```
procedure _VarToDisp(var Dispatch: IDispatch; const V: TVarData);
```

C++

```
_VarToDisp(IDispatch Dispatch, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToDisp.

54.3.55 Variants._VarToDynArray Function

Pascal

```
procedure _VarToDynArray(var DynArray: Pointer; const V: TVarData; TypeInfo: Pointer);
```

C++

```
_VarToDynArray(void * DynArray, const TVarData V, void * TypeInfo);
```

File

Variants

Description

This is function Variants._VarToDynArray.

54.3.56 Variants._VarToInt64 Function

Pascal

```
function _VarToInt64(const V: TVarData): Int64;
```

C++

```
Int64 _VarToInt64(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToInt64.

54.3.57 Variants._VarToInteger Function

Pascal

```
function _VarToInteger(const V: TVarData): Integer;
```

C++

```
int _VarToInteger(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToInteger.

54.3.58 Variants._VarToIntf Function

Pascal

```
procedure _VarToIntf(var Intf: IInterface; const V: TVarData);
```

C++

```
_VarToIntf(IInterface Intf, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToIntf.

54.3.59 Variants._VarToLStr Function

Pascal

```
procedure _VarToLStr(var S: AnsiString; const V: TVarData);
```

C++

```
_VarToLStr(AnsiString S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToLStr.

54.3.60 Variants._VarToPStr Function

Pascal

```
procedure _VarToPStr(var S; const V: TVarData);
```

C++

```
_VarToPStr( S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToPStr.

54.3.61 Variants._VarToReal Function

Pascal

```
function _VarToReal(const V: TVarData): Extended;
```

C++

```
Extended _VarToReal(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToReal.

54.3.62 Variants._VarToUInt64 Function

Pascal

```
function _VarToUInt64(const V: TVarData): UInt64;
```

C++

```
UInt64 _VarToUInt64(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToUInt64.

54.3.63 Variants._VarToUStr Function

Pascal

```
procedure _VarToUStr(var S: UnicodeString; const V: TVarData);
```

C++

```
_VarToUStr(UnicodeString S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToUStr.

54.3.64 Variants._VarToWStr Function

Pascal

```
procedure _VarToWStr(var S: WideString; const V: TVarData);
```

C++

```
_VarToWStr(BSTR S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToWStr.

54.3.65 Variants._VarXor Function

Pascal

```
procedure _VarXor(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarXor(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarXor.

54.3.66 Variants.DynArrayFromVariant Function

Creates a dynamic array from a Variant (see page 3430).

Pascal

```
procedure DynArrayFromVariant(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
```

C++

```
DynArrayFromVariant(void * DynArray, const Variant V, void * TypeInfo);
```

File

Variants

Description

Call DynArrayFromVariant to copy the contents of a variant (V) to an existing dynamic array variable (DynArray).

DynArray is a dynamic array variable whose value is assigned from the Variant (see page 3430).

V is the Variant (see page 3430) to copy. Its value must be an array.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayToVariant (see page 5072)

TypeInfo

54.3.67 Variants.DynArrayToVariant Function

Creates a Variant (see page 3430) array from a dynamic array.

Pascal

```
procedure DynArrayToVariant(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
DynArrayToVariant(Variant V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

Call DynArrayToVariant to assign the contents of a dynamic array as the value of a Variant (see page 3430) variable.

V is the Variant (see page 3430) whose value is assigned by DynArrayToVariant.

DynArray is a dynamic array to copy.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayFromVariant (see page 5072)

TypeInfo

54.3.68 Variants.FindCustomVariantType Function

Retrieves the object that implements a custom Variant (see page 3430) type.

Pascal

```
function FindCustomVariantType(const AVarType: TVarType; out CustomVariantType:
TCustomVariantType): Boolean; overload;
function FindCustomVariantType(const TypeName: string; out CustomVariantType:
TCustomVariantType): Boolean; overload;
```

C++

```
Boolean FindCustomVariantType(const TVarType AVarType, TCustomVariantType CustomVariantType);
Boolean FindCustomVariantType(const UnicodeString TypeName, TCustomVariantType
CustomVariantType);
```

File

Variants

Description

Use FindCustomVariantType to obtain a reference to the object that implements a custom Variant (see page 3430) type.

TypeName is the name of the TCustomVariantType (see page 5011) descendant class you want to retrieve.

AVarType is the type code of Variant (see page 3430) whose implementation class you want to retrieve. If you have an instance of a Variant (see page 3430) whose value is a custom Variant (see page 3430) Type, you can obtain a value for this parameter using the VarType (see page 5101) function.

CustomVariantType returns an instance of the TCustomVariantType (see page 5011) descendant that implements the indicated custom Variant (see page 3430) type.

See Also

VarType (see page 5101)

VarIsCustom (see page 5090)

54.3.69 Variants.FindVarData Function

Returns pointer to variant's data.

Pascal

```
function FindVarData(const V: Variant): PVarData;
```

C++

```
PVarData FindVarData(const Variant V);
```

File

Variants

Description

FindVarData returns a pointer to the Variant (see page 3430) data contained referenced by a Variant (see page 3430). If the Variant (see page 3430) contains another Variant (see page 3430), FindVarData follows the chain of references until it finds the actual data.

See Also

VarType (see page 5101)

54.3.70 Variants.HandleConversionException Function

Handles exceptions during variant type conversions.

Pascal

```
procedure HandleConversionException(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
HandleConversionException(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

HandleConversionException handles exceptions resulting from attempts to convert between variant types. Code that implements such conversions should call HandleConversionException when a conversion throws an exception.

ASourceType and ADestType indicate the variant types of the source and destination. HandleConversionException uses these to format an appropriate error string. If the current exception corresponds to a variant exception, HandleConversionException raises that exception. Otherwise, HandleConversionException simply re-raises the original exception.

See Also

VarResultCheck (see page 5097)

EVariantError

54.3.71 Variants.Null Function

Returns a Null variant.

Pascal

```
function Null: Variant;
```

C++

```
Variant Null();
```

File

Variants

Description

Use Null to obtain a Null variant that can indicate unknown or missing data. Null Variants (see page 4997) can be assigned to variant variables in an application that must contain a null value. Assigning Null to a variant variable does not cause an error, and Null can be returned from any function with a variant return value.

Assigning Null to a variable of any type other than Variant (see page 3430) causes either a compile-time error or raises an EVariantTypeMismatch exception. For example, in the following code the assignment of v, the Null variant, to variant q is successful; whereas the conversion of variant v, which is now Null, to the integer return type of the Test function, raises an exception.

function Test(v: Variant (see page 3430)): Integer;

```
var
  q: Variant;
  msg: string;
begin
  q := v; { this is ok, since q is a variant }
  if VarIsNull(q) then
    msg := 'q is a null variant'
  else
    msg := 'q is not a null variant';
  ShowMessage(msg);

  Result := v; { this raises an exception!!!}
end;
```

procedure TForm1.Button1Click(Sender: TObject (see page 3364));

```
begin
  Test(Null);
end;
```

int Test(Variant (see page 3430) v)

```
{
  Variant q = v; // this is ok, since q is a variant
  AnsiString msg;
  if (VarIsNull(q))
    msg = "q is a null variant";
  else
    msg = "q is not a null variant";
  ShowMessage(msg);

   return v; // this throws an exception!!!
```

```
}
void __fastcall TForm1::Button1Click(TObject (see page 3364) *Sender)
{
  Test(Null());
}
```

Except for comparisons, which always result in Boolean values, expressions involving Null variants always result in Null variants.

Note: The Null variant returned by the Null function is different from the nil constant in Delphi code and the NULL macro in C++ code. Neither nil nor NULL is a variant. The nil constant is an object reference that does not refer to any actual object. The NULL macro is a pointer that does not refer to any actual object or value.

See Also

Unassigned (see page 5076)

VarIsNull (see page 5093)

VarType (see page 5101)

NullEqualityRule (see page 5112)

NullMagnitudeRule (see page 5113)

54.3.72 Variants.Unassigned Function

Returns an "empty" variant.

Pascal

```
function Unassigned: Variant;
```

C++

```
Variant Unassigned();
```

File

Variants

Description

A variant variable can be "empty", meaning it has not yet been assigned to. The Unassigned function returns an empty variant, which can be assigned to a variant variable to restore the variable to its initial state.

Use the VarIsEmpty (see page 5091) function to test whether a variant is empty. When used on an empty variant, the VarType (see page 5101) standard function returns varEmpty (see page 3933).

If an empty variant is cast to another type (for example, by assigning to a non-variant variable or calling VarAsType (see page 5083)) the following conversions occur:

Destination	Conversion result
Numeric type	Zero
String type	Empty string
Boolean	False.

Note: Unassigned is useful with variants referencing OLE Automation Objects that you want to keep "alive" until another value is

assigned to the variant.

There's corresponding functionality under Linux. Should document when time permits.

See Also

Null (see page 5075)

VarIsEmpty (see page 5091)

VarType (see page 5101)

VarAsType (see page 5083)

54.3.73 Variants.VarArrayAsPSafeArray Function

Pascal

```
function VarArrayAsPSafeArray(const A: Variant): PVarArray;
```

C++

```
PVarArray VarArrayAsPSafeArray(const Variant A);
```

File

Variants

Description

This is function Variants.VarArrayAsPSafeArray.

54.3.74 Variants.VarArrayCreate Function

Creates a variant array.

Pascal

```
function VarArrayCreate(const Bounds: array of Integer; AVarType: TVarType): Variant;
```

C++

```
Variant VarArrayCreate(const array of Integer Bounds, TVarType AVarType);
```

File

Variants

Description

VarArrayCreate creates a variant array with the bounds given by Bounds (see page 4685) and the element type given by VarType (see page 5101).

The Bounds (see page 4685) parameter must contain an even number of values, where each pair of values specifies the upper and lower bounds of one dimension of the array.

Note: The Bounds_size parameter is the index of the last value in the Bounds (see page 4685) array. This is one less than the total number of values. Thus, Bounds_size must always be an odd number.

The element type of the array, given by the VarType (see page 5101) parameter, is a variant type code. This must be one of the constants defined in the System (see page 3165) unit. It cannot include the varArray (see page 3929) or varByRef (

see page 3930) bits. The element type cannot be varString (see page 3938) or a custom Variant (see page 3430) type. To create a variant array of strings use the varOleStr (see page 3936) type code. If the element type is varVariant (see page 3941), the elements of the array are themselves variants and can in turn contain variant arrays.

If VarArrayCreate is unable to create the array, an EVariantArrayCreateError (see page 5002) is raised.

Note: :Variant (see page 3430) arrays with an element type of varByte (see page 3931) are the preferred method of passing binary data between OLE Automation controllers and servers. Such arrays are subject to no translation of their data, and can be efficiently accessed using the VarArrayLock (see page 5080) and VarArrayUnlock (see page 5083) routines.

There's corresponding functionality under Linux. Should document when time permits.

See Also

VarArrayOf (see page 5081)

VarArrayLock (see page 5080)

VarArrayUnlock (see page 5083)

EVariantArrayCreateError (see page 5002)

54.3.75 Variants.VarArrayCreateError Function

Indicates an array creation error.

Pascal

```
procedure VarArrayCreateError;
```

C++

```
VarArrayCreateError();
```

File

Variants

See Also

EVariantArrayCreateError (see page 5002)

54.3.76 Variants.VarArrayDimCount Function

Returns number of dimensions of a variant array.

Pascal

```
function VarArrayDimCount(const A: Variant): Integer;
```

C++

```
int VarArrayDimCount(const Variant A);
```

File

Variants

Description

VarArrayDimCount returns the number of dimensions of the given variant. The resulting value is zero if the given variant is not a

variant array.

See Also

VarArrayHighBound (see page 5079)

VarArrayLowBound (see page 5080)

54.3.77 Variants.VarArrayGet Function

Returns a Variant (see page 3430) that represents a single value from a multi-dimensional Variant (see page 3430) array.

Pascal

```
function VarArrayGet(const A: Variant; const Indices: array of Integer): Variant;
```

C++

```
Variant VarArrayGet(const Variant A, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayGet to obtain the value of a single cell from a multi-dimensional Variant (see page 3430) array.

A is the Variant (see page 3430) array that contains the desired value.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayGet raises an EVariantBadIndexError (see page 5004).

See Also

VarArrayPut (see page 5081)

EVariantBadIndexError (see page 5004)

54.3.78 Variants.VarArrayHighBound Function

Returns high bound for a dimension in a variant array.

Pascal

```
function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayHighBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayHighBound returns the high bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an

array (EVariantInvalidArgError (see page 5005)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 5004)).

See Also

VarArrayDimCount (see page 5078)

VarArrayLowBound (see page 5080)

EVariantInvalidArgError (see page 5005)

EVariantBadIndexError (see page 5004)

54.3.79 Variants.VarArrayLock Function

Locks a variant array and returns a pointer to the data.

Pascal

```
function VarArrayLock(const A: Variant): Pointer;
```

C++

```
void * VarArrayLock(const Variant A);
```

File

Variants

Description

VarArrayLock locks the given variant array and returns a pointer to the data stored in the array. While locked, a variant array can not be resized and any calls to VarArrayRedim (see page 3786) will fail. Once locked, a variant array must later be unlocked using VarArrayUnlock (see page 5083). An EVariantInvalidArgError (see page 5005) exception is raise if the variant given by A is not an array.

The pointer returned by VarArrayLock points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, VarArrayLock can be used to gain direct access to the array data, which results in improved performance.

See Also

VarArrayUnlock (see page 5083)

VarArrayRedim

54.3.80 Variants.VarArrayLowBound Function

Returns the low bound of a dimension in a variant array.

Pascal

```
function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayLowBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayLowBound returns the low bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an array (EVariantInvalidArgError (see page 5005)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 5004))..

See Also

VarArrayDimCount (see page 5078)

VarArrayHighBound (see page 5079)

EVariantInvalidArgError (see page 5005)

EVariantBadIndexError (see page 5004)

54.3.81 Variants.VarArrayOf Function

Creates and fills a one-dimensional variant array.

Pascal

```
function VarArrayOf(const Values: array of Variant): Variant;
```

C++

```
Variant VarArrayOf(const array of Variant Values);
```

File

Variants

Description

VarArrayOf returns a one-dimensional variant array with the elements given by the Values parameter

Values is an array of Variant (see page 3430) values to convert into a Variant (see page 3430) array.

Note: Values_Size indicates the index of the last value in the Values array (one less than the number of values).

The low bound of the returned array is zero, the high bound is the number of values given by the Values parameter less one (Values_Size), and the element type is Variant (see page 3430).

See Also

VarArrayCreate (see page 5077)

54.3.82 Variants.VarArrayPut Function

Sets the value of a single cell in a multi-dimensional Variant (see page 3430) array.

Pascal

```
procedure VarArrayPut(var A: Variant; const Value: Variant; const Indices: array of Integer);
```

C++

```
VarArrayPut(Variant A, const Variant Value, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayPut to set the value of a single cell in a multi-dimensional Variant (see page 3430) array.

A is the Variant (see page 3430) array that contains the cell to set.

Value is the value to assign to the specified cell.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayPut raises an EVariantBadIndexError (see page 5004).

See Also

VarArrayGet (see page 5079)

EVariantBadIndexError (see page 5004)

54.3.83 Variants.VarArrayRef Function

Returns a reference to the specified variant array.

Pascal

```
function VarArrayRef(const A: Variant): Variant;
```

C++

```
Variant VarArrayRef(const Variant A);
```

File

Variants

Description

Use VarArrayRef to reference a variant array when a referenced variant array is expected in an API call. VarArrayRef creates a new variant array of the same type as A, but with an extra level of indirection if A does not include the varByRef (see page 3930) type bit. The returned variant includes the varByRef (see page 3930) type bit.

If A is not an array, VarArrayRef raises an EVariantInvalidArgError (see page 5005) exception.

See Also

VarArrayCreate (see page 5077)

TVarType

VarCopyNoInd (see page 5086)

EVariantInvalidArgError (see page 5005)

54.3.84 Variants.VarArrayUnlock Function

Unlocks a variant array.

Pascal

```
procedure VarArrayUnlock(const A: Variant);
```

C++

```
VarArrayUnlock(const Variant A);
```

File

Variants

Description

VarArrayUnlock unlocks a variant array that was previously locked by VarArrayLock (see page 5080). If the variant given by A is not an array, VarArrayUnlock raises an EVariantInvalidArtgError exception.

See Also

VarArrayLock (see page 5080)

54.3.85 Variants.VarAsError Function

Converts a HRESULT (see page 3826) into an Error (see page 3662) variant.

Pascal

```
function VarAsError(AResult: HRESULT): Variant;
```

C++

```
Variant VarAsError(HRESULT AResult);
```

File

Variants

Description

VarAsError converts an HRESULT (see page 3826) result code into an equivalent variant of type varError (see page 3934).

See Also

VarType (see page 5101)

54.3.86 Variants.VarAsType Function

Converts a variant to specified type.

Pascal

```
function VarAsType(const V: Variant; AVarType: TVarType): Variant;
```

C++

```
Variant VarAsType(const Variant V, TVarType AVarType);
```

File

Variants

Description

VarAsType converts a variant to a specified type and returns a new Variant (see page 3430) that has the specified type.

V is the Variant (see page 3430) to convert.

VarType (see page 5101) is a variant type code that indicates the type to which V should be converted. This can be one of the constants defined in the System (see page 3165) unit, or it can be the type code for a custom variant type. It cannot include the varArray (see page 3929) or varByRef (see page 3930) bits.

VarAsType raises an exception if the variant cannot be converted to the given type. This will be EVariantError (see page 4014) or one of its descendants, such as EVariantOverflowError (see page 5009) or EVariantTypeMismatchError.

See Also

VarType (see page 5101)

54.3.87 Variants.VarCastError Function

Raises an EVariantTypeCastError (see page 5010) exception.

Pascal

```
procedure VarCastError; overload;
procedure VarCastError(const ASourceType: TVarType; const ADestType: TVarType); overload;
```

C++

```
VarCastError();
VarCastError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarCastError to indicate that a variant operation has failed due to cast error. VarCastError raises EVariantTypeCastError (see page 5010). If no arguments are specified, the error string is a general error message. If ASourceType and ADestType are specified, the error string incorporates the names of the specified Variant (see page 3430) Types (see page 4683).

See Also

EVariantTypeCastError (see page 5010)

TVarData

VarType (see page 5101)

54.3.88 Variants.VarCheckEmpty Function

Raises an exception if a specified variant's value is Unassigned (see page 5076).

Pascal

```
procedure VarCheckEmpty(const V: Variant);
```

C++

```
VarCheckEmpty(const Variant V);
```

File

Variants

Description

Call VarCheckEmpty when your code requires that a Variant (see page 3430) have a value, and you want to raise an exception if it doesn't. VarCheckEmpty checks whether the Variant (see page 3430) specified by V has a value of Unassigned (see page 5076). If so, it raises an EVariantTypeMismatchError exception.

To check whether the Variant (see page 3430) is Unassigned (see page 5076) without raising an exception, use the VarIsEmpty (see page 5091) function instead. To check whether the Variant (see page 3430)'s value is undefined, use the VarIsClear (see page 5090) function instead.

Note: Do not confuse an Unassigned (see page 5076) variant with a Variant (see page 3430) whose value is Null (see page 5075). A Null (see page 5075) variant is still assigned, but has the value Null (see page 5075). Unlike unassigned variants, Null (see page 5075) variants can be used in expressions and can be converted to other types of variants.

See Also

VarIsEmpty (see page 5091)

Unassigned (see page 5076)

VarType (see page 5101)

VarIsNull (see page 5093)

VarIsClear (see page 5090)

EVariantTypeMismatchError

54.3.89 Variants.VarCompareValue Function

Returns the relationship between Variants (see page 4997).

Pascal

```
function VarCompareValue(const A: Variant; const B: Variant): TVariantRelationship;
```

C++

```
TVariantRelationship VarCompareValue(const Variant A, const Variant B);
```

File

Variants

Description

Call VarCompareValue to determine the relationship between two Variants (see page 4997).

A and B are the values to compare.

VarCompareValue returns a value that indicates whether A and B are equal (vrEqual), whether one is greater than the other (vrLessThan or vrGreaterThan), or whether the two values can't be ordered relative to each other (vrNotEqual).

See Also

VarSameValue (see page 5097)

CompareValue

54.3.90 Variants.VarCopyNoInd Function

Copies a Variant (see page 3430).

Pascal

```
procedure VarCopyNoInd(var Dest: Variant; const Source: Variant);
```

C++

```
VarCopyNoInd(Variant Dest, const Variant Source);
```

File

Variants

Description

VarCopyNoInd copies the variant given by Source into the variant given by Dest. Calling VarCopyNoInd in C++ or Delphi code is equivalent to a simple variant assignment in Delphi code.

Dest can be a Variant (see page 3430) or an OleVariant (see page 3281), and it must be possible to assign a value to it.

Source is defined as a Variant (see page 3430), but an OleVariant (see page 3281) is also allowed because it is a compatible type. However, Source can't have a VarType (see page 5101) that includes the varByRef (see page 3930) bit.

The following table lists the expected behavior when using Variant (see page 3430) and OleVariant (see page 3281) types as parameters:

Dest	Source	Behavior
Variant (see page 3430)	OleVariant (see page 3281)	Works as expected.
Variant (see page 3430)	Variant (see page 3430)	Works as expected.
OleVariant (see page 3281)	OleVariant (see page 3281)	Works as expected.
OleVariant (see page 3281)	Variant (see page 3430)	Source should be implicitly cast to an OleVariant (see page 3281).

54.3.91 Variants.VarEnsureRange Function

Returns a Variant (see page 3430) with a value in a specified range.

Pascal

```
function VarEnsureRange(const AValue: Variant; const AMin: Variant; const AMax: Variant):
Variant;
```

C++

```
Variant VarEnsureRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarEnsureRange returns a Variant (see page 3430) whose value is guaranteed to be within a specified range.

AValue is a Variant (see page 3430) with the value you want as long as it is in range. If AValue is in range, VarEnsureRange returns AValue.

AMin is a Variant (see page 3430) with the minimum acceptable value. If AValue is less than AMin, VarEnsureRange returns AMin.

AMax is a Variant (see page 3430) with the maximum acceptable value. If AValue is greater than AMax, VarEnsureRange returns AMax.

See Also

VarInRange (see page 5087)

54.3.92 Variants.VarFromDateTime Function

Returns a variant containing a specified data and time.

Pascal

```
function VarFromDateTime(const DateTime: TDateTime): Variant;
```

C++

```
Variant VarFromDateTime(const TDateTime DateTime);
```

File

Variants

Description

VarFromDateTime returns a Variant (see page 3430) that represents the date and time of the given TDateTime (see page 3335) value, DateTime. The type code of the resulting variant is varDate (see page 3932).

See Also

VarToDateTime (see page 5098)

VarType (see page 5101)

54.3.93 Variants.VarInRange Function

Indicates whether a Variant (see page 3430)'s value is in a specified range.

Pascal

```
function VarInRange(const AValue: Variant; const AMin: Variant; const AMax: Variant): Boolean;
```

C++

```
Boolean VarInRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarInRange checks whether a Variant (see page 3430) has a value in a specified range.

AValue is a Variant (see page 3430) to check.

AMin is a Variant (see page 3430) with the minimum acceptable value.

AMax is a Variant (see page 3430) with the maximum acceptable value.

VarInRange returns true if AValue is greater than or equal to AMin and less than or equal to AMax. Otherwise, it returns false.

See Also

VarEnsureRange (see page 5086)

54.3.94 Variants.VarInvalidNullOp Function

Indicate variant operation on null.

Pascal

```
procedure VarInvalidNullOp;
```

C++

```
VarInvalidNullOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 5007)

54.3.95 Variants.VarInvalidOp Function

Indicate invalid variant operation.

Pascal

```
procedure VarInvalidOp;
```

C++

```
VarInvalidOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 5007)

54.3.96 Variants.VarIsArray Function

Indicates whether the specified variant is an array.

Pascal

```
function VarIsArray(const A: Variant): Boolean; overload;
function VarIsArray(const A: Variant; AResolveByRef: Boolean): Boolean; overload;
```

C++

```
Boolean VarIsArray(const Variant A);
Boolean VarIsArray(const Variant A, Boolean AResolveByRef);
```

File

Variants

Description

VarIsArray returns true if the given variant is an array. Otherwise, the function result is false.

A is the Variant (see page 3430) to check.

AResolveByRef indicates how VarIsArray should handle the case when A is a reference to a Variant (see page 3430) (when the VarType (see page 5101) of A is varByRef (see page 3930) or varVariant (see page 3941)).When AResolveByRef is true, VarIsArray checks whether A is an array or A points to an array. When AResolveByRef is false, VarIsArray returns false when A has a VarType (see page 5101) of varByRef (see page 3930) or varVariant (see page 3941).

See Also

VarArrayCreate (see page 5077)

VarType (see page 5101)

VarIsType (see page 5095)

54.3.97 Variants.VarIsByRef Function

Indicates whether the specified variant's value is by reference.

Pascal

```
function VarIsByRef(const V: Variant): Boolean;
```

C++

```
Boolean VarIsByRef(const Variant V);
```

File

Variants

Description

VarIsByRef returns true if the given variant's type code includes the includes the varByRef (see page 3930) bit. Otherwise, the function result is false.

See Also

VarType (see page 5101)

VarIsType (see page 5095)

54.3.98 Variants.VarIsClear Function

Indicates whether the specified variant has an undefined value.

Pascal

```
function VarIsClear(const V: Variant): Boolean;
```

C++

```
Boolean VarIsClear(const Variant V);
```

File

Variants

Description

VarIsClear returns true if the given variant's value is undefined. The value can be undefined for any of several reasons:

The Variant (see page 3430) may have had its value set to Unassigned (see page 5076).

The Variant (see page 3430)'s value may be an interface type that has been set to nil (Delphi) or NULL (C++).

The Variant (see page 3430) may be a custom variant that returns true from its IsClear method.

In all other cases, the function result is false.

Note: Do not confuse an unassigned variant with a Null (see page 5075) variant. A Null (see page 5075) variant is still assigned, but has the value Null (see page 5075). Unlike unassigned variants, Null (see page 5075) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 5076)

VarType (see page 5101)

VarCheckEmpty (see page 5084)

VarIsNull (see page 5093)

VarIsEmpty (see page 5091)

VarIsType (see page 5095)

TCustomVariantType (see page 5011)

54.3.99 Variants.VarIsCustom Function

Indicates whether the specified variant is a custom variant.

Pascal

```
function VarIsCustom(const V: Variant): Boolean;
```

C++

```
Boolean VarIsCustom(const Variant V);
```

File

Variants

Description

VarIsCustom returns true if the given variant represents a custom variant type. Otherwise, the function result is false.

See Also

VarType (see page 5101)

FindCustomVariantType (see page 5073)

VarIsType (see page 5095)

54.3.100 Variants.VarIsEmpty Function

Indicates whether the specified variant is unassigned.

Pascal

```
function VarIsEmpty(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmpty(const Variant V);
```

File

Variants

Description

VarIsEmpty returns true if the given variant contains the value Unassigned (see page 5076). If the variant contains any other value, the function result is false.

Note: VarIsEmpty does not indicate when a Variant (see page 3430) that represents an interface has had the interface pointer cleared or when a custom Variant (see page 3430) is clear. If either of these situations may exist, use the VarIsClear (see page 5090) method instead.

Note: Do not confuse an unassigned variant with a Null (see page 5075) variant. A Null (see page 5075) variant is still assigned, but has the value Null (see page 5075). Unlike unassigned variants, Null (see page 5075) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 5076)

VarType (see page 5101)

VarCheckEmpty (see page 5084)

VarIsNull (see page 5093)

VarIsClear (see page 5090)

VarIsEmptyParam (see page 5092)

VarIsType (see page 5095)

54.3.101 Variants.VarIsEmptyParam Function

Indicates whether the specified variant represents an unassigned optional parameter.

Pascal

```
function VarIsEmptyParam(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmptyParam(const Variant V);
```

File

Variants

Description

VarIsEmptyParam returns true if the given variant represents an unassigned optional parameter. If the variant contains any other value, the function result is false.

See Also

VarIsClear (see page 5090)

VarIsEmpty (see page 5091)

VarIsNull (see page 5093)

EmptyParam (see page 5111)

VarIsType (see page 5095)

54.3.102 Variants.VarIsError Function

Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3826).

Pascal

```
function VarIsError(const V: Variant; out AResult: HRESULT): Boolean; overload;
function VarIsError(const V: Variant): Boolean; overload;
```

C++

```
Boolean VarIsError(const Variant V, HRESULT AResult);
Boolean VarIsError(const Variant V);
```

File

Variants

Description

VarIsError returns true if the given variant is an Error (see page 3662) variant (type varError (see page 3934)). If AResult is specified, it is set to the corresponding result code.

See Also

VarType (see page 5101)

VarIsType (see page 5095)

54.3.103 Variants.VarIsFloat Function

Indicates whether the specified variant represents a floating-point value.

Pascal

```
function VarIsFloat(const V: Variant): Boolean;
```

C++

```
Boolean VarIsFloat(const Variant V);
```

File

Variants

Description

VarIsFloat returns true if the given variant represents a floating-point value (In Delphi: Single, Double, or Currency (see page 3242); In C++: float, double, or Currency (see page 3242)). If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 5094)

VarIsNumeric (see page 5094)

VarIsStr (see page 5095)

VarType (see page 5101)

VarIsType (see page 5095)

54.3.104 Variants.VarIsNull Function

Indicates whether the specified variant is Null (see page 5075).

Pascal

```
function VarIsNull(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNull(const Variant V);
```

File

Variants

Description

VarIsNull returns true if the given variant contains the value Null (see page 5075). If the variant contains any other value, the function result is false.

Note: Do not confuse a Null (see page 5075) variant with an unassigned variant. A Null (see page 5075) variant is still assigned, but has the value Null (see page 5075). Unlike unassigned variants, Null (see page 5075) variants can be used in expressions and can be converted to other types of variants.

See Also

Null (see page 5075)

VarType (see page 5101)

VarIsEmpty (see page 5091)

VarIsClear (see page 5090)

VarIsEmptyParam (see page 5092)

VarIsType (see page 5095)

54.3.105 Variants.VarIsNumeric Function

Indicates whether the specified variant represents a numeric value.

Pascal

```
function VarIsNumeric(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNumeric(const Variant V);
```

File

Variants

Description

VarIsNumeric returns true if the given variant's type code indicates a numeric value of some sort. This can be either a floating-point value or an ordinal value. If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 5094)

VarIsFloat (see page 5093)

VarIsStr (see page 5095)

VarType (see page 5101)

VarIsType (see page 5095)

54.3.106 Variants.VarIsOrdinal Function

Indicates whether the specified variant represents an ordinal value.

Pascal

```
function VarIsOrdinal(const V: Variant): Boolean;
```

C++

```
Boolean VarIsOrdinal(const Variant V);
```

File

Variants

Description

VarIsOrdinal returns true if the given variant represents an ordinal value (such as an integer or boolean type). If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 5094)

VarIsFloat (see page 5093)

VarIsStr (see page 5095)

VarType (see page 5101)

VarIsType (see page 5095)

54.3.107 Variants.VarIsStr Function

Indicates whether the specified variant represents a string value.

Pascal

```
function VarIsStr(const V: Variant): Boolean;
```

C++

```
Boolean VarIsStr(const Variant V);
```

File

Variants

Description

VarIsStr returns true if the given variant's type code is varString (see page 3938) or varOleStr (see page 3936). If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 5094)

VarIsFloat (see page 5093)

VarIsOrdinal (see page 5094)

VarType (see page 5101)

VarIsType (see page 5095)

54.3.108 Variants.VarIsType Function

Indicates whether the specified variant represents a specified type.

Pascal

```
function VarIsType(const V: Variant; AVarType: TVarType): Boolean; overload;
function VarIsType(const V: Variant; const AVarTypes: array of TVarType): Boolean; overload;
```

C++

```
Boolean VarIsType(const Variant V, TVarType AVarType);
Boolean VarIsType(const Variant V, const array of TVarType AVarTypes);
```

File

Variants

Description

VarIsType checks whether the type code for the Variant (see page 3430) specified by V matches the specified type code or one of the type codes in the specified array.

V is the Variant (see page 3430) to check.

AVarType is the type code against which to check V.

AVarTypes is an array of valid type codes against which to check V.

AVarTypes_size is the index of the last entry in AVarTypes (one less than the number of entries).

VarIsType returns true if the given variant's type code matches the supplied type code or codes, false otherwise.

See Also

VarType (see page 5101)

54.3.109 Variants.VarOverflowError Function

Raises an EVariantOverflowError (see page 5009) exception.

Pascal

```
procedure VarOverflowError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarOverflowError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarOverflowError to indicate that a variant operation has failed due to numeric overflow. ASourceType and ADestType should contain the variant type codes for the source and destination variant types. VarOverflowError uses this information to compose an error string and raise EVariantOverflowError (see page 5009).

See Also

EVariantOverflowError (see page 5009)

TVarType

54.3.110 Variants.VarRangeCheckError Function

Raises an EVariantRangeCheckError (see page 5010) exception.

Pascal

```
procedure VarRangeCheckError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarRangeCheckError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarRangeCheckError to indicate that a variant operation has failed because the result is not in the allowed range for the destination variant type. ASourceType and ADestType are the variant types of the source and destination.

See Also

EVariantRangeCheckError (see page 5010)

54.3.111 Variants.VarResultCheck Function

Raises variant exception based on HRESULT (see page 3826) value.

Pascal

```
procedure VarResultCheck(AResult: HRESULT); overload;
procedure VarResultCheck(AResult: HRESULT; ASourceType: TVarType; ADestType: TVarType);
overload;
```

C++

```
VarResultCheck(HRESULT AResult);
VarResultCheck(HRESULT AResult, TVarType ASourceType, TVarType ADestType);
```

File

Variants

Description

VarResultCheck examines the HRESULT (see page 3826) value returned by many variant routines. If AResult is S_OK (see page 3927), VarResultCheck does nothing. Otherwise, VarResultCheck attempts to map AResult to a variant exception and raise that exception. If VarResultCheck is unable to map AResult to a specific variant Exception (see page 4015), it raises EVariantError (see page 4014).

Specify ASourceType and ADestType when the operation involves conversion between two variant types.

See Also

EVariantError

54.3.112 Variants.VarSameValue Function

Indicates whether two Variants (see page 4997) have equivalent values.

Pascal

```
function VarSameValue(const A: Variant; const B: Variant): Boolean;
```

C++

```
Boolean VarSameValue(const Variant A, const Variant B);
```

File

Variants

Description

VarSameValue returns true when A has the same value as B, false if the two values are not equivalent.

If A and B can't be compared, VarSameValue raises an exception.

See Also

VarCompareValue (see page 5085)

SameValue

54.3.113 Variants.VarSupports Function

Indicates whether the Variant (see page 3430)'s value is an interface that supports a specified interface.

Pascal

```
function VarSupports(const V: Variant; const IID: TGUID; out Intf): Boolean; overload;
function VarSupports(const V: Variant; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean VarSupports(const Variant V, const TGUID IID,  Intf);
Boolean VarSupports(const Variant V, const TGUID IID);
```

File

Variants

Description

VarSupports returns the lower twelve bits of a variant type code. These indicate the type of the variant. For a list of possible variant type codes, see VarType (see page 5101).

Call VarSupports to determine whether the Variant (see page 3430) represents an interface that supports the interface identified by the IID parameter. If V supports the interface, VarSupports returns the interface as the Intf parameter (if using the first syntax) and returns true (both versions). If the interface specified by IID is not supported, VarSupports returns false.

Unlike calling the QueryInterface method of the interface, VarSupports handles the case when the Variant (see page 3430)'s value is nil (Delphi) or NULL (C++).

See Also

Supports

54.3.114 Variants.VarToDateTime Function

Converts a specified variant to a TDateTime (see page 3335) value.

Pascal

```
function VarToDateTime(const V: Variant): TDateTime;
```

C++

```
TDateTime VarToDateTime(const Variant V);
```

File

Variants

Description

VarToDateTime converts the given variant to a TDateTime (see page 3335) value. VarToDateTime raises an exception if the conversion fails.

See Also

VarFromDateTime (see page 5087)

54.3.115 Variants.VarToStr Function

Converts a variant's value to a string.

Pascal

```
function VarToStr(const V: Variant): string;
```

C++

```
UnicodeString VarToStr(const Variant V);
```

File

Variants

Description

VarToStr converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStr returns an empty string.

See Also

VarToStrDef (see page 5099)

VarToWideStr (see page 5100)

VarToWideStrDef (see page 5100)

54.3.116 Variants.VarToStrDef Function

Converts a variant's value to a string.

Pascal

```
function VarToStrDef(const V: Variant; const ADefault: string): string;
```

C++

```
UnicodeString VarToStrDef(const Variant V, const UnicodeString ADefault);
```

File

Variants

Description

VarToStrDef converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStrDef returns ADefault.

See Also

VarToStr (see page 5099)

VarToWideStr (see page 5100)

VarToWideStrDef (see page 5100)

54.3.117 Variants.VarToWideStr Function

Converts a variant's value to a WideString (see page 3476).

Pascal

```
function VarToWideStr(const V: Variant): WideString;
```

C++

```
BSTR VarToWideStr(const Variant V);
```

File

Variants

Description

VarToWideStr converts the data in the variant V to a WideString (see page 3476) and returns the result. If the variant has a null value, VarToWideStr returns an empty string.

See Also

VarToStrDef (see page 5099)

VarToStr (see page 5099)

VarToWideStrDef (see page 5100)

54.3.118 Variants.VarToWideStrDef Function

Converts a variant's value to a WideString (see page 3476).

Pascal

```
function VarToWideStrDef(const V: Variant; const ADefault: WideString): WideString;
```

C++

```
BSTR VarToWideStrDef(const Variant V, const BSTR ADefault);
```

File

Variants

Description

VarToWideStrDef converts the data in the variant V to a WideString (see page 3476) and returns the result. If the variant has a

null value, VarToWideStrDef returns ADefault.

See Also

VarToStr (see page 5099)

VarToWideStr (see page 5100)

VarToStrDef (see page 5099)

54.3.119 Variants.VarType Function

Returns the type code of a specified variant.

Pascal

```
function VarType(const V: Variant): TVarType;
```

C++

```
TVarType VarType(const Variant V);
```

File

Variants

Description

VarType returns the type code of the given variant. The resulting value is either the type code for a custom Variant (see page 3430) type, or it is constructed from the constants declared in the System (see page 3165) unit.

The lower twelve bits of a variant type code (the bits defined by the varTypeMask (see page 3939) bit mask) define the type of the variant. The varArray (see page 3929) bit is set if the variant is an array of the given type. The varByRef (see page 3930) bit is set if the variant is a reference to a value of the given type as opposed to an actual value.

The following table describes the meaning of each of the variant type codes defined in the System (see page 3165) unit:

VarType	Contents of variant
varEmpty (see page 3933)	The variant is Unassigned (see page 5076).
varNull (see page 3936)	The variant is Null (see page 5075).
varSmallint (see page 3938)	16-bit signed integer (type Smallint in Delphi, short in C++).
varInteger (see page 3935)	32-bit signed integer (type Integer in Delphi, int in C++).
varSingle (see page 3937)	Single-precision floating-point value (type Single in Delphi, float in C++).
varDouble (see page 3933)	Double-precision floating-point value (type double).
varCurrency (see page 3931)	Currency (see page 3242) floating-point value (type Currency (see page 3242)).
varDate (see page 3932)	Date (see page 4151) and time value (type TDateTime (see page 3335)).
varOleStr (see page 3936)	Reference to a dynamically allocated UNICODE string.

varDispatch (see page 3932)	Reference to an Automation object (an IDispatch (see page 3498) interface pointer).
varError (see page 3934)	Operating system error code.
varBoolean (see page 3930)	16-bit boolean (type WordBool).
varVariant (see page 3941)	A variant.
varUnknown (see page 3940)	Reference to an unknown object (an IInterface (see page 3503) or IUnknown (see page 3828) interface pointer).
varShortInt (see page 3937)	8-bit signed integer (type ShortInt in Delphi or signed char in C++)
varByte (see page 3931)	A Byte
varWord (see page 3941)	unsigned 16-bit value (Word)
varLongWord (see page 3935)	unsigned 32-bit value (type LongWord in Delphi or unsigned long in C++)
varInt64 (see page 3934)	64-bit signed integer (Int64 in Delphi or __int64 in C++)
varStrArg (see page 3938)	COM-compatible string.
varString (see page 3938)	Reference to a dynamically allocated string (not COM compatible).

The value returned by VarType corresponds to the VType field of a TVarData (see page 3813) record.

The type of a variant can be changed using VarAsType (see page 5083).

See Also

TVarData

VarAsType (see page 5083)

VarTypeToDataType

VarTypeAsText (see page 5102)

54.3.120 Variants.VarTypeAsText Function

Returns variant type descriptive string.

Pascal

```
function VarTypeAsText(const AType: TVarType): string;
```

C++

```
UnicodeString VarTypeAsText(const TVarType AType);
```

File

Variants

Description

VarTypeAsText returns a short string that names the variant type indicated by AType.

See Also

TVarData

VarAsType (see page 5083)

VarTypeToDataType

VarType (see page 5101)

54.3.121 Variants.VarTypeIsValidArrayType Function

Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.

Pascal

```
function VarTypeIsValidArrayType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidArrayType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidArrayType indicates whether AVarType is the type code for a Variant (see page 3430) type that can represent the elements of a Variant (see page 3430) array. This function is only intended for use with built-in Variant (see page 3430) types. It returns false for custom Variant (see page 3430) types, although they can be used for the elements of Variant (see page 3430) arrays.

See Also

VarArrayCreate (see page 5077)

VarTypeIsValidElementType (see page 5103)

54.3.122 Variants.VarTypeIsValidElementType Function

Indicates whether a Variant (see page 3430) type code represents a type that can be used for the elements in a Variant (see page 3430) array.

Pascal

```
function VarTypeIsValidElementType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidElementType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidElementType indicates whether AVarType is the type code for a Variant (see page 3430) type that can represent the elements of a Variant (see page 3430) array. Unlike the VarTypeIsValidArrayType (see page 5103) function, this function returns true for custom Variant (see page 3430) types.

See Also

VarArrayCreate (see page 5077)

VarTypeIsValidArrayType (see page 5103)

54.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBooleanToStringRule (see page 5105)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5110).
	TNullCompareRule (see page 5105)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 5106)	This is record Variants.TVarCompareResult.
	TVariantRelationship (see page 5106)	TVariantRelationship describes the results of comparing two Variants (see page 4997).

54.4.1 Variants.TBooleanToStringRule Enumeration

TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5110).

Pascal

```
TBooleanToStringRule = (
  bsrAsIs,
  bsrLower,
  bsrUpper
);
```

C++

```
enum TBooleanToStringRule {
  bsrAsIs,
  bsrLower,
  bsrUpper
};
```

File

Variants

Description

TBooleanToStringRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 5112) and NullMagnitudeRule (see page 5113).

54.4.2 Variants.TNullCompareRule Enumeration

TNullCompareRule enumerates possible rules for comparing variant nulls with other values.

Pascal

```
TNullCompareRule = (
  ncrError,
  ncrStrict,
  ncrLoose
);
```

C++

```
enum TNullCompareRule {
  ncrError,
  ncrStrict,
  ncrLoose
};
```

File

Variants

Description

TNullCompareRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 5112) and NullMagnitudeRule (see page 5113).

54.4.3 Variants.TVarCompareResult Enumeration

Pascal

```
TVarCompareResult = (
  crLessThan,
  crEqual,
  crGreaterThan
);
```

C++

```
enum TVarCompareResult {
  crLessThan,
  crEqual,
  crGreaterThan
};
```

File

Variants

Description

This is record Variants.TVarCompareResult.

54.4.4 Variants.TVariantRelationship Enumeration

TVariantRelationship describes the results of comparing two Variants (see page 4997).

Pascal

```
TVariantRelationship = (
  vrEqual,
  vrLessThan,
  vrGreaterThan,
  vrNotEqual
);
```

C++

```
enum TVariantRelationship {
  vrEqual,
  vrLessThan,
```

```
    vrGreaterThan,
    vrNotEqual
};
```

File

Variants

Description

TVariantRelationship describes the relationship between two Variant (see page 3430) values. The following table lists the possible values:

Value	Meaning
vrEqual	A is equivalent to B. Note that two NULL Variants (see page 4997) are considered equal, and two empty Variants (see page 4997) are considered equal, but a NULL Variant (see page 3430) does not equal an empty Variant (see page 3430).
vrLessThan	A is less than B.
vrGreaterThan	A is larger than B.
vrNotEqual	A and B are not equal, but the values are such that they can't be ordered relative to each other. For example, this describes the relationship when either A or B is empty or NULL, and the other one is not.

See Also

VarCompareValue

54.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAnyProc (see page 5108)	This is type Variants.TAnyProc.
TCustomVariantTypeClass (see page 5108)	This is type Variants.TCustomVariantTypeClass.
TVarDataArray (see page 5109)	This is type Variants.TVarDataArray.
TVarDispProc (see page 5109)	This is type Variants.TVarDispProc.

54.5.1 Variants.TAnyProc Type

Pascal

```
TAnyProc = procedure (var V: TVarData);
```

C++

```
(var V: TVarData) ( TAnyProc)();
```

File

Variants

Description

This is type Variants.TAnyProc.

54.5.2 Variants.TCustomVariantTypeClass Type

Pascal

```
TCustomVariantTypeClass = class of TCustomVariantType;
```

C++

```
TCustomVariantType TCustomVariantTypeClass;
```

File

Variants

Description

This is type Variants.TCustomVariantTypeClass.

54.5.3 Variants.TVarDataArray Type

Pascal

```
TVarDataArray = array of TVarData;
```

C++

```
array of TVarData TVarDataArray;
```

File

Variants

Description

This is type Variants.TVarDataArray.

54.5.4 Variants.TVarDispProc Type

Pascal

```
TVarDispProc = procedure (Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params:
Pointer);
```

C++

```
(Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params: Pointer) (
TVarDispProc)();
```

File

Variants

Description

This is type Variants.TVarDispProc.

54.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanToStringRule (see page 5110)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 5111)	This is variable Variants.ChangeAnyProc.
ClearAnyProc (see page 5111)	This is variable Variants.ClearAnyProc.
EmptyParam (see page 5111)	Contains an OleVariant (see page 3281) that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 5112)	Specifies string value of Null (see page 5075) variants.
NullEqualityRule (see page 5112)	Specifies rule for Null (see page 5075) variant equality comparison.
NullMagnitudeRule (see page 5113)	Specifies rule for Null (see page 5075) variant magnitude comparison.
NullStrictConvert (see page 5114)	Specifies whether Null (see page 5075) conversion is an error.
OleVariantInt64AsDouble (see page 5114)	This is variable Variants.OleVariantInt64AsDouble.
PackVarCreation (see page 5115)	Specifies whether space-efficient Variant (see page 3430) Types (see page 4683) are preferred.
RefAnyProc (see page 5115)	This is variable Variants.RefAnyProc.
VarDispProc (see page 5115)	Provides the support for IDispatch (see page 3498)-based Automation.

54.6.1 Variants.BooleanToStringRule Variable

Specifies rule fixing up Boolean values converted to strings.

Pascal

```
BooleanToStringRule: TBooleanToStringRule = bsrAsIs;
```

C++

```
TBooleanToStringRule BooleanToStringRule = bsrAsIs;
```

File

Variants

Description

BooleanToStringRule determines the rule for fixing up the string generated by a conversion from Boolean to string.

Constant	Meaning
bsrAsIs	(Default.) The value obtained from the locale is left as is.
bsrUpper	The string is converted to all upper case.
bsrLower	The string is converted to all lower case.

54.6.2 Variants.ChangeAnyProc Variable

Pascal

```
ChangeAnyProc: TAnyProc;
```

C++

```
TAnyProc ChangeAnyProc;
```

File

Variants

Description

This is variable Variants.ChangeAnyProc.

54.6.3 Variants.ClearAnyProc Variable

Pascal

```
ClearAnyProc: TAnyProc;
```

C++

```
TAnyProc ClearAnyProc;
```

File

Variants

Description

This is variable Variants.ClearAnyProc.

54.6.4 Variants.EmptyParam Variable

Contains an OleVariant (see page 3281) that represents an unused optional parameter on a dual interface.

Pascal

```
EmptyParam: OleVariant;
```

C++

```
OleVariant EmptyParam;
```

File

Variants

Description

EmptyParam can be used for Variants (see page 4997) that represent optional parameters whose value is unassigned. This is useful when the code that marshals method calls requires a fixed number of parameters, even if some of them are optional.

When marshaling interface calls that include optional parameters, COM requires a value for those parameters even when they are not used. EmptyParam returns an OleVariant (see page 3281) you can pass as that value to indicate that the parameter is

not used.

Marshaling interfaces also exist under Linux, but we can't spare the resources to document them right now.

See Also

VarIsEmptyParam (see page 5092)

54.6.5 Variants.NullAsStringValue Variable

Specifies string value of Null (see page 5075) variants.

Pascal

```
NullAsStringValue: string = '';
```

C++

```
UnicodeString NullAsStringValue = '';
```

File

Variants

Description

NullAsStringValue specifies the string that is the result of a conversion from a Null (see page 5075) variant to a string type. By default, NullAsStringValue is the empty string.

NullAsStringValue is only used when the NullStrictConvert (see page 5114) global variable is set to false.

See Also

NullStrictConvert (see page 5114)

Null (see page 5075)

54.6.6 Variants.NullEqualityRule Variable

Specifies rule for Null (see page 5075) variant equality comparison.

Pascal

```
NullEqualityRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullEqualityRule = ncrLoose;
```

File

Variants

Description

NullEqualityRule determines the rule for handling equality comparisons (equals or not-equals) when one of the values compared is a variant containing a null value. NullEqualityRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is only equal to another variant containing a null.

ncrStrict	An equality comparison of a variant containing a null is always false, regardless of the other value.
ncrError	An equality comparison raises an EVariantInvalidNullOpError (see page 5006) if either value is a variant containing a null.

See Also

Null (see page 5075)

NullMagnitudeRule (see page 5113)

NullStrictConvert (see page 5114)

EVariantInvalidNullOpError (see page 5006)

54.6.7 Variants.NullMagnitudeRule Variable

Specifies rule for Null (see page 5075) variant magnitude comparison.

Pascal

```
NullMagnitudeRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullMagnitudeRule = ncrLoose;
```

File

Variants

Description

NullMagnitudeRule determines the rule for handling magnitude comparisons (one value is greater than, greater than or equals, less than, or less than or equals to another) when one of the values compared is a variant containing a null value. NullMagnitudeRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is less than any other value, but equal to another variant containing a null.
ncrStrict	A magnitude comparison of a variant containing a null is always false, regardless of the other value.
ncrError	A magnitude comparison raises an EVariantInvalidNullOpError (see page 5006) if either value is a variant containing a null.

See Also

Null (see page 5075)

NullMagnitudeRule

NullStrictConvert (see page 5114)

EVariantInvalidNullOpError (see page 5006)

54.6.8 Variants.NullStrictConvert Variable

Specifies whether Null (see page 5075) conversion is an error.

Pascal

```
NullStrictConvert: Boolean = True;
```

C++

```
Boolean NullStrictConvert = True;
```

File

Variants

Description

NullStrictConvert determines the outcome of attempts to convert Null (see page 5075) variants to other types. If NullStrictConvert is true (default), attempting to convert a Null (see page 5075) variant raises a EVariantTypeCastError (see page 5010), unless the conversion is to a custom variant that defines a conversion from Null (see page 5075). If NullStrictConvert is false, then conversion from Null (see page 5075) follows the following rules:

Destination	Conversion result
Numeric type	Zero
String type	Value of NullAsStringValue (see page 5112) global variable. (Default: empty string.)
Boolean	False.

See Also

TVarData

VarType (see page 5101)

NullAsStringValue (see page 5112)

EVariantTypeCastError (see page 5010)

54.6.9 Variants.OleVariantInt64AsDouble Variable

Pascal

```
OleVariantInt64AsDouble: Boolean = False;
```

C++

```
Boolean OleVariantInt64AsDouble = False;
```

File

Variants

Description

This is variable Variants.OleVariantInt64AsDouble.

54.6.10 Variants.PackVarCreation Variable

Specifies whether space-efficient Variant (see page 3430) Types (see page 4683) are preferred.

Pascal

```
PackVarCreation: Boolean = True;
```

C++

```
Boolean PackVarCreation = True;
```

File

Variants

Description

PackVarCreation controls the way Variant (see page 3430) Types (see page 4683) are chosen for newly-created Variants (see page 4997). If PackVarCreation is true (default), a new Variant (see page 3430) has the smallest possible Variant (see page 3430) Type. If PackVarCreation is false, a new Variant (see page 3430) has the Variant (see page 3430) Type that minimizes execution time.

See Also

TVarData

VarType (see page 5101)

54.6.11 Variants.RefAnyProc Variable

Pascal

```
RefAnyProc: TAnyProc;
```

C++

```
TAnyProc RefAnyProc;
```

File

Variants

Description

This is variable Variants.RefAnyProc.

54.6.12 Variants.VarDispProc Variable

Provides the support for IDispatch (see page 3498)-based Automation.

Pascal

```
VarDispProc: TVarDispProc;
```

C++

```
TVarDispProc VarDispProc;
```

File

Variants

Description

In Delphi, VarDispProc points to a routine that is called to execute methods (or access properties) of an interface that is assigned to a Variant (see page 3430). The Variants (see page 4997) unit assigns a procedure that generates error 222 or raises an EVariantDispatchError (see page 5005) exception. The ComObj unit reassigns this variable to a procedure that generates the appropriate COM IDispatch (see page 3498) calls.

To control how Variants (see page 4997) execute methods when they have a value that is an interface, assign your own procedure to VarDispProc.

The Linux implementation of IDispatch (see page 3498) is currently undocumented, due to resource constraints. When that changes, decolorize and revise this topic.

See Also

EVariantDispatchError (see page 5005)

55 VarUtils Namespace

This is namespace VarUtils.

Topics

Name	Description
Classes (see page 5120)	The following table lists classes in this documentation.
Functions (see page 5124)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5134)	The following table lists structs, records, enums in this documentation.
Variables (see page 5135)	The following table lists variables in this documentation.
Constants (see page 5143)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESafeArrayBoundsError (see page 5120)	This is class VarUtils.ESafeArrayBoundsError.
	ESafeArrayCreateError (see page 5120)	This is class VarUtils.ESafeArrayCreateError.
	ESafeArrayError (see page 5121)	This is class VarUtils.ESafeArrayError.
	ESafeArrayLockedError (see page 5123)	This is class VarUtils.ESafeArrayLockedError.

Constants

Name	Description
ARR_DISPATCH (see page 5143)	This is constant VarUtils.ARR_DISPATCH.
ARR_FIXEDSIZE (see page 5144)	This is constant VarUtils.ARR_FIXEDSIZE.
ARR_NONE (see page 5144)	This is constant VarUtils.ARR_NONE.
ARR_OLESTR (see page 5144)	This is constant VarUtils.ARR_OLESTR.
ARR_UNKNOWN (see page 5145)	This is constant VarUtils.ARR_UNKNOWN.
ARR_VARIANT (see page 5145)	This is constant VarUtils.ARR_VARIANT.
CMaxArrayVarType (see page 5145)	This is constant VarUtils.CMaxArrayVarType.
CMinArrayVarType (see page 5146)	This is constant VarUtils.CMinArrayVarType.
CVarTypeToElementInfo (see page 5146)	This is constant VarUtils.CVarTypeToElementInfo.
GenericOperations (see page 5147)	This is constant VarUtils.GenericOperations.
GenericSafeArrays (see page 5147)	This is constant VarUtils.GenericSafeArrays.
GenericSafeArrayUsesLibC (see page 5147)	This is constant VarUtils.GenericSafeArrayUsesLibC.
GenericVariants (see page 5148)	This is constant VarUtils.GenericVariants.
GenericVarUtils (see page 5148)	This is constant VarUtils.GenericVarUtils.
VAR_ARRAYISLOCKED (see page 5148)	This is constant VarUtils.VAR_ARRAYISLOCKED.
VAR_BADINDEX (see page 5149)	This is constant VarUtils.VAR_BADINDEX.

VAR_BADVARTYPE (see page 5149)	This is constant VarUtils.VAR_BADVARTYPE.
VAR_CMP_EQ (see page 5149)	This is constant VarUtils.VAR_CMP_EQ.
VAR_CMP_GT (see page 5150)	This is constant VarUtils.VAR_CMP_GT.
VAR_CMP_LT (see page 5150)	This is constant VarUtils.VAR_CMP_LT.
VAR_CMP_NULL (see page 5150)	This is constant VarUtils.VAR_CMP_NULL.
VAR_EXCEPTION (see page 5151)	This is constant VarUtils.VAR_EXCEPTION.
VAR_INVALIDARG (see page 5151)	This is constant VarUtils.VAR_INVALIDARG.
VAR_LOCALE_USER_DEFAULT (see page 5151)	This is constant VarUtils.VAR_LOCALE_USER_DEFAULT.
VAR_NOTIMPL (see page 5152)	This is constant VarUtils.VAR_NOTIMPL.
VAR_OK (see page 5152)	This is constant VarUtils.VAR_OK.
VAR_OUTOFMEMORY (see page 5152)	This is constant VarUtils.VAR_OUTOFMEMORY.
VAR_OVERFLOW (see page 5153)	This is constant VarUtils.VAR_OVERFLOW.
VAR_PARAMNOTFOUND (see page 5153)	This is constant VarUtils.VAR_PARAMNOTFOUND.
VAR_TYPEMISMATCH (see page 5153)	This is constant VarUtils.VAR_TYPEMISMATCH.
VAR_UNEXPECTED (see page 5154)	This is constant VarUtils.VAR_UNEXPECTED.

Functions

	Name	Description
	SafeArrayAccessData (see page 5124)	This is function VarUtils.SafeArrayAccessData.
	SafeArrayAllocData (see page 5125)	This is function VarUtils.SafeArrayAllocData.
	SafeArrayAllocDescriptor (see page 5125)	This is function VarUtils.SafeArrayAllocDescriptor.
	SafeArrayCheck (see page 5125)	This is function VarUtils.SafeArrayCheck.
	SafeArrayCopy (see page 5126)	This is function VarUtils.SafeArrayCopy.
	SafeArrayCopyData (see page 5126)	This is function VarUtils.SafeArrayCopyData.
	SafeArrayCreate (see page 5126)	This is function VarUtils.SafeArrayCreate.
	SafeArrayDestroy (see page 5127)	This is function VarUtils.SafeArrayDestroy.
	SafeArrayDestroyData (see page 5127)	This is function VarUtils.SafeArrayDestroyData.
	SafeArrayDestroyDescriptor (see page 5127)	This is function VarUtils.SafeArrayDestroyDescriptor.
	SafeArrayError (see page 5128)	This is function VarUtils.SafeArrayError.
	SafeArrayGetDim (see page 5128)	This is function VarUtils.SafeArrayGetDim.
	SafeArrayGetElement (see page 5128)	This is function VarUtils.SafeArrayGetElement.
	SafeArrayGetElemSize (see page 5129)	This is function VarUtils.SafeArrayGetElemSize.
	SafeArrayGetLBound (see page 5129)	This is function VarUtils.SafeArrayGetLBound.
	SafeArrayGetUBound (see page 5129)	This is function VarUtils.SafeArrayGetUBound.
	SafeArrayLock (see page 5130)	This is function VarUtils.SafeArrayLock.
	SafeArrayPtrOfIndex (see page 5130)	This is function VarUtils.SafeArrayPtrOfIndex.
	SafeArrayPutElement (see page 5130)	This is function VarUtils.SafeArrayPutElement.
	SafeArrayRedim (see page 5131)	This is function VarUtils.SafeArrayRedim.
	SafeArrayUnaccessData (see page 5131)	This is function VarUtils.SafeArrayUnaccessData.
	SafeArrayUnlock (see page 5131)	This is function VarUtils.SafeArrayUnlock.
	VarExceptionToResult (see page 5132)	This is function VarUtils.VarExceptionToResult.
	VariantChangeType (see page 5132)	This is function VarUtils.VariantChangeType.
	VariantClear (see page 5132)	This is function VarUtils.VariantClear.
	VariantCopy (see page 5133)	This is function VarUtils.VariantCopy.
	VariantCopyInd (see page 5133)	This is function VarUtils.VariantCopyInd.
	VariantInit (see page 5133)	This is function VarUtils.VariantInit.

Structs, Records, Enums

	Name	Description
	TVarTypeToElementInfo (see page 5134)	This is record VarUtils.TVarTypeToElementInfo.

Variables

Name	Description
VarAdd (see page 5135)	This is variable VarUtils.VarAdd.
VarAnd (see page 5136)	This is variable VarUtils.VarAnd.

VarBoolFromStr (see page 5136)	This is variable VarUtils.VarBoolFromStr.
VarBstrFromBool (see page 5136)	This is variable VarUtils.VarBstrFromBool.
VarBstrFromCy (see page 5137)	This is variable VarUtils.VarBstrFromCy.
VarBstrFromDate (see page 5137)	This is variable VarUtils.VarBstrFromDate.
VarCmp (see page 5137)	This is variable VarUtils.VarCmp.
VarCyFromStr (see page 5138)	This is variable VarUtils.VarCyFromStr.
VarDateFromStr (see page 5138)	This is variable VarUtils.VarDateFromStr.
VarDiv (see page 5138)	This is variable VarUtils.VarDiv.
VarI4FromStr (see page 5139)	This is variable VarUtils.VarI4FromStr.
VariantChangeTypeEx (see page 5139)	This is variable VarUtils.VariantChangeTypeEx.
VarIDiv (see page 5139)	This is variable VarUtils.VarIDiv.
VarMod (see page 5140)	This is variable VarUtils.VarMod.
VarMul (see page 5140)	This is variable VarUtils.VarMul.
VarNeg (see page 5140)	This is variable VarUtils.VarNeg.
VarNot (see page 5141)	This is variable VarUtils.VarNot.
VarOr (see page 5141)	This is variable VarUtils.VarOr.
VarR4FromStr (see page 5141)	This is variable VarUtils.VarR4FromStr.
VarR8FromStr (see page 5142)	This is variable VarUtils.VarR8FromStr.
VarSub (see page 5142)	This is variable VarUtils.VarSub.
VarXor (see page 5142)	This is variable VarUtils.VarXor.

55.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESafeArrayBoundsError (see page 5120)	This is class VarUtils.ESafeArrayBoundsError.
	ESafeArrayCreateError (see page 5120)	This is class VarUtils.ESafeArrayCreateError.
	ESafeArrayError (see page 5121)	This is class VarUtils.ESafeArrayError.
	ESafeArrayLockedError (see page 5123)	This is class VarUtils.ESafeArrayLockedError.

55.1.1 ESafeArrayBoundsError Class

Class Hierarchy

Pascal

```
ESafeArrayBoundsError = class(ESafeArrayError);
```

C++

```
class ESafeArrayBoundsError : public ESafeArrayError;
```

File

VarUtils

Description

This is class VarUtils.ESafeArrayBoundsError.

55.1.2 ESafeArrayCreateError Class

Class Hierarchy

Pascal

```
ESafeArrayCreateError = class(ESafeArrayError);
```

C++

```
class ESafeArrayCreateError : public ESafeArrayError;
```

File

VarUtils

Description

This is class VarUtils.ESafeArrayCreateError.

55.1.3 ESafeArrayError Class

Class Hierarchy

Pascal

```
ESafeArrayError = class(Exception);
```

C++

```
class ESafeArrayError : public Exception;
```

File

VarUtils

Description

This is class VarUtils.ESafeArrayError.

55.1.3.1 ESafeArrayError Members

The following tables list the members exposed by ESafeArrayError.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.

	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateHResult (see page 5122)	This is CreateHResult, a member of class ESafeArrayError.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 4026)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3391)	Returns a string containing the class name.
V	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	ErrorCode (see page 5123)	This is ErrorCode, a member of class ESafeArrayError.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

55.1.3.2 ESafeArrayError.CreateHResult Constructor

Pascal

```
constructor CreateHResult(AResult: HRESULT; const AMessage: string = '');
```

C++

```
__fastcall ESafeArrayError(HRESULT AResult, const UnicodeString AMessage = '');
```

Description

This is CreateHResult, a member of class ESafeArrayError.

55.1.3.3 ESafeArrayError Properties

The properties of the ESafeArrayError class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 5123)	This is ErrorCode, a member of class ESafeArrayError.

55.1.3.3.1 ESafeArrayError.ErrorCode Property

Pascal

```
property ErrorCode: HRESULT;
```

C++

```
__property HRESULT ErrorCode;
```

Description

This is ErrorCode, a member of class ESafeArrayError.

55.1.4 ESafeArrayLockedError Class

Class Hierarchy

Pascal

```
ESafeArrayLockedError = class(ESafeArrayError);
```

C++

```
class ESafeArrayLockedError : public ESafeArrayError;
```

File

VarUtils

Description

This is class VarUtils.ESafeArrayLockedError.

55.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SafeArrayAccessData (see page 5124)	This is function VarUtils.SafeArrayAccessData.
	SafeArrayAllocData (see page 5125)	This is function VarUtils.SafeArrayAllocData.
	SafeArrayAllocDescriptor (see page 5125)	This is function VarUtils.SafeArrayAllocDescriptor.
	SafeArrayCheck (see page 5125)	This is function VarUtils.SafeArrayCheck.
	SafeArrayCopy (see page 5126)	This is function VarUtils.SafeArrayCopy.
	SafeArrayCopyData (see page 5126)	This is function VarUtils.SafeArrayCopyData.
	SafeArrayCreate (see page 5126)	This is function VarUtils.SafeArrayCreate.
	SafeArrayDestroy (see page 5127)	This is function VarUtils.SafeArrayDestroy.
	SafeArrayDestroyData (see page 5127)	This is function VarUtils.SafeArrayDestroyData.
	SafeArrayDestroyDescriptor (see page 5127)	This is function VarUtils.SafeArrayDestroyDescriptor.
	SafeArrayError (see page 5128)	This is function VarUtils.SafeArrayError.
	SafeArrayGetDim (see page 5128)	This is function VarUtils.SafeArrayGetDim.
	SafeArrayGetElement (see page 5128)	This is function VarUtils.SafeArrayGetElement.
	SafeArrayGetElemSize (see page 5129)	This is function VarUtils.SafeArrayGetElemSize.
	SafeArrayGetLBound (see page 5129)	This is function VarUtils.SafeArrayGetLBound.
	SafeArrayGetUBound (see page 5129)	This is function VarUtils.SafeArrayGetUBound.
	SafeArrayLock (see page 5130)	This is function VarUtils.SafeArrayLock.
	SafeArrayPtrOfIndex (see page 5130)	This is function VarUtils.SafeArrayPtrOfIndex.
	SafeArrayPutElement (see page 5130)	This is function VarUtils.SafeArrayPutElement.
	SafeArrayRedim (see page 5131)	This is function VarUtils.SafeArrayRedim.
	SafeArrayUnaccessData (see page 5131)	This is function VarUtils.SafeArrayUnaccessData.
	SafeArrayUnlock (see page 5131)	This is function VarUtils.SafeArrayUnlock.
	VarExceptionToResult (see page 5132)	This is function VarUtils.VarExceptionToResult.
	VariantChangeType (see page 5132)	This is function VarUtils.VariantChangeType.
	VariantClear (see page 5132)	This is function VarUtils.VariantClear.
	VariantCopy (see page 5133)	This is function VarUtils.VariantCopy.
	VariantCopyInd (see page 5133)	This is function VarUtils.VariantCopyInd.
	VariantInit (see page 5133)	This is function VarUtils.VariantInit.

55.2.1 VarUtils.SafeArrayAccessData Function

Pascal

```
function SafeArrayAccessData(VarArray: PVarArray; out Data: Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAccessData(PVarArray VarArray, void * Data);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayAccessData.

55.2.2 VarUtils.SafeArrayAllocData Function

Pascal

```
function SafeArrayAllocData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAllocData(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayAllocData.

55.2.3 VarUtils.SafeArrayAllocDescriptor Function

Pascal

```
function SafeArrayAllocDescriptor(DimCount: Integer; out VarArray: PVarArray): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAllocDescriptor(int DimCount, PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayAllocDescriptor.

55.2.4 VarUtils.SafeArrayCheck Function

Pascal

```
procedure SafeArrayCheck(AResult: HRESULT);
```

C++

```
SafeArrayCheck(HRESULT AResult);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayCheck.

55.2.5 VarUtils.SafeArrayCopy Function

Pascal

```
function SafeArrayCopy(SourceArray: PVarArray; out TargetArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayCopy(PVarArray SourceArray, PVarArray TargetArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayCopy.

55.2.6 VarUtils.SafeArrayCopyData Function

Pascal

```
function SafeArrayCopyData(SourceArray: PVarArray; TargetArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayCopyData(PVarArray SourceArray, PVarArray TargetArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayCopyData.

55.2.7 VarUtils.SafeArrayCreate Function

Pascal

```
function SafeArrayCreate(VarType: Integer; DimCount: Integer; const Bounds:
TVarArrayBoundArray): PVarArray; stdcall;
```

C++

```
__stdcall PVarArray SafeArrayCreate(int VarType, int DimCount, const TVarArrayBoundArray
Bounds);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayCreate.

55.2.8 VarUtils.SafeArrayDestroy Function

Pascal

```
function SafeArrayDestroy(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroy(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayDestroy.

55.2.9 VarUtils.SafeArrayDestroyData Function

Pascal

```
function SafeArrayDestroyData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroyData(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayDestroyData.

55.2.10 VarUtils.SafeArrayDestroyDescriptor Function

Pascal

```
function SafeArrayDestroyDescriptor(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroyDescriptor(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayDestroyDescriptor.

55.2.11 VarUtils.SafeArrayError Function

Pascal

```
procedure SafeArrayError(AResult: HRESULT);
```

C++

```
SafeArrayError(HRESULT AResult);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayError.

55.2.12 VarUtils.SafeArrayGetDim Function

Pascal

```
function SafeArrayGetDim(VarArray: PVarArray): Integer; stdcall;
```

C++

```
__stdcall int SafeArrayGetDim(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayGetDim.

55.2.13 VarUtils.SafeArrayGetElement Function

Pascal

```
function SafeArrayGetElement(VarArray: PVarArray; Indices: PVarArrayCoorArray; Data: Pointer):
HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetElement(PVarArray VarArray, PVarArrayCoorArray Indices, void *
Data);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayGetElement.

55.2.14 VarUtils.SafeArrayGetElemSize Function

Pascal

```
function SafeArrayGetElemSize(VarArray: PVarArray): LongWord; stdcall;
```

C++

```
__stdcall LongWord SafeArrayGetElemSize(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayGetElemSize.

55.2.15 VarUtils.SafeArrayGetLBound Function

Pascal

```
function SafeArrayGetLBound(VarArray: PVarArray; Dim: Integer; out LBound: Integer): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetLBound(PVarArray VarArray, int Dim, int LBound);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayGetLBound.

55.2.16 VarUtils.SafeArrayGetUBound Function

Pascal

```
function SafeArrayGetUBound(VarArray: PVarArray; Dim: Integer; out UBound: Integer): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetUBound(PVarArray VarArray, int Dim, int UBound);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayGetUBound.

55.2.17 VarUtils.SafeArrayLock Function

Pascal

```
function SafeArrayLock(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayLock(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayLock.

55.2.18 VarUtils.SafeArrayPtrOfIndex Function

Pascal

```
function SafeArrayPtrOfIndex(VarArray: PVarArray; Indices: PVarArrayCoorArray; var Address:
Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayPtrOfIndex(PVarArray VarArray, PVarArrayCoorArray Indices, void *
Address);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayPtrOfIndex.

55.2.19 VarUtils.SafeArrayPutElement Function

Pascal

```
function SafeArrayPutElement(VarArray: PVarArray; Indices: PVarArrayCoorArray; const Data:
Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayPutElement(PVarArray VarArray, PVarArrayCoorArray Indices, const
void * Data);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayPutElement.

55.2.20 VarUtils.SafeArrayRedim Function

Pascal

```
function SafeArrayRedim(VarArray: PVarArray; const NewBound: TVarArrayBound): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayRedim(PVarArray VarArray, const TVarArrayBound NewBound);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayRedim.

55.2.21 VarUtils.SafeArrayUnaccessData Function

Pascal

```
function SafeArrayUnaccessData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayUnaccessData(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayUnaccessData.

55.2.22 VarUtils.SafeArrayUnlock Function

Pascal

```
function SafeArrayUnlock(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayUnlock(PVarArray VarArray);
```

File

VarUtils

Description

This is function VarUtils.SafeArrayUnlock.

55.2.23 VarUtils.VarExceptionToResult Function

Pascal

```
function VarExceptionToResult(const E: Exception): HRESULT;
```

C++

```
HRESULT VarExceptionToResult(const Exception E);
```

File

VarUtils

Description

This is function VarUtils.VarExceptionToResult.

55.2.24 VarUtils.VariantChangeType Function

Pascal

```
function VariantChangeType(var Dest: TVarData; const Source: TVarData; wFlags: Word; VarType:
Word): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantChangeType(TVarData Dest, const TVarData Source, Word wFlags, Word
VarType);
```

File

VarUtils

Description

This is function VarUtils.VariantChangeType.

55.2.25 VarUtils.VariantClear Function

Pascal

```
function VariantClear(var V: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantClear(TVarData V);
```

File

VarUtils

Description

This is function VarUtils.VariantClear.

55.2.26 VarUtils.VariantCopy Function

Pascal

```
function VariantCopy(var Dest: TVarData; const Source: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantCopy(TVarData Dest, const TVarData Source);
```

File

VarUtils

Description

This is function VarUtils.VariantCopy.

55.2.27 VarUtils.VariantCopyInd Function

Pascal

```
function VariantCopyInd(var Dest: TVarData; const Source: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantCopyInd(TVarData Dest, const TVarData Source);
```

File

VarUtils

Description

This is function VarUtils.VariantCopyInd.

55.2.28 VarUtils.VariantInit Function

Pascal

```
procedure VariantInit(var V: TVarData); stdcall;
```

C++

```
__stdcall VariantInit(TVarData V);
```

File

VarUtils

Description

This is function VarUtils.VariantInit.

55.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TVarTypeToElementInfo (see page 5134)	This is record VarUtils.TVarTypeToElementInfo.

55.3.1 VarUtils.TVarTypeToElementInfo Record

Pascal

```
TVarTypeToElementInfo = record
  ValidBase: Boolean;
  ValidElement: Boolean;
  Size: Integer;
  Flags: Word;
end;
```

C++

```
struct TVarTypeToElementInfo {
  Boolean ValidBase;
  Boolean ValidElement;
  int Size;
  Word Flags;
};
```

File

VarUtils

Description

This is record VarUtils.TVarTypeToElementInfo.

55.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
VarAdd (see page 5135)	This is variable VarUtils.VarAdd.
VarAnd (see page 5136)	This is variable VarUtils.VarAnd.
VarBoolFromStr (see page 5136)	This is variable VarUtils.VarBoolFromStr.
VarBstrFromBool (see page 5136)	This is variable VarUtils.VarBstrFromBool.
VarBstrFromCy (see page 5137)	This is variable VarUtils.VarBstrFromCy.
VarBstrFromDate (see page 5137)	This is variable VarUtils.VarBstrFromDate.
VarCmp (see page 5137)	This is variable VarUtils.VarCmp.
VarCyFromStr (see page 5138)	This is variable VarUtils.VarCyFromStr.
VarDateFromStr (see page 5138)	This is variable VarUtils.VarDateFromStr.
VarDiv (see page 5138)	This is variable VarUtils.VarDiv.
VarI4FromStr (see page 5139)	This is variable VarUtils.VarI4FromStr.
VariantChangeTypeEx (see page 5139)	This is variable VarUtils.VariantChangeTypeEx.
VarIDiv (see page 5139)	This is variable VarUtils.VarIDiv.
VarMod (see page 5140)	This is variable VarUtils.VarMod.
VarMul (see page 5140)	This is variable VarUtils.VarMul.
VarNeg (see page 5140)	This is variable VarUtils.VarNeg.
VarNot (see page 5141)	This is variable VarUtils.VarNot.
VarOr (see page 5141)	This is variable VarUtils.VarOr.
VarR4FromStr (see page 5141)	This is variable VarUtils.VarR4FromStr.
VarR8FromStr (see page 5142)	This is variable VarUtils.VarR8FromStr.
VarSub (see page 5142)	This is variable VarUtils.VarSub.
VarXor (see page 5142)	This is variable VarUtils.VarXor.

55.4.1 VarUtils.VarAdd Variable

Pascal

```
VarAdd: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarAdd;
```

File

VarUtils

Description

This is variable VarUtils.VarAdd.

55.4.2 VarUtils.VarAnd Variable

Pascal

```
VarAnd: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarAnd;
```

File

VarUtils

Description

This is variable VarUtils.VarAnd.

55.4.3 VarUtils.VarBoolFromStr Variable

Pascal

```
VarBoolFromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out
boolOut: WordBool): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out boolOut: WordBool):
HRESULT VarBoolFromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarBoolFromStr.

55.4.4 VarUtils.VarBstrFromBool Variable

Pascal

```
VarBstrFromBool: function(boolIn: WordBool; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(boolIn: WordBool; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromBool;
```

File

VarUtils

Description

This is variable VarUtils.VarBstrFromBool.

55.4.5 VarUtils.VarBstrFromCy Variable

Pascal

```
VarBstrFromCy: function(cyIn: Currency; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(cyIn: Currency; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromCy;
```

File

VarUtils

Description

This is variable VarUtils.VarBstrFromCy.

55.4.6 VarUtils.VarBstrFromDate Variable

Pascal

```
VarBstrFromDate: function(dateIn: TDateTime; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(dateIn: TDateTime; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromDate;
```

File

VarUtils

Description

This is variable VarUtils.VarBstrFromDate.

55.4.7 VarUtils.VarCmp Variable

Pascal

```
VarCmp: function(const Left, Right: TVarData; LCID: Integer; Flags: LongWord): HRESULT;
```

C++

```
function(const Left, Right: TVarData; LCID: Integer; Flags: LongWord): HRESULT VarCmp;
```

File

VarUtils

Description

This is variable VarUtils.VarCmp.

55.4.8 VarUtils.VarCyFromStr Variable

Pascal

```
VarCyFromStr: function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out cyOut:
Currency): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out cyOut: Currency): HRESULT
VarCyFromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarCyFromStr.

55.4.9 VarUtils.VarDateFromStr Variable

Pascal

```
VarDateFromStr: function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out dateOut:
TDateTime): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out dateOut: TDateTime):
HRESULT VarDateFromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarDateFromStr.

55.4.10 VarUtils.VarDiv Variable

Pascal

```
VarDiv: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarDiv;
```

File

VarUtils

Description

This is variable VarUtils.VarDiv.

55.4.11 VarUtils.VarI4FromStr Variable

Pascal

```
VarI4FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out lOut:
Longint): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out lOut: Longint): HRESULT
VarI4FromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarI4FromStr.

55.4.12 VarUtils.VariantChangeTypeEx Variable

Pascal

```
VariantChangeTypeEx: function(var Dest: TVarData; const Source: TVarData; LCID: Integer;
wFlags: Word; VarType: Word): HRESULT;
```

C++

```
function(var Dest: TVarData; const Source: TVarData; LCID: Integer; wFlags: Word; VarType:
Word): HRESULT VariantChangeTypeEx;
```

File

VarUtils

Description

This is variable VarUtils.VariantChangeTypeEx.

55.4.13 VarUtils.VarIDiv Variable

Pascal

```
VarIDiv: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarIDiv;
```

File

VarUtils

Description

This is variable VarUtils.VarIDiv.

55.4.14 VarUtils.VarMod Variable

Pascal

```
VarMod: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarMod;
```

File

VarUtils

Description

This is variable VarUtils.VarMod.

55.4.15 VarUtils.VarMul Variable

Pascal

```
VarMul: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarMul;
```

File

VarUtils

Description

This is variable VarUtils.VarMul.

55.4.16 VarUtils.VarNeg Variable

Pascal

```
VarNeg: function(const Source: TVarData; var Dest: TVarData): HRESULT;
```

C++

```
function(const Source: TVarData; var Dest: TVarData): HRESULT VarNeg;
```

File

VarUtils

Description

This is variable VarUtils.VarNeg.

55.4.17 VarUtils.VarNot Variable

Pascal

```
VarNot: function(const Source: TVarData; var Dest: TVarData): HRESULT;
```

C++

```
function(const Source: TVarData; var Dest: TVarData): HRESULT VarNot;
```

File

VarUtils

Description

This is variable VarUtils.VarNot.

55.4.18 VarUtils.VarOr Variable

Pascal

```
VarOr: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarOr;
```

File

VarUtils

Description

This is variable VarUtils.VarOr.

55.4.19 VarUtils.VarR4FromStr Variable

Pascal

```
VarR4FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out fltOut:
Single): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out fltOut: Single):
HRESULT VarR4FromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarR4FromStr.

55.4.20 VarUtils.VarR8FromStr Variable

Pascal

```
VarR8FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out dblOut:
Double): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out dblOut: Double):
HRESULT VarR8FromStr;
```

File

VarUtils

Description

This is variable VarUtils.VarR8FromStr.

55.4.21 VarUtils.VarSub Variable

Pascal

```
VarSub: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarSub;
```

File

VarUtils

Description

This is variable VarUtils.VarSub.

55.4.22 VarUtils.VarXor Variable

Pascal

```
VarXor: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarXor;
```

File

VarUtils

Description

This is variable VarUtils.VarXor.

55.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ARR_DISPATCH (see page 5143)	This is constant VarUtils.ARR_DISPATCH.
ARR_FIXEDSIZE (see page 5144)	This is constant VarUtils.ARR_FIXEDSIZE.
ARR_NONE (see page 5144)	This is constant VarUtils.ARR_NONE.
ARR_OLESTR (see page 5144)	This is constant VarUtils.ARR_OLESTR.
ARR_UNKNOWN (see page 5145)	This is constant VarUtils.ARR_UNKNOWN.
ARR_VARIANT (see page 5145)	This is constant VarUtils.ARR_VARIANT.
CMaxArrayVarType (see page 5145)	This is constant VarUtils.CMaxArrayVarType.
CMinArrayVarType (see page 5146)	This is constant VarUtils.CMinArrayVarType.
CVarTypeToElementInfo (see page 5146)	This is constant VarUtils.CVarTypeToElementInfo.
GenericOperations (see page 5147)	This is constant VarUtils.GenericOperations.
GenericSafeArrays (see page 5147)	This is constant VarUtils.GenericSafeArrays.
GenericSafeArrayUsesLibC (see page 5147)	This is constant VarUtils.GenericSafeArrayUsesLibC.
GenericVariants (see page 5148)	This is constant VarUtils.GenericVariants.
GenericVarUtils (see page 5148)	This is constant VarUtils.GenericVarUtils.
VAR_ARRAYISLOCKED (see page 5148)	This is constant VarUtils.VAR_ARRAYISLOCKED.
VAR_BADINDEX (see page 5149)	This is constant VarUtils.VAR_BADINDEX.
VAR_BADVARTYPE (see page 5149)	This is constant VarUtils.VAR_BADVARTYPE.
VAR_CMP_EQ (see page 5149)	This is constant VarUtils.VAR_CMP_EQ.
VAR_CMP_GT (see page 5150)	This is constant VarUtils.VAR_CMP_GT.
VAR_CMP_LT (see page 5150)	This is constant VarUtils.VAR_CMP_LT.
VAR_CMP_NULL (see page 5150)	This is constant VarUtils.VAR_CMP_NULL.
VAR_EXCEPTION (see page 5151)	This is constant VarUtils.VAR_EXCEPTION.
VAR_INVALIDARG (see page 5151)	This is constant VarUtils.VAR_INVALIDARG.
VAR_LOCALE_USER_DEFAULT (see page 5151)	This is constant VarUtils.VAR_LOCALE_USER_DEFAULT.
VAR_NOTIMPL (see page 5152)	This is constant VarUtils.VAR_NOTIMPL.
VAR_OK (see page 5152)	This is constant VarUtils.VAR_OK.
VAR_OUTOFMEMORY (see page 5152)	This is constant VarUtils.VAR_OUTOFMEMORY.
VAR_OVERFLOW (see page 5153)	This is constant VarUtils.VAR_OVERFLOW.
VAR_PARAMNOTFOUND (see page 5153)	This is constant VarUtils.VAR_PARAMNOTFOUND.
VAR_TYPEMISMATCH (see page 5153)	This is constant VarUtils.VAR_TYPEMISMATCH.
VAR_UNEXPECTED (see page 5154)	This is constant VarUtils.VAR_UNEXPECTED.

55.5.1 VarUtils.ARR_DISPATCH Constant

Pascal

```
ARR_DISPATCH = $0400;
```

C++

```
ARR_DISPATCH = 0x0400;
```

File

VarUtils

Description

This is constant VarUtils.ARR_DISPATCH.

55.5.2 VarUtils.ARR_FIXEDSIZE Constant

Pascal

```
ARR_FIXEDSIZE = $0010;
```

C++

```
ARR_FIXEDSIZE = 0x0010;
```

File

VarUtils

Description

This is constant VarUtils.ARR_FIXEDSIZE.

55.5.3 VarUtils.ARR_NONE Constant

Pascal

```
ARR_NONE = $0000;
```

C++

```
ARR_NONE = 0x0000;
```

File

VarUtils

Description

This is constant VarUtils.ARR_NONE.

55.5.4 VarUtils.ARR_OLESTR Constant

Pascal

```
ARR_OLESTR = $0100;
```

C++

```
ARR_OLESTR = 0x0100;
```

File

VarUtils

Description

This is constant VarUtils.ARR_OLESTR.

55.5.5 VarUtils.ARR_UNKNOWN Constant

Pascal

```
ARR_UNKNOWN = $0200;
```

C++

```
ARR_UNKNOWN = 0x0200;
```

File

VarUtils

Description

This is constant VarUtils.ARR_UNKNOWN.

55.5.6 VarUtils.ARR_VARIANT Constant

Pascal

```
ARR_VARIANT = $0800;
```

C++

```
ARR_VARIANT = 0x0800;
```

File

VarUtils

Description

This is constant VarUtils.ARR_VARIANT.

55.5.7 VarUtils.CMaxArrayVarType Constant

Pascal

```
CMaxArrayVarType = varUInt64;
```

C++

```
CMaxArrayVarType = varUInt64;
```

File

VarUtils

Description

This is constant VarUtils.CMaxArrayVarType.

55.5.8 VarUtils.CMinArrayVarType Constant

Pascal

```
CMinArrayVarType = varEmpty;
```

C++

```
CMinArrayVarType = varEmpty;
```

File

VarUtils

Description

This is constant VarUtils.CMinArrayVarType.

55.5.9 VarUtils.CVarTypeToElementInfo Constant

Pascal

```
CVarTypeToElementInfo: array [CMinArrayVarType..CMaxArrayVarType] of TVarTypeToElementInfo = (
(ValidBase: False; ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: False;
ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: True; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 4; Flags: ARR_OLESTR), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_DISPATCH), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 16; Flags: ARR_VARIANT), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_UNKNOWN), (ValidBase: False; ValidElement: False; Size: 14; Flags: ARR_NONE),
(ValidBase: False; ValidElement: False; Size: 0; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 1; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 1;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: False; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: False; ValidElement: False; Size: 8; Flags: ARR_NONE));
```

C++

```
array [CMinArrayVarType..CMaxArrayVarType] of TVarTypeToElementInfo CVarTypeToElementInfo = (
(ValidBase: False; ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: False;
ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: True; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 4; Flags: ARR_OLESTR), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_DISPATCH), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 16; Flags: ARR_VARIANT), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_UNKNOWN), (ValidBase: False; ValidElement: False; Size: 14; Flags: ARR_NONE),
(ValidBase: False; ValidElement: False; Size: 0; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 1; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 1;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: False; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: False; ValidElement: False; Size: 8; Flags: ARR_NONE));
```

File

VarUtils

Description

This is constant VarUtils.CVarTypeToElementInfo.

55.5.10 VarUtils.GenericOperations Constant

Pascal

```
GenericOperations = GenericVariants;
```

C++

```
GenericOperations = GenericVariants;
```

File

VarUtils

Description

This is constant VarUtils.GenericOperations.

55.5.11 VarUtils.GenericSafeArrays Constant

Pascal

```
GenericSafeArrays = GenericVarUtils;
```

C++

```
GenericSafeArrays = GenericVarUtils;
```

File

VarUtils

Description

This is constant VarUtils.GenericSafeArrays.

55.5.12 VarUtils.GenericSafeArrayUsesLibC Constant

Pascal

```
GenericSafeArrayUsesLibC = FALSE;
```

C++

```
GenericSafeArrayUsesLibC = FALSE;
```

File

VarUtils

Description

This is constant VarUtils.GenericSafeArrayUsesLibC.

55.5.13 VarUtils.GenericVariants Constant

Pascal

```
GenericVariants = GenericVarUtils;
```

C++

```
GenericVariants = GenericVarUtils;
```

File

VarUtils

Description

This is constant VarUtils.GenericVariants.

55.5.14 VarUtils.GenericVarUtils Constant

Pascal

```
GenericVarUtils = False;
```

C++

```
GenericVarUtils = False;
```

File

VarUtils

Description

This is constant VarUtils.GenericVarUtils.

55.5.15 VarUtils.VAR_ARRAYISLOCKED Constant

Pascal

```
VAR_ARRAYISLOCKED = HRESULT($8002000D);
```

C++

```
VAR_ARRAYISLOCKED = HRESULT($8002000D);
```

File

VarUtils

Description

This is constant VarUtils.VAR_ARRAYISLOCKED.

55.5.16 VarUtils.VAR_BADINDEX Constant

Pascal

```
VAR_BADINDEX = HRESULT($8002000B);
```

C++

```
VAR_BADINDEX = HRESULT($8002000B);
```

File

VarUtils

Description

This is constant VarUtils.VAR_BADINDEX.

55.5.17 VarUtils.VAR_BADVARTYPE Constant

Pascal

```
VAR_BADVARTYPE = HRESULT($80020008);
```

C++

```
VAR_BADVARTYPE = HRESULT($80020008);
```

File

VarUtils

Description

This is constant VarUtils.VAR_BADVARTYPE.

55.5.18 VarUtils.VAR_CMP_EQ Constant

Pascal

```
VAR_CMP_EQ = 1;
```

C++

```
VAR_CMP_EQ = 1;
```

File

VarUtils

Description

This is constant VarUtils.VAR_CMP_EQ.

55.5.19 VarUtils.VAR_CMP_GT Constant

Pascal

```
VAR_CMP_GT = 2;
```

C++

```
VAR_CMP_GT = 2;
```

File

VarUtils

Description

This is constant VarUtils.VAR_CMP_GT.

55.5.20 VarUtils.VAR_CMP_LT Constant

Pascal

```
VAR_CMP_LT = 0;
```

C++

```
VAR_CMP_LT = 0;
```

File

VarUtils

Description

This is constant VarUtils.VAR_CMP_LT.

55.5.21 VarUtils.VAR_CMP_NULL Constant

Pascal

```
VAR_CMP_NULL = 3;
```

C++

```
VAR_CMP_NULL = 3;
```

File

VarUtils

Description

This is constant VarUtils.VAR_CMP_NULL.

55.5.22 VarUtils.VAR_EXCEPTION Constant

Pascal

```
VAR_EXCEPTION = HRESULT($80020009);
```

C++

```
VAR_EXCEPTION = HRESULT($80020009);
```

File

VarUtils

Description

This is constant VarUtils.VAR_EXCEPTION.

55.5.23 VarUtils.VAR_INVALIDARG Constant

Pascal

```
VAR_INVALIDARG = HRESULT($80070057);
```

C++

```
VAR_INVALIDARG = HRESULT($80070057);
```

File

VarUtils

Description

This is constant VarUtils.VAR_INVALIDARG.

55.5.24 VarUtils.VAR_LOCALE_USER_DEFAULT Constant

Pascal

```
VAR_LOCALE_USER_DEFAULT = $400;
```

C++

```
VAR_LOCALE_USER_DEFAULT = 0x400;
```

File

VarUtils

Description

This is constant VarUtils.VAR_LOCALE_USER_DEFAULT.

55.5.25 VarUtils.VAR_NOTIMPL Constant

Pascal

```
VAR_NOTIMPL = HRESULT($80004001);
```

C++

```
VAR_NOTIMPL = HRESULT($80004001);
```

File

VarUtils

Description

This is constant VarUtils.VAR_NOTIMPL.

55.5.26 VarUtils.VAR_OK Constant

Pascal

```
VAR_OK = HRESULT($00000000);
```

C++

```
VAR_OK = HRESULT($00000000);
```

File

VarUtils

Description

This is constant VarUtils.VAR_OK.

55.5.27 VarUtils.VAR_OUTOFMEMORY Constant

Pascal

```
VAR_OUTOFMEMORY = HRESULT($8007000E);
```

C++

```
VAR_OUTOFMEMORY = HRESULT($8007000E);
```

File

VarUtils

Description

This is constant VarUtils.VAR_OUTOFMEMORY.

55.5.28 VarUtils.VAR_OVERFLOW Constant

Pascal

```
VAR_OVERFLOW = HRESULT($8002000A);
```

C++

```
VAR_OVERFLOW = HRESULT($8002000A);
```

File

VarUtils

Description

This is constant VarUtils.VAR_OVERFLOW.

55.5.29 VarUtils.VAR_PARAMNOTFOUND Constant

Pascal

```
VAR_PARAMNOTFOUND = HRESULT($80020004);
```

C++

```
VAR_PARAMNOTFOUND = HRESULT($80020004);
```

File

VarUtils

Description

This is constant VarUtils.VAR_PARAMNOTFOUND.

55.5.30 VarUtils.VAR_TYPEMISMATCH Constant

Pascal

```
VAR_TYPEMISMATCH = HRESULT($80020005);
```

C++

```
VAR_TYPEMISMATCH = HRESULT($80020005);
```

File

VarUtils

Description

This is constant VarUtils.VAR_TYPEMISMATCH.

55.5.31 VarUtils.VAR_UNEXPECTED Constant

Pascal

```
VAR_UNEXPECTED = HRESULT($8000FFFF);
```

C++

```
VAR_UNEXPECTED = HRESULT($8000FFFF);
```

File

VarUtils

Description

This is constant VarUtils.VAR_UNEXPECTED.

56 VCLCom Namespace

This is namespace VCLCom.

Topics

Name	Description
Classes (see page 5156)	The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 5156)	TComponentFactory is the class factory for creating instances of components that support interfaces.

56.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 5156)	TComponentFactory is the class factory for creating instances of components that support interfaces.

56.1.1 TComponentFactory Class

TComponentFactory is the class factory for creating instances of components that support interfaces.

Class Hierarchy

Pascal

```
TComponentFactory = class(TAutoObjectFactory, IClassFactory);
```

C++

```
class TComponentFactory : public TAutoObjectFactory, public IClassFactory;
```

File

VCLCom

Description

TComponentFactory creates instances of component classes that implement interfaces. TComponentFactory supports the IClassFactory interface and inherits support for the IClassFactory2 interface.

Note: These components are actually created as contained objects of an automation object (aggregate).

56.1.1.1 TComponentFactory Members

The following tables list the members exposed by TComponentFactory.

Protected Methods

	Name	Description
	CreateInstance (see page 5158)	This is CreateInstance, a member of class TComponentFactory.

Public Methods

	Name	Description
	Create (see page 5157)	Instantiates an instance of a TComponentFactory (see page 5156) object.
	CreateComObject (see page 5157)	Instantiates a TComObject object.
	UpdateRegistry (see page 5158)	Registers the typed COM object class.

56.1.1.2 TComponentFactory.Create Constructor

Instantiates an instance of a TComponentFactory (see page 5156) object.

Pascal

```
constructor Create(ComServer: TComServerObject; ComponentClass: TComponentClass; const
ClassID: TGUID; Instancing: TClassInstancing; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TComponentFactory(TComServerObject ComServer, TComponentClass ComponentClass, const
TGUID ClassID, TClassInstancing Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

Create allocates memory for a TComponentFactory (see page 5156) object, and calls the inherited Create, passing its parameters in the same order. The ComponentClass parameter is passed cast as a TAutoClass and passed to the AutoClass parameter.

See Also

Create

56.1.1.3 TComponentFactory Methods

The methods of the TComponentFactory class are listed here.

Protected Methods

	Name	Description
	CreateInstance (see page 5158)	This is CreateInstance, a member of class TComponentFactory.

Public Methods

	Name	Description
	CreateComObject (see page 5157)	Instantiates a TComObject object.
	UpdateRegistry (see page 5158)	Registers the typed COM object class.

56.1.1.3.1 TComponentFactory.CreateComObject Method

Instantiates a TComObject object.

Pascal

```
function CreateComObject(const Controller: IUnknown): TComObject; override;
```

C++

```
virtual __fastcall TComObject CreateComObject(const IUnknown Controller);
```

Description

CreateComObject is the method that instantiates the class type for which TComObjectFactory is the class factory. CreateComObject calls the CreateFromFactory constructor of the TComObject class to create a COM object. CreateComObject passes itself as the Factory parameter to CreateFromFactory.

CreateComClass creates a COM object based upon the value of the ComClass property. ComClass is the class identifier (CLSID) used by other applications to identify the COM class for which they request an object instance.

See Also

comclass

CreateFromFactory

TComObject

56.1.1.3.2 TComponentFactory.CreateInstance Method

Pascal

```
function CreateInstance(const UnkOuter: IUnknown; const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const IUnknown UnkOuter, const TGUID IID,  Obj);
```

Description

This is CreateInstance, a member of class TComponentFactory.

56.1.1.3.3 TComponentFactory.UpdateRegistry Method

Registers the typed COM object class.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject

57 WebAdapt Namespace

This is namespace WebAdapt.

Topics

Name	Description
Classes (see page 5164)	The following table lists classes in this documentation.
Interfaces (see page 5860)	The following table lists interfaces in this documentation.
Functions (see page 5883)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5887)	The following table lists structs, records, enums in this documentation.
Types (see page 5889)	The following table lists types in this documentation.
Variables (see page 5906)	The following table lists variables in this documentation.
Constants (see page 5907)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterException (see page 5166)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 5167)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 5169)	tabstractfieldvaluesadapter is the base class for adapters that have a single default field.
	TAdapter (see page 5179)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 5190)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 5198)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 5203)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5427) component.
	TAdapterApplicationTitleField (see page 5207)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 5212)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 5220)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 5226)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 5233)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 5237)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 5241)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

	TAdapterError (see page 5245)	TAdapterError is the implementation of IAdapterError (see page 934).
	TAdapterErrorsList (see page 5248)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 5256)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 5265)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 5270)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 5280)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterHiddenField (see page 5288)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 5290)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 5297)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 5308)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5810) component.
	TAdapterMemoField (see page 5312)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 5320)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 5331)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 5339)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 5358)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterNextPageField (see page 5367)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 5375)	This is class WebAdapt.TAdapterObjectErrorsList.
	TAdapterPasswordField (see page 5379)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 5384)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterSubComponent (see page 5394)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 5407)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 5422)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 5427)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 5435)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 5467)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 5475)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 5485)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 5504)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 5515)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 5519)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 5565)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 5575)	tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 5583)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 5590)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 5600)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).
	TCustomAdapterImageField (see page 5609)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 5624)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

	TCustomAdapterMultiValueField (see page 5632)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 5642)	TCustomPagedAdapterAction (see page 5708) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 5649)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 5657)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5649).
	TCustomEndUserAdapter (see page 5662)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 5673)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5662).
	TCustomEndUserSessionAdapter (see page 5678)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 5687)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 5703)	TCustomEndUserAdapterAction (see page 5673) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5687).
	TCustomPagedAdapter (see page 5707)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 5708)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5733).
	TCustomStringsValuesList (see page 5714)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 5724)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 5733)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.
	TEndUserAdapter (see page 5750)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 5762)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 5770)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 5779)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 5789)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 5799)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5789).
	TImplementedAdapterAction (see page 5803)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 5805)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 5810)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 5821)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5810).
	TPagedAdapter (see page 5830)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 5843)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 5850)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 5852)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 5855)	TValuesListValueField is an adapter field used to access the Value (see page 5357) of the current list item of a values list adapter.

Constants

Name	Description
sActionRequestLong (see page 5907)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 5907)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 5908)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 5908)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 5908)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 5909)	This is constant WebAdapt.sLoginUserName.

sNextPageAction (see page 5909)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 5909)	This is constant WebAdapt.sPrevPageAction.

Functions

	Name	Description
	AddDefaultActions (see page 5883)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 5883)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 5884)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 5884)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 5884)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 5885)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 5885)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 5885)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 5886)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 5886)	This is function WebAdapt.RaiseMultipleValuesException.

Interfaces

	Name	Description
	IAdapterEditor (see page 5860)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 5861)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 5862)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 5863)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 5864)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 5865)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 5866)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 5868)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 5869)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 5871)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 5872)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 5873)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 5874)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 5875)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 5875)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 5876)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 5877)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 5878)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 5879)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 5879)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 5880)	This is class WebAdapt.IWebGetFieldPrevValue.

Structs, Records, Enums

	Name	Description
	TAdapterHiddenFieldOption (see page 5887)	This is record WebAdapt.TAdapterHiddenFieldOption.
	TIteratorMethod (see page 5887)	This is record WebAdapt.TIteratorMethod.

Types

Name	Description
IActionFieldValue (see page 5890)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 5890)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 5890)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 5891)	This is type WebAdapt.TActionExecuteEvent.
TActionExecuteHandledEvent (see page 5891)	This is type WebAdapt.TActionExecuteHandledEvent.
TActionGetEnabledEvent (see page 5891)	This is type WebAdapt.TActionGetEnabledEvent.
TActionGetParamsEvent (see page 5892)	This is type WebAdapt.TActionGetParamsEvent.

TActionRequestFieldsEvent (see page 5892)	This is type WebAdapt.TActionRequestFieldsEvent.
TActionRequestFieldsHandledEvent (see page 5892)	This is type WebAdapt.TActionRequestFieldsHandledEvent.
TAdapterActionExecuteEvent (see page 5893)	This is type WebAdapt.TAdapterActionExecuteEvent.
TAdapterActionExecuteHandledEvent (see page 5893)	This is type WebAdapt.TAdapterActionExecuteHandledEvent.
TAdapterActionGetParamsEvent (see page 5893)	This is type WebAdapt.TAdapterActionGetParamsEvent.
TAdapterActionsClass (see page 5894)	This is type WebAdapt.TAdapterActionsClass.
TAdapterFieldBooleanValueEvent (see page 5894)	This is type WebAdapt.TAdapterFieldBooleanValueEvent.
TAdapterFieldFileEvent (see page 5894)	This is type WebAdapt.TAdapterFieldFileEvent.
TAdapterFieldFileListEvent (see page 5895)	This is type WebAdapt.TAdapterFieldFileListEvent.
TAdapterFieldGetBooleanEvent (see page 5895)	This is type WebAdapt.TAdapterFieldGetBooleanEvent.
TAdapterFieldGetMultiValueEvent (see page 5895)	This is type WebAdapt.TAdapterFieldGetMultiValueEvent.
TAdapterFieldGetMultiValueStringEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.
TAdapterFieldGetStringEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetStringEvent.
TAdapterFieldGetValueCountEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetValueCountEvent.
TAdapterFieldGetValueEvent (see page 5897)	This is type WebAdapt.TAdapterFieldGetValueEvent.
TAdapterFieldHasValueEvent (see page 5897)	This is type WebAdapt.TAdapterFieldHasValueEvent.
TAdapterFieldsClass (see page 5897)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 5898)	This is type WebAdapt.TAdapterFieldStringEvent.
TAdapterFieldStringsEvent (see page 5898)	This is type WebAdapt.TAdapterFieldStringsEvent.
TAdapterFieldValidateValueEvent (see page 5898)	This is type WebAdapt.TAdapterFieldValidateValueEvent.
TAdapterFieldValueEvent (see page 5899)	This is type WebAdapt.TAdapterFieldValueEvent.
TAdapterHiddenFieldOptions (see page 5899)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 5899)	This is type WebAdapt.TEndUserGetNameEvent.
TEndUserGetUserIDEvent (see page 5900)	This is type WebAdapt.TEndUserGetUserIDEvent.
TEndUserHandledEvent (see page 5900)	This is type WebAdapt.TEndUserHandledEvent.
TEndUserIsLoggedInEvent (see page 5900)	This is type WebAdapt.TEndUserIsLoggedInEvent.
TEndUserRightsEvent (see page 5901)	This is type WebAdapt.TEndUserRightsEvent.
TEndUserUserIDEvent (see page 5901)	This is type WebAdapt.TEndUserUserIDEvent.
TGetEOFEvent (see page 5901)	This is type WebAdapt.TGetEOFEvent.
TGetRecordCountEvent (see page 5902)	This is type WebAdapt.TGetRecordCountEvent.
TGetRecordEvent (see page 5902)	This is type WebAdapt.TGetRecordEvent.
TGetRecordIndexEvent (see page 5902)	This is type WebAdapt.TGetRecordIndexEvent.
THandledEvent (see page 5903)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 5903)	This is type WebAdapt.TImageFieldGetHREFEvent.
TImageFieldGetImageEvent (see page 5903)	This is type WebAdapt.TImageFieldGetImageEvent.
TImageFieldGetParamsEvent (see page 5904)	This is type WebAdapt.TImageFieldGetParamsEvent.
TIterateRecordsEvent (see page 5904)	This is type WebAdapt.TIterateRecordsEvent.
TLoginUserIDEvent (see page 5904)	This is type WebAdapt.TLoginUserIDEvent.
TObjectList (see page 5905)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 5905)	This is type WebAdapt.TValidateAdapterFieldEvent.
TWebFormGetPageNameEvent (see page 5905)	This is type WebAdapt.TWebFormGetPageNameEvent.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 5906)	This is variable WebAdapt.NotifyPossibleHTMLChange.

57.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterException (see page 5166)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 5167)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 5169)	tabstractfieldvaluesadapter is the base class for adapters that have a single default field.
	TAdapter (see page 5179)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 5190)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 5198)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 5203)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5427) component.
	TAdapterApplicationTitleField (see page 5207)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 5212)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 5220)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 5226)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 5233)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 5237)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 5241)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.
	TAdapterError (see page 5245)	TAdapterError is the implementation of IAdapterError (see page 934).
	TAdapterErrorsList (see page 5248)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 5256)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 5265)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 5270)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 5280)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterHiddenField (see page 5288)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 5290)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 5297)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 5308)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5810) component.
	TAdapterMemoField (see page 5312)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 5320)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 5331)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 5339)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 5358)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterNextPageField (see page 5367)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 5375)	This is class WebAdapt.TAdapterObjectErrorsList.

	TAdapterPasswordField (see page 5379)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 5384)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5733).
	TAdapterSubComponent (see page 5394)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 5407)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 5422)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 5427)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 5435)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 5467)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 5475)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 5485)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 5504)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 5515)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 5519)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 5565)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 5575)	tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 5583)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 5590)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 5600)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).
	TCustomAdapterImageField (see page 5609)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 5624)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.
	TCustomAdapterMultiValueField (see page 5632)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 5642)	TCustomPagedAdapterAction (see page 5708) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 5649)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 5657)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5649).
	TCustomEndUserAdapter (see page 5662)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 5673)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5662).
	TCustomEndUserSessionAdapter (see page 5678)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 5687)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 5703)	TCustomEndUserAdapterAction (see page 5673) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5687).
	TCustomPagedAdapter (see page 5707)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 5708)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5733).
	TCustomStringsValuesList (see page 5714)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 5724)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 5733)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

	TEndUserAdapter (see page 5750)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 5762)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 5770)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 5779)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 5789)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 5799)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5789).
	TImplementedAdapterAction (see page 5803)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 5805)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 5810)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 5821)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5810).
	TPagedAdapter (see page 5830)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 5843)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 5850)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 5852)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 5855)	TValuesListValueField is an adapter field used to access the Value (see page 5357) of the current list item of a values list adapter.

57.1.1 EAdapterException Class

EAdapterException is raised by WebSnap adapter objects.

Class Hierarchy

Pascal

```
EAdapterException = class(EWebBrokerException);
```

C++

```
class EAdapterException : public EWebBrokerException;
```

File

WebAdapt

Description

EAdapterException is raised when a WebSnap adapter object encounters an error. EAdapterException provides an ID (see page 5167) property to record application-defined errors.

57.1.1.1 EAdapterException Members

The following tables list the members exposed by EAdapterException.

Public Methods

	Name	Description
	CreateID (see page 5167)	Create exception object with error ID (see page 5167).

Public Properties

	Name	Description
	ID (see page 5167)	Application error ID.

57.1.1.2 EAdapterException.CreateID Constructor

Create exception object with error ID (see page 5167).

Pascal

```
constructor CreateID(const AMsg: string; AID: Integer; Reserved: Integer = 0);
```

C++

```
__fastcall EAdapterException(const UnicodeString AMsg, int AID, int Reserved = 0);
```

Description

CreateID creates an EAdapterException (see page 5166) object in the usual way, and also assigns AID to the ID (see page 5167) property.

Reserved is not implemented.

57.1.1.3 EAdapterException Properties

The properties of the EAdapterException class are listed here.

Public Properties

	Name	Description
	ID (see page 5167)	Application error ID.

57.1.1.3.1 EAdapterException.ID Property

Application error ID.

Pascal

```
property ID: Integer;
```

C++

```
__property int ID;
```

Description

The ID property returns an application-defined value relating to the error. To specify an ID value, create an EAdapterException (see page 5166) object using the CreateID (see page 5167) constructor (Delphi) or the constructor syntax that includes an ID as a parameter (C++). For exceptions raised by built-in WebSnap adapters, ID is always 0.

See Also

CreateID (see page 5167)

57.1.2 EAdapterFieldException Class

EAdapterFieldException is an exception class for errors by adapters using fields.

Class Hierarchy

Pascal

```
EAdapterFieldException = class(EAdapterException);
```

C++

```
class EAdapterFieldException : public EAdapterException;
```

File

WebAdapt

Description

EAdapterFieldException is raised when an adapter encounters an error using a field.

57.1.2.1 EAdapterFieldException Members

The following tables list the members exposed by EAdapterFieldException.

Public Methods

	Name	Description
	Create (see page 5168)	Create an EAdapterFieldException (see page 5167) instance.
	CreateID (see page 5167)	Create exception object with error ID (see page 5167).

Public Properties

	Name	Description
	FieldName (see page 5169)	FieldName contains the name of the field associated with the error.
	ID (see page 5167)	Application error ID.

57.1.2.2 EAdapterFieldException.Create Constructor

Create an EAdapterFieldException (see page 5167) instance.

Pascal

```
constructor Create(const AMsg: string; const AFieldName: string; AID: Integer = 0);
```

C++

```
__fastcall EAdapterFieldException(const UnicodeString AMsg, const UnicodeString AFieldName,
int AID = 0);
```

Description

Create instantiates an instance of EAdapterFieldException (see page 5167).

AMsg is the string containing the runtime error message to display in the exception dialog box. AMsg can be a hard-coded string, or can be a function call that returns a string.

AFieldName and AID are assigned to the FieldName (see page 5169) and ID (see page 5167) properties.

See Also

FieldName (see page 5169)

ID (see page 5167)

57.1.2.3 EAdapterFieldException Properties

The properties of the EAdapterFieldException class are listed here.

Public Properties

	Name	Description
	FieldName (see page 5169)	FieldName contains the name of the field associated with the error.

57.1.2.3.1 EAdapterFieldException.FieldName Property

FieldName contains the name of the field associated with the error.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

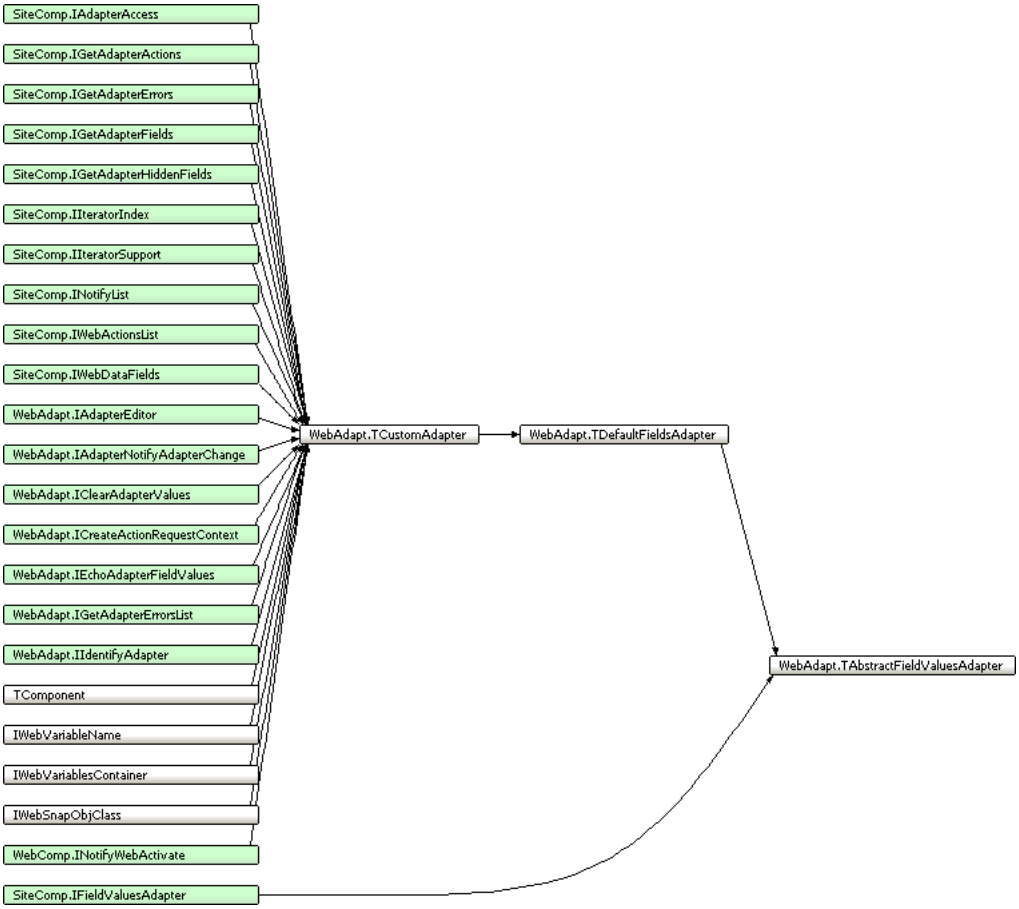
```
__property AnsiString FieldName = {read=FFieldName, write=FFieldName};
property FieldName: string;
```

FieldName contains the name of the field associated with the error.

57.1.3 TAbstractFieldValuesAdapter Class

tabstractfieldvaluesadapter is the base class for adapters that have a single default field.

Class Hierarchy



Pascal

```
TAbstractFieldValuesAdapter = class(TDefaultFieldsAdapter, IFieldValuesAdapter);
```

C++

```
class TAbstractFieldValuesAdapter : public TDefaultFieldsAdapter, public IFieldValuesAdapter;
```

File

WebAdapt

Description

Derive from tabstractfieldvaluesadapter to define adapter components that automatically include a single adapter field called ‘ValueField'.

Do not use TAbstractFieldValuesAdpter directly. It includes many abstract or, in C++ parlance, pure virtual methods, which must be implemented by a descendant class such as TFieldValuesAdapter (see page 5789).

See Also

TFieldValuesAdapter (see page 5789)

57.1.3.1 TAbstractFieldValuesAdapter Members

The following tables list the members exposed by TAbstractFieldValuesAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetField (see page 5175)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.

A	GetValueFieldClass (see page 5175)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 5176)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5176)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5177)	Provides the underlying implementation for the EndIterator (see page 5529) method.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
A	ImplGetField (see page 5177)	Provides the underlying implementation for the GetField (see page 5175) method.
V	ImplGetFieldsList (see page 5177)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
A	ImplGetValueIndex (see page 5178)	Provides the underlying implementation for the GetValueIndex (see page 5176) method.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5178)	Provides the underlying implementation for the NextIteration (see page 5551) method.
V	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
V	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
V	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
V	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
V	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
V	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5179)	Provides the underlying implementation for the StartIterator (see page 5553) method.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.

	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.

	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.3.2 TAbstractFieldValuesAdapter Methods

The methods of the TAbstractFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
	GetField (see page 5175)	Returns the object that represents the single field of this adapter.
A	GetValueFieldClass (see page 5175)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 5176)	Returns an index that identifies the current adapter field value.
V	ImplCanAddFieldClass (see page 5176)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.

	ImplEndIterator (see page 5177)	Provides the underlying implementation for the EndIterator (see page 5529) method.
	ImplGetField (see page 5177)	Provides the underlying implementation for the GetField (see page 5175) method.
	ImplGetFieldsList (see page 5177)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.
	ImplGetValueIndex (see page 5178)	Provides the underlying implementation for the GetValueIndex (see page 5176) method.
	ImplNextIteration (see page 5178)	Provides the underlying implementation for the NextIteration (see page 5551) method.
	ImplStartIterator (see page 5179)	Provides the underlying implementation for the StartIterator (see page 5553) method.

57.1.3.2.1 TAbstractFieldValuesAdapter.GetField Method

Returns the object that represents the single field of this adapter.

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

Call GetField to access the single field for this adapter. GetField returns an adapter field that is an instance of the class that the GetValueFieldClass (see page 5175) method returns.

GetField calls the abstract or, in C++ terminology, pure virtual ImplGetField (see page 5177) method to obtain the adapter field instance it returns. Descendant classes override ImplGetField (see page 5177) to provide the underlying implementation for GetField.

See Also

ImplGetField (see page 5177)

GetValueFieldClass (see page 5175)

57.1.3.2.2 TAbstractFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the single adapter field this adapter includes.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual; abstract;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass() = 0;
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field that the GetField (see page 5175) method returns. The return value must be a descendant of TComponent and is typically a descendant of TAdapterNamedField (see page 5339).

In TAbstractFieldValuesAdapter (see page 5169), GetValueFieldClass is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to indicate the type of field this adapter includes.

See Also

GetField (see page 5175)

TAdapterNamedField (see page 5339)

57.1.3.2.3 TAbstractFieldValuesAdapter.GetValueIndex Method

Returns an index that identifies the current adapter field value.

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

Although this adapter only includes a single adapter field, that field may have several values (for example, if it represents a column in a table). When the field has several values, GetValueIndex is the index of the current value. This index may be the same as the value that the GetIteratorIndex (see page 5537) method returns if the application is using the adapter's iteration support. However, TAbstractFieldValuesAdapter (see page 5169) introduces GetValueIndex as a separate method for descendants that support multiple values that change using some other mechanism.

GetValueIndex calls the abstract or, in C++ terminology, pure virtual ImplGetValueIndex (see page 5178) method to obtain the index of the current value. Descendant classes override ImplGetValueIndex (see page 5178) to provide the underlying implementation for GetValueIndex.

See Also

ImplGetValueIndex (see page 5178)

GetIteratorIndex (see page 5537)

57.1.3.2.4 TAbstractFieldValuesAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddFieldClass (see page 5527) method calls ImplCanAddFieldClass to determine whether this adapter can include a field that is an instance of a specified class. This allows the adapter editor to indicate the available fields at design time.

AParent is an instance of this adapter class to which the field is to be added.

AClass is the class of the object that represents the adapter field.

ImplCanAddFieldClass returns true if AClass is the same as (or a descendant of) the class that the GetValueFieldClass (see page 5175) method returns, indicating that this is the only child field this adapter supports. It returns false for any other class.

See Also

CanAddFieldClass (see page 5527)

GetValueFieldClass (see page 5175)

57.1.3.2.5 TAbstractFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 5529) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 5529) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator raises an exception, because this method should never be called at the TAbstractFieldValuesAdapter (see page 5169) level: The TAbstractFieldValuesAdapter (see page 5169) ImplStartIterator (see page 5179) method returns false, indicating that this adapter does not support iteration.

Descendant classes that implement support for iterating over field values override this method to end that iteration process.

See Also

EndIterator (see page 5529)

ImplStartIterator (see page 5179)

ImplNextIteration (see page 5178)

57.1.3.2.6 TAbstractFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 5175) method.

Pascal

```
function ImplGetField: TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * ImplGetField() = 0;
```

Description

The GetField (see page 5175) method calls ImplGetField to obtain the object that represents this adapter's single field. Unlike the GetField (see page 5175) method, ImplGetField is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining the adapter field object.

In TAbstractFieldValuesAdapter (see page 5169), ImplGetField is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetField (see page 5175) method.

See Also

GetField (see page 5175)

57.1.3.2.7 TAbstractFieldValuesAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 5536) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The GetFieldsList (see page 5536) method calls ImplGetFieldsList to fill a string list with a list of all fields that the adapter includes. Each entry in the list is the name of one of the adapter's field, and has an associated object that is the class of the object that represents that adapter field. GetFieldsList (see page 5536) is part of the design-time support for editing adapters.

AList is an existing string list to fill with entries for all of the fields this adapter can include.

In TAbstractFieldValuesAdapter (see page 5169), ImplGetFieldsList clears AList and inserts a single entry for a field named 'ValueField'. It associates the class that the GetValueFieldClass (see page 5175) method returns with this field name.

See Also

GetValueFieldClass (see page 5175)

GetFieldsList (see page 5536)

57.1.3.2.8 TAbstractFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 5176) method.

Pascal

```
function ImplGetValueIndex: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int ImplGetValueIndex() = 0;
```

Description

The GetValueIndex (see page 5176) method calls ImplGetValueIndex to obtain the index of current value of this adapter's single field. Unlike the GetValueIndex (see page 5176) method, ImplGetValueIndex is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining this index.

In TAbstractFieldValuesAdapter (see page 5169), ImplGetValueIndex is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetValueIndex (see page 5176) method.

See Also

GetValueIndex (see page 5176)

57.1.3.2.9 TAbstractFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 5551) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 5551) method calls ImplNextIteration when iterating over field values to determine whether there

is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 5551) or to StartIterator (see page 5553).

In TAbstractFieldValuesAdapter (see page 5169), ImplNextIteration always returns false, because TAbstractFieldValuesAdapter (see page 5169) does not implement support for iteration. Descendant classes that support iterating over field values override this method to check whether their is a next value and if so, move to it.

See Also

NextIteration (see page 5551)

ImplStartIterator (see page 5179)

ImplEndIterator (see page 5177)

57.1.3.2.10 TAbstractFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 5553) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 5553) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way. Depending on the adapter, it may be supplied by the caller, or created by the ImplStartIterator method.

In TAbstractFieldValuesAdapter (see page 5169), ImplNextIteration (see page 5178) always returns false, because TAbstractFieldValuesAdapter (see page 5169) does not implement support for obtaining any field values. Descendant classes that override this method to check whether they can move to the first value in the field's set of values. If so, they move to the first value and return true. Otherwise they return false.

See Also

StartIterator (see page 5553)

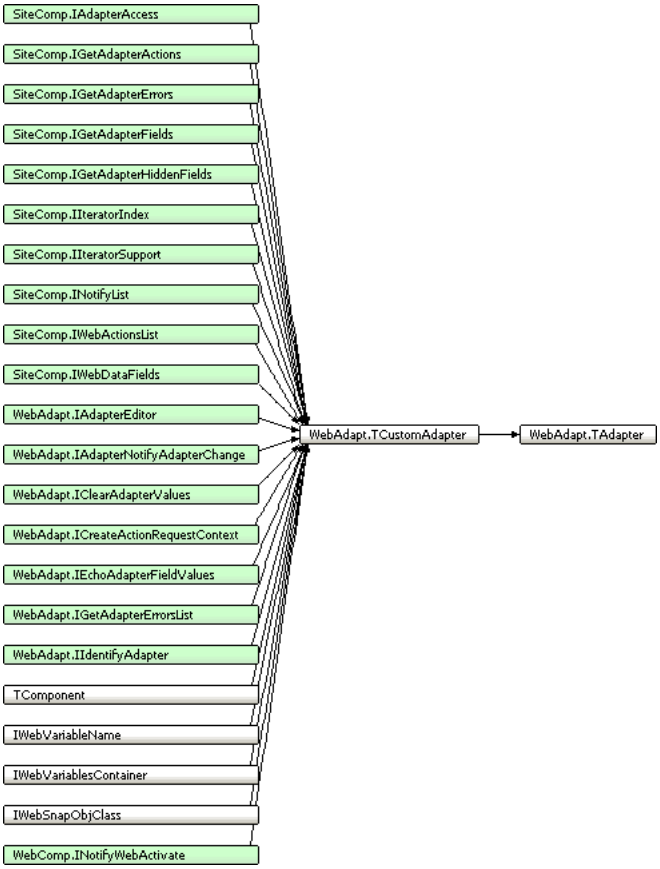
ImplEndIterator (see page 5177)

ImplNextIteration (see page 5178)

57.1.4 TAdapter Class

TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.

Class Hierarchy



Pascal

```
TAdapter = class(TCustomAdapter);
```

C++

```
class TAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Use TAdapter to define a custom interface to application logic. Adapters contain field components and action components that represent data items and commands, respectively. Server-side script statements access the value of Adapter fields and the parameters of Adapter actions in order to build HTML forms and tables. Server-side script can be written by hand or automatically generated by the TAdapterPageProducer component. When an HTML form containing Adapter commands is submitted, the TAdapterDispatcher may automatically call an Adapter action component to execute the command. When an HTTP request for an adapter image field is received the TAdapterDispatcher may automatically call an Adapter field to retrieve the image.

For each field component added to TAdapter, you must implement an event handler to retrieve the data. For each action component added to TAdapter, you must implement an event handler to execute the command. Other adapter components, such as TDataSetAdapter and TLoginFormAdapter (see page 5810), have built in fields and actions that do not require events.

57.1.4.1 TAdapter Members

The following tables list the members exposed by TAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.

	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
V	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
V	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
V	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.

	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5186)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5186)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5187)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5187)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5187)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5188)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5188)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5189)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5189)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5189)	Occurs when the adapter needs to iterate over its data records.

Published Properties

	Name	Description
	Actions (see page 5184)	Actions is the container for adapter actions components.
	Data (see page 5185)	Data is the container for adapter field components.
	ModifyAccess (see page 5185)	This is ModifyAccess, a member of class TAdapter.
	ViewAccess (see page 5185)	Identifies the rights required to view this adapter's fields.

57.1.4.2 TAdapter Properties

The properties of the TAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5184)	Actions is the container for adapter actions components.
	Data (see page 5185)	Data is the container for adapter field components.
	ModifyAccess (see page 5185)	This is ModifyAccess, a member of class TAdapter.
	ViewAccess (see page 5185)	Identifies the rights required to view this adapter's fields.

57.1.4.2.1 TAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion,

deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 5198)

57.1.4.2.2 TAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 5265)

57.1.4.2.3 TAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TAdapter.

57.1.4.2.4 TAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

implhasviewaccess (see page 5546)

57.1.4.3 TAdapter Events

The events of the TAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5186)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5186)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5187)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5187)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5187)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5188)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5188)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5189)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5189)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5189)	Occurs when the adapter needs to iterate over its data records.

57.1.4.3.1 TAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.4.3.2 TAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.4.3.3 TAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnBeforeUpdateFields (see page 5564)

Errors (see page 5559)

57.1.4.3.4 TAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5555)

OnBeforeValidateFields (see page 5564)

Errors (see page 5559)

57.1.4.3.5 TAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.4.3.6 TAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.4.3.7 TAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnAfterUpdateFields (see page 5562)

Errors (see page 5559)

57.1.4.3.8 TAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5555)

OnAfterValidateFields (see page 5563)

Errors (see page 5559)

57.1.4.3.9 TAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.4.3.10 TAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when called to retrieve their value.

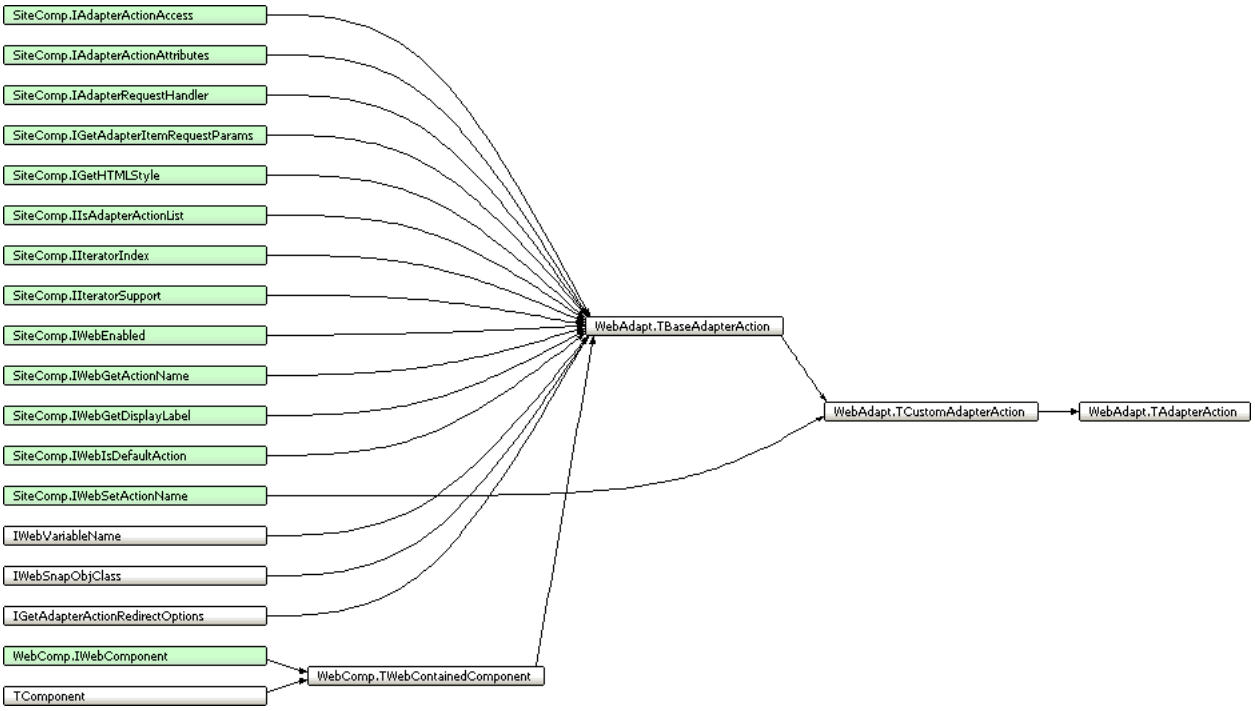
See Also

IIteratorSupport (see page 996)

57.1.5 TAdapterAction Class

TAdapterAction is a generic adapter action that relies on event handlers.

Class Hierarchy



Pascal

```
TAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

TAdapterAction publishes the properties in TCustomAdapterAction (see page 5565). TCustomAdapterAction (see page 5565) calls event handlers to retrieve the adapter action's parameters, get the state of the action, and execute the adapter action. Add TAdapterAction to TAdapter (see page 5179) or other adapter components types when you need an adapter action that will be controlled by event handlers. Other adapter action components declare events but also have built-in logic to work with a particular type of adapter. For example, TAdapterNextPageAction (see page 5358) has built-in logic to take the end user to the next page provided by TPagedAdapter (see page 5830). TAdapterAction, on the other hand, does not have this sort of built-in logic.

See Also

TAdapterActions (see page 5198)

TAdapter (see page 5179)

TAdapterDispatcher (see page 6122)

57.1.5.1 TAdapterAction Members

The following tables list the members exposed by TAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.

	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.

R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5196)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5196)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5197)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5197)	Called before the adapter action response is generated.

	OnExecute (see page 5197)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5198)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5198)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ActionName (see page 5194)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5194)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5195)	Defines the rights required to execute this action.
	RedirectOptions (see page 5195)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.5.2 TAdapterAction Properties

The properties of the TAdapterAction class are listed here.

Published Properties

	Name	Description
	ActionName (see page 5194)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5194)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5195)	Defines the rights required to execute this action.
	RedirectOptions (see page 5195)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.5.2.1 TAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;__property UnicodeString ActionName;__property
UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 5570)

ImplSetActionName (see page 5571)

57.1.5.2.2 TAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.5.2.3 TAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6759)

hasexecuteaccess (see page 5539)

57.1.5.2.4 TAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

57.1.5.3 TAdapterAction Events

The events of the TAdapterAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5196)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5196)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5197)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5197)	Called before the adapter action response is generated.
	OnExecute (see page 5197)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5198)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5198)	Specifies name/value pairs needed to execute this adapter action.

57.1.5.3.1 TAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.5.3.2 TAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.5.3.3 TAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.5.3.4 TAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.5.3.5 TAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest.

See Also

ImplExecuteActionRequest (see page 5570)

OnGetParams (see page 5466)

57.1.5.3.6 TAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that this adapter action should not be enabled.

See Also

ImplWebEnabled (see page 5571)

57.1.5.3.7 TAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

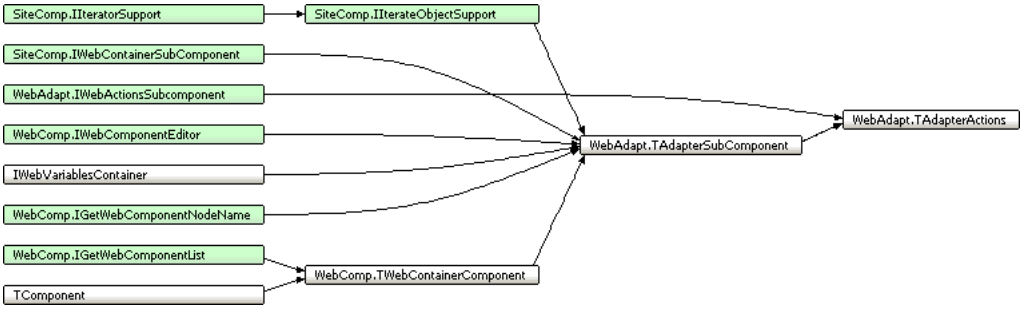
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.6 TAdapterActions Class

TAdapterActions manages the list of actions that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterActions = class(TAdapterSubComponent, IWebActionsSubcomponent);
```

C++

```
class TAdapterActions : public TAdapterSubComponent, public IWebActionsSubcomponent;
```

File

WebAdapt

Description

TAdapterActions is the type of the Actions property on an adapter component. It manages a list that contains all the actions that belong to the adapter. The action objects (TBaseAdapterAction (see page 5435) descendants) are available using the WebActions (see page 5203) property. TAdapterActions also includes support for associating the actions with variable names so that server-side script can execute the actions.

TAdapterActions represents the list of explicitly assigned actions. Adapters that include a set of default actions use a TAdapterDefaultActions (see page 5220) object instead.

See Also

TCustomAdapter (see page 5519)

TAdapterFields (see page 5265)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

TBaseAdapterAction (see page 5435)

57.1.6.1 TAdapterActions Members

The following tables list the members exposed by TAdapterActions.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5201)	Indicates whether the associated adapter supports a specified class of action component.
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.

	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5399)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5401)	Returns the value of the VariablesInUse (see page 5407) property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5201)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
V	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5202)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5404)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetDefaultWebComponents (see page 6040)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5202)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
V	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
V	Destroy (see page 6038)	Frees an instance of TWebContainerComponent (see page 6036).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5400)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
V	HasParent (see page 5402)	Indicates whether this component has an associated adapter.
	NextIteration (see page 997)	Advances to the objects next item.
V	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	WebActions (see page 5203)	Lists the action components that this object manages.

57.1.6.2 TAdapterActions Methods

The methods of the TAdapterActions class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5201)	Indicates whether the associated adapter supports a specified class of action component.
	ImplCanAddClass (see page 5201)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
	ImplGetWebComponentNodeName (see page 5202)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	SetWebParent (see page 5202)	Introduces a method to associate this component with an adapter.

57.1.6.2.1 TAdapterActions.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of action component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 5198) uses this method internally as part of its implementation for the CanAddClass (see page 5396) method. This allows the TAdapterActions (see page 5198) object to indicate, at design time, which types of action components are valid as components in the WebActions (see page 5203) property list.

AParent is the instance of TAdapterActions (see page 5198) to which the child actions are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child action, false otherwise.

AdapterCanAddClass calls the CanAddActionClass method of its associated adapter to determine whether it accepts the specified action class.

See Also

WebActions (see page 5203)

Adapter (see page 5406)

ImplCanAddClass (see page 5201)

CanAddActionClass (see page 5527)

57.1.6.2.2 TAdapterActions.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5396) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5396) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterActions (see page 5198) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterActions (see page 5198) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter action class that the associated adapter recognizes and accepts.

AClass is a custom adapter action (a descendant of TCustomAdapterAction (see page 5565) that is not a descendant of TImplementedAdapterAction (see page 5803)).

See Also

CanAddClass (see page 5396)

AdapterCanAddClass (see page 5201)

TCustomAdapterAction (see page 5565)

TImplementedAdapterAction (see page 5803)

57.1.6.2.3 TAdapterActions.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5401) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Actions'.

See Also

GetWebComponentNodeName (see page 5401)

57.1.6.2.4 TAdapterActions.SetWebParent Method

Introduces a method to associate this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

TAdapterSubComponent introduces SetWebParent as a virtual method for setting the read-only Adapter property.

Value is the adapter component that acts as a parent to this collection.

In TAdapterSubComponent, SetWebParent does nothing. Descendant classes override this method to set the Adapter property and populate the list with actions or fields appropriate to that adapter.

See Also

Adapter (see page 5406)

57.1.6.3 TAdapterActions Properties

The properties of the TAdapterActions class are listed here.

Public Properties

	Name	Description
	WebActions (see page 5203)	Lists the action components that this object manages.

57.1.6.3.1 TAdapterActions.WebActions Property

Lists the action components that this object manages.

Pascal

```
property WebActions: TWebComponentList;
```

C++

```
__property TWebComponentList WebActions;
```

Description

Read (see page 3727) WebActions to access the list of action components. Each component in this list is a descendant of TBaseAdapterAction (see page 5435).

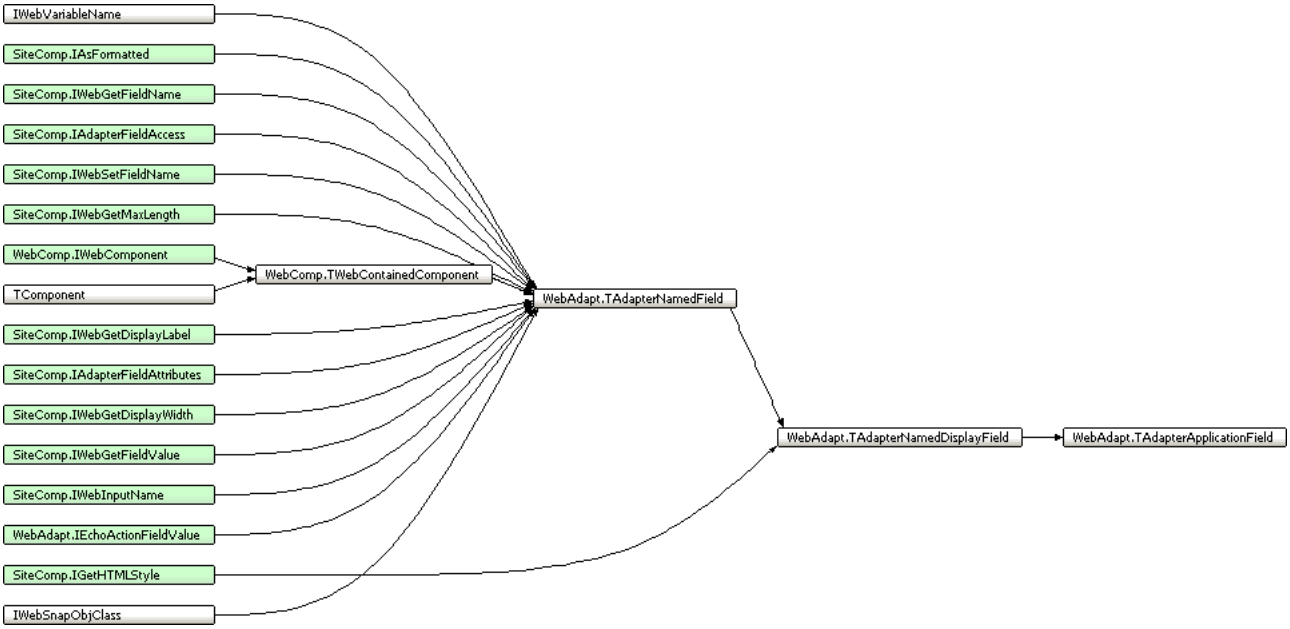
See Also

TBaseAdapterAction (see page 5435)

57.1.7 TAdapterApplicationField Class

TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5427) component.

Class Hierarchy



Pascal

```
TAdapterApplicationField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterApplicationField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterApplicationField implements functionality useful to an adapter field that can be a child of TApplicationAdapter (see page 5427).

See Also

TApplicationAdapter (see page 5427)

TAdapterNamedDisplayField (see page 5331)

TAdapterApplicationTitleField (see page 5207)

57.1.7.1 TAdapterApplicationField Members

The following tables list the members exposed by TAdapterApplicationField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.

	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.

	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
R	Adapter (see page 5207)	Returns the parent adapter of this adapter field.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.

	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.7.2 TAdapterApplicationField Properties

The properties of the TAdapterApplicationField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5207)	Returns the parent adapter of this adapter field.

57.1.7.2.1 TAdapterApplicationField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

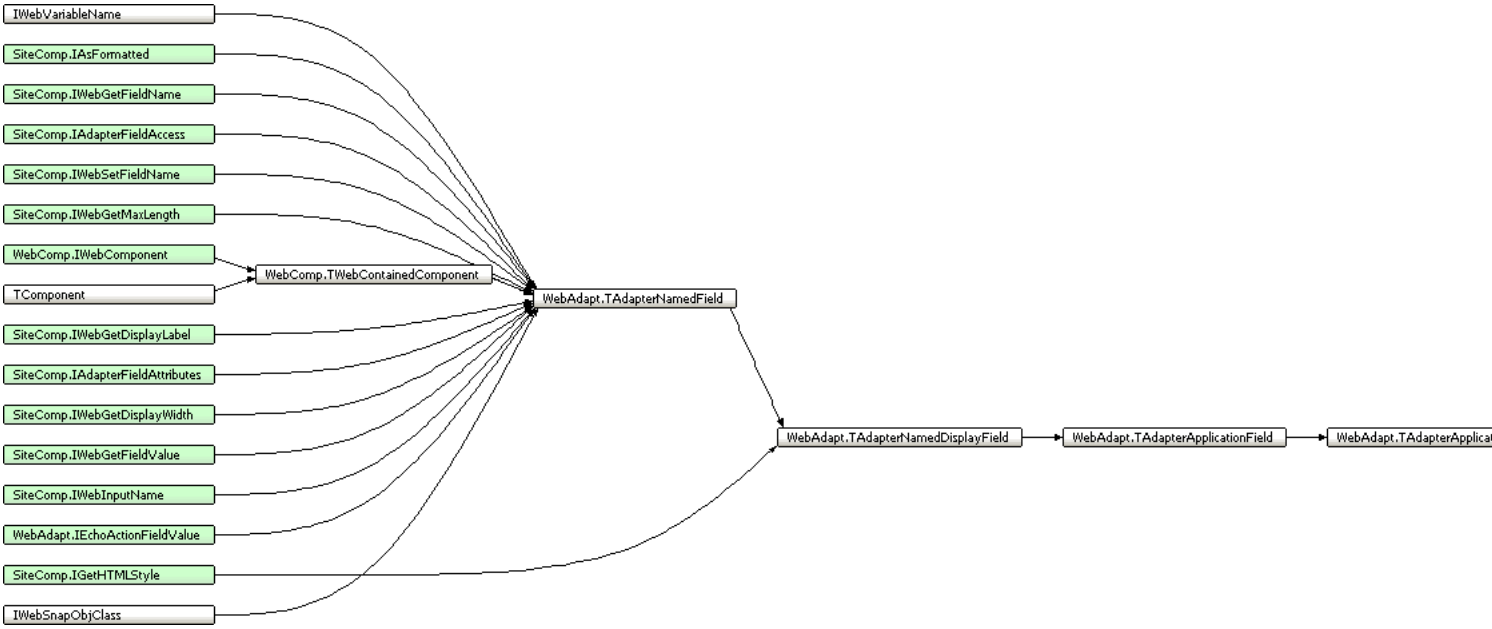
See Also

TCustomApplicationAdapter (see page 5649)

57.1.8 TAdapterApplicationTitleField Class

TAdapterApplicationTitleField is an adapter field used to access the application title.

Class Hierarchy



Pascal

```
TAdapterApplicationTitleField = class(TAdapterApplicationField);
```

C++

```
class TAdapterApplicationTitleField : public TAdapterApplicationField;
```

File

WebAdapt

Description

TAdapterApplicationTitleField is added to TApplicationAdapter (see page 5427).

See Also

TAdapterApplicationField (see page 5203)

TApplicationAdapter (see page 5427)

TCustomApplicationAdapter (see page 5649)

57.1.8.1 TAdapterApplicationTitleField Members

The following tables list the members exposed by TAdapterApplicationTitleField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.

	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5211)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5211)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5207)	Returns the parent adapter of this adapter field.
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.8.2 TAdapterApplicationTitleField Methods

The methods of the TAdapterApplicationTitleField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5211)	Returns the name used to access this adapter field with server-side script.
	ImplGetValue (see page 5211)	Returns the value associated with this adapter field.

57.1.8.2.1 TAdapterApplicationTitleField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'Title'. This allows server-side script to access this adapter field using the parent adapter's Title property.

57.1.8.2.2 TAdapterApplicationTitleField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of the parent adapter's Title property. Server-side script accesses the result of this function using the DisplayText property of this adapter field. Server-side script may use this value to display the application title on a Web page.

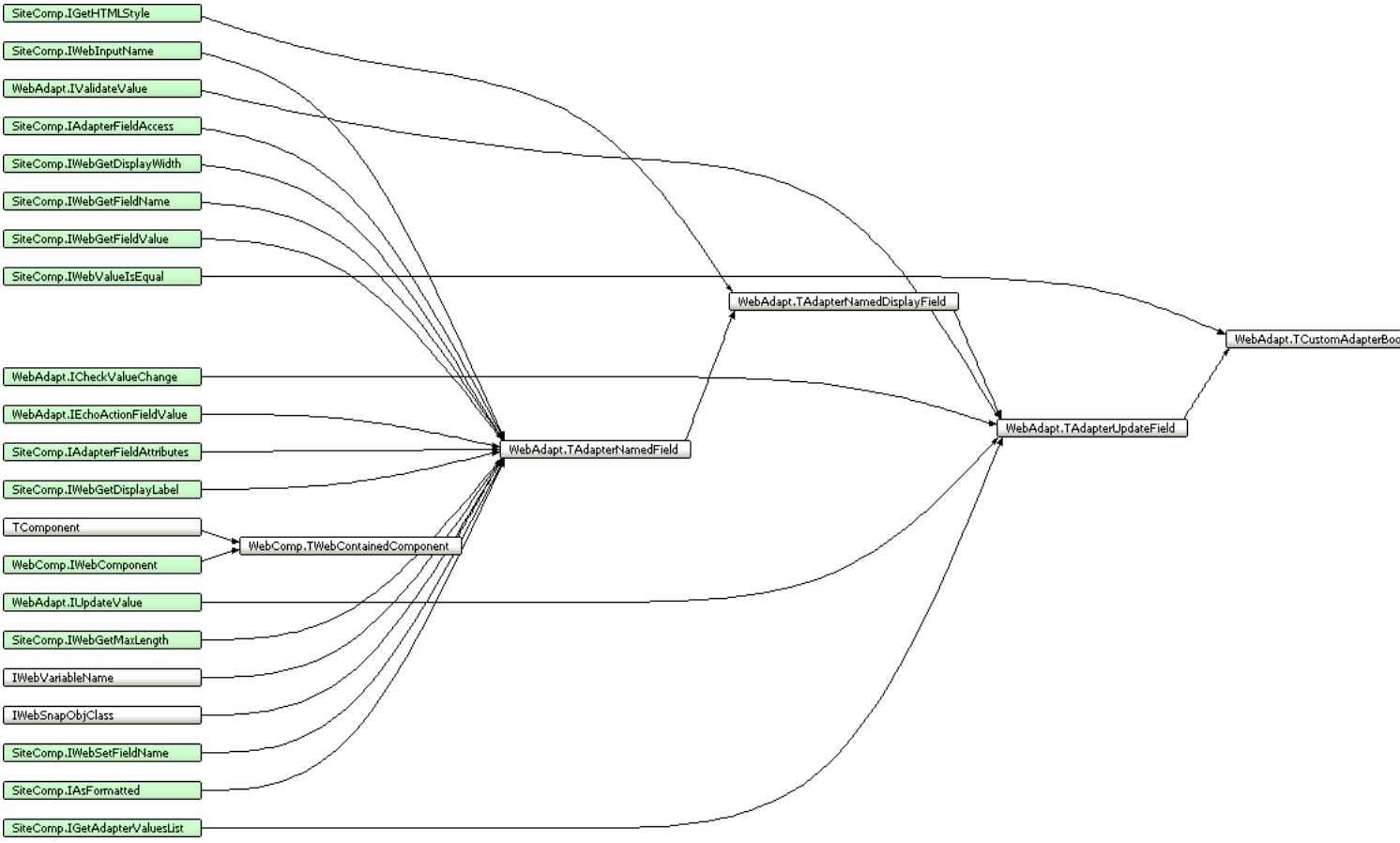
See Also

ApplicationTitle (see page 5657)

57.1.9 TAdapterBooleanField Class

TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.

Class Hierarchy



Pascal

```
TAdapterBooleanField = class(TCustomAdapterBooleanField);
```

C++

```
class TAdapterBooleanField : public TCustomAdapterBooleanField;
```

File

WebAdapt

Description

TAdapterBooleanField publishes the properties in TCustomAdapterBooleanField (see page 5575). TCustomAdapterBooleanField (see page 5575) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterBooleanField to TAdapter (see page 5179) or other adapter components types when you need an adapter field that works with a Boolean value. There are other adapter field components that use event handlers. They differ in the type and

number of values that are supported. See TAdapterField (see page 5256), TAdapterMultiValueField (see page 5320), TAdapterMemoField (see page 5312), TAdapterFileField (see page 5270) and TAdapterImageField (see page 5297).

See Also

TAdapter (see page 5179)

TAdapterField (see page 5256)

TAdapterMemoField (see page 5312)

TAdapterMultiValueField (see page 5320)

TAdapterFileField (see page 5270)

TAdapterImageField (see page 5297)

57.1.9.1 TAdapterBooleanField Members

The following tables list the members exposed by TAdapterBooleanField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValue (see page 5580)	Retrieves a Boolean value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5580)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5581)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).

	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetValue (see page 5581)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5582)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.

	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
R	Adapter (see page 5356)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
V	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	IsEqual (see page 1060)	This is IsEqual, a member of class IWebValueIsEqual.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 5219)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5219)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5220)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5217)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5217)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5217)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5218)	Identifies the rights required to update this field.
	ReadOnly (see page 5218)	Indicates whether this adapter field is read-only.
	ValuesList (see page 5218)	Associates a list of values with the adapter field.
	ViewAccess (see page 5219)	Identifies the rights required to view this field.

57.1.9.2 TAdapterBooleanField Properties

The properties of the TAdapterBooleanField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5217)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5217)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5217)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5218)	Identifies the rights required to update this field.
	ReadOnly (see page 5218)	Indicates whether this adapter field is read-only.
	ValuesList (see page 5218)	Associates a list of values with the adapter field.
	ViewAccess (see page 5219)	Identifies the rights required to view this field.

57.1.9.2.1 TAdapterBooleanField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.9.2.2 TAdapterBooleanField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.9.2.3 TAdapterBooleanField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.9.2.4 TAdapterBooleanField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.9.2.5 TAdapterBooleanField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.9.2.6 TAdapterBooleanField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1014)

TStringsValuesList (see page 5843)

TDataSetValuesList

57.1.9.2.7 TAdapterBooleanField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.9.3 TAdapterBooleanField Events

The events of the TAdapterBooleanField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 5219)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5219)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5220)	Event handler called to validate value(s) entered by the end-user.

57.1.9.3.1 TAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5581)

57.1.9.3.2 TAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5582)

57.1.9.3.3 TAdapterBooleanField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

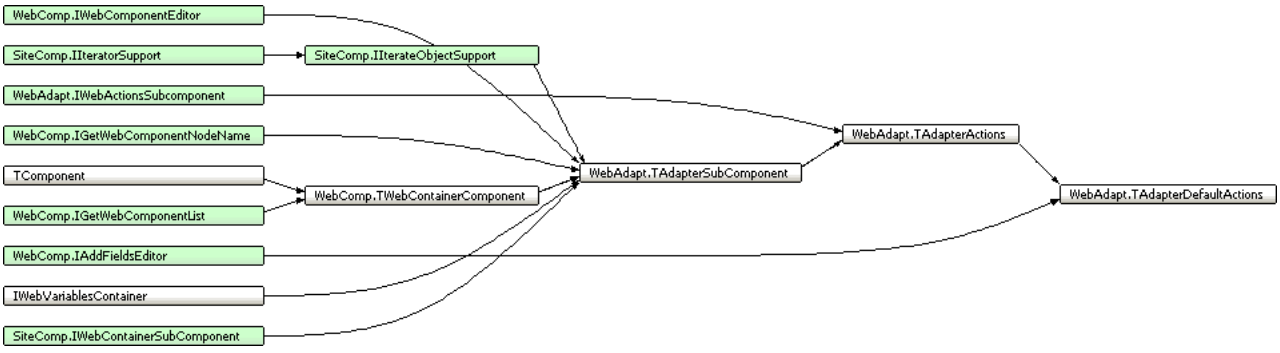
DoOnValidateValue (see page 5414)

IActionFieldValue

57.1.10 TAdapterDefaultActions Class

TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.

Class Hierarchy



Pascal

```
TAdapterDefaultActions = class(TAdapterActions, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultActions : public TAdapterActions, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultActions is the type of the Actions property on an adapter component. It manages two lists of actions (TBaseAdapterAction (see page 5435) descendants) that belong to the adapter: the list of actions that have been explicitly added to the adapter, and the list of actions that the adapter includes by default. The explicitly added action objects are available using the WebActions (see page 5203) property. The default actions are available using the DefaultWebComponents (see page 5226) property.

See Also

TDefaultFieldsAdapter (see page 5724)

TAdapterFields (see page 5265)

TAdapterActions (see page 5198)

TAdapterDefaultFields (see page 5226)

TBaseAdapterAction (see page 5435)

57.1.10.1 TAdapterDefaultActions Members

The following tables list the members exposed by TAdapterDefaultActions.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5201)	Indicates whether the associated adapter supports a specified class of action component.
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 5224)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 5224)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 5224)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
V	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5224)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5225)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5225)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5201)	Provides the underlying implementation for the CanAddClass (see page 5396) method.

	ImplCanAddClass (see page 5402)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetWebComponentNodeName (see page 5202)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	ImplGetWebComponentNodeName (see page 5403)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 5225)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5202)	Introduces a method to associate this component with an adapter.
	SetWebParent (see page 5405)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.
	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
	ClearDefaultComponents (see page 5223)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 5226) property.
	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
	Destroy (see page 5222)	Frees an instance of TAdapterDefaultActions (see page 5220).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetParentComponent (see page 5400)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 5402)	Indicates whether this component has an associated adapter.
	NextIteration (see page 997)	Advances to the objects next item.
	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5226)	Lists the action components that the associated adapter includes by default.
	WebActions (see page 5203)	Lists the action components that this object manages.

57.1.10.2 TAdapterDefaultActions.Destroy Destructor

Frees an instance of TAdapterDefaultActions (see page 5220).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultActions();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3382)

57.1.10.3 TAdapterDefaultActions Methods

The methods of the TAdapterDefaultActions class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 5224)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 5224)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 5224)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
	GetDefaultWebComponents (see page 5224)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5225)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
	GetVariablesInUse (see page 5225)	Returns the value of the VariablesInUse property.
	SetDefaultWebComponents (see page 5225)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
	ClearDefaultComponents (see page 5223)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 5226) property.

57.1.10.3.1 TAdapterDefaultActions.ClearDefaultComponents Method

Empties the list of default actions that is the value of the DefaultWebComponents (see page 5226) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default actions. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default actions.

See Also

DefaultWebComponents (see page 5226)

57.1.10.3.2 TAdapterDefaultActions.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.

57.1.10.3.3 TAdapterDefaultActions.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.

57.1.10.3.4 TAdapterDefaultActions.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultActions.

57.1.10.3.5 TAdapterDefaultActions.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6039)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

57.1.10.3.6 TAdapterDefaultActions.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.

57.1.10.3.7 TAdapterDefaultActions.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5407)

GetWebComponents (see page 6039)

57.1.10.3.8 TAdapterDefaultActions.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and

TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5399)

IsDefaultField (see page 6015)

57.1.10.4 TAdapterDefaultActions Properties

The properties of the TAdapterDefaultActions class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5226)	Lists the action components that the associated adapter includes by default.

57.1.10.4.1 TAdapterDefaultActions.DefaultWebComponents Property

Lists the action components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read (see page 3727) DefaultWebComponents to access the list of action components that the adapter uses when you do not explicitly add any actions to this object. Each default action is a descendant of TAdapterNamedAction.

If you explicitly add action components to this object, the adapter uses the actions in the WebActions (see page 5203) property instead.

See Also

TBaseAdapterAction (see page 5435)

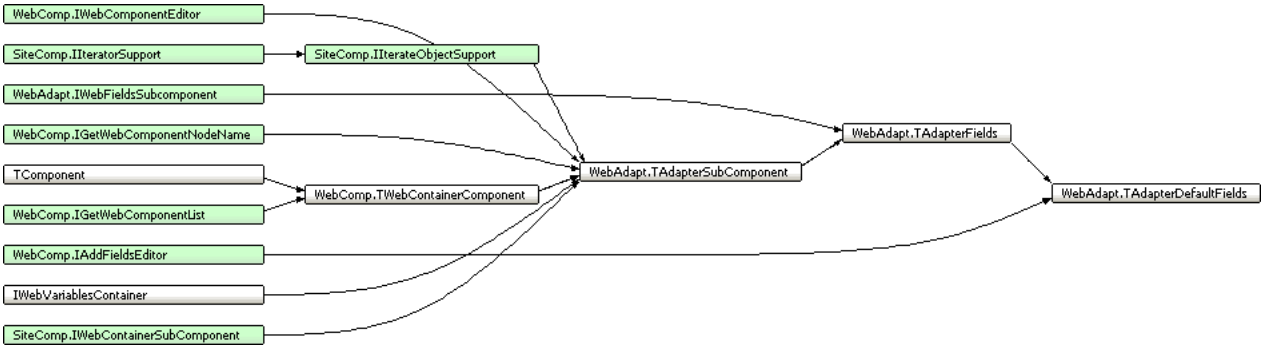
WebActions (see page 5203)

Adapter (see page 5406)

57.1.11 TAdapterDefaultFields Class

TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.

Class Hierarchy



Pascal

```
TAdapterDefaultFields = class(TAdapterFields, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultFields : public TAdapterFields, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultFields is the type of the Data property on an adapter component. It manages two lists of fields (TAdapterNamedField (see page 5339) descendants) that belong to the adapter: the list of fields that have been explicitly added to the adapter, and the list of fields that the adapter includes by default. The explicitly added field objects are available using the WebFields (see page 5270) property. The default fields are available using the DefaultWebComponents (see page 5232) property.

See Also

TDefaultFieldsAdapter (see page 5724)

TAdapterActions (see page 5198)

TAdapterDefaultActions (see page 5220)

TAdapterFields (see page 5265)

TAdapterNamedField (see page 5339)

57.1.11.1 TAdapterDefaultFields Members

The following tables list the members exposed by TAdapterDefaultFields.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5267)	Indicates whether the associated adapter supports a specified class of field component.
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.

	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 5230)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 5230)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 5230)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
V	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5230)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5231)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5231)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5268)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
V	ImplCanAddClass (see page 5402)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
V	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5269)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
V	ImplGetWebComponentNodeName (see page 5403)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5231)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5269)	Associates this component with an adapter.
V	SetWebParent (see page 5405)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
V	ClearDefaultComponents (see page 5229)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 5232) property.
V	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
V	Destroy (see page 5229)	Frees an instance of TAdapterDefaultFields (see page 5226).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5400)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
V	HasParent (see page 5402)	Indicates whether this component has an associated adapter.

	NextIteration (see page 997)	Advances to the objects next item.
V	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
R	DefaultWebComponents (see page 5232)	Lists the field components that the associated adapter includes by default.
R	WebFields (see page 5270)	Lists the field components that this object manages.

57.1.11.2 TAdapterDefaultFields.Destroy Destructor

Frees an instance of TAdapterDefaultFields (see page 5226).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultFields();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3382)

57.1.11.3 TAdapterDefaultFields Methods

The methods of the TAdapterDefaultFields class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 5230)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 5230)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 5230)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
V	GetDefaultWebComponents (see page 5230)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5231)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
V	GetVariablesInUse (see page 5231)	Returns the value of the VariablesInUse property.
V	SetDefaultWebComponents (see page 5231)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
V	ClearDefaultComponents (see page 5229)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 5232) property.

57.1.11.3.1 TAdapterDefaultFields.ClearDefaultComponents Method

Empties the list of default fields that is the value of the DefaultWebComponents (see page 5232) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default fields. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default fields.

See Also

DefaultWebComponents (see page 5232)

57.1.11.3.2 TAdapterDefaultFields.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.

57.1.11.3.3 TAdapterDefaultFields.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.

57.1.11.3.4 TAdapterDefaultFields.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultFields.

57.1.11.3.5 TAdapterDefaultFields.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6039)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

57.1.11.3.6 TAdapterDefaultFields.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.

57.1.11.3.7 TAdapterDefaultFields.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5407)

GetWebComponents (see page 6039)

57.1.11.3.8 TAdapterDefaultFields.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5399)

IsDefaultField (see page 6015)

57.1.11.4 TAdapterDefaultFields Properties

The properties of the TAdapterDefaultFields class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5232)	Lists the field components that the associated adapter includes by default.

57.1.11.4.1 TAdapterDefaultFields.DefaultWebComponents Property

Lists the field components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read (see page 3727) DefaultWebComponents to access the list of field components that the adapter uses when you do not explicitly add any field components to this object. Each default field component is a descendant of TAdapterNamedField (see page 5339).

If you explicitly add field components to this object, the adapter uses the fields in the WebFields (see page 5270) property instead.

See Also

TAdapterNamedField (see page 5339)

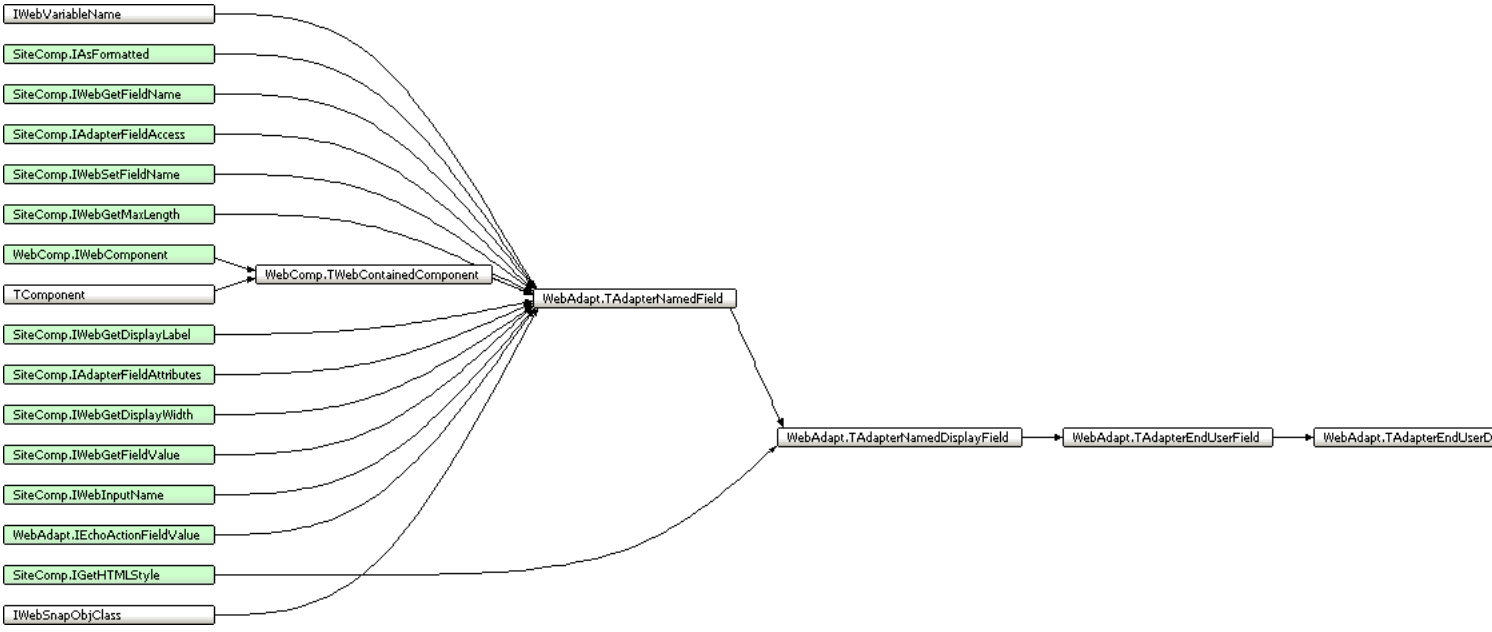
WebFields (see page 5270)

Adapter (see page 5406)

57.1.12 TAdapterEndUserDisplayNameField Class

TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.

Class Hierarchy



Pascal

```
TAdapterEndUserDisplayNameField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserDisplayNameField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserDisplayNameField is added to TApplicationAdapter (see page 5427).

See Also

TAdapterEndUserField (see page 5237)

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

57.1.12.1 TAdapterEndUserDisplayNameField Members

The following tables list the members exposed by TAdapterEndUserDisplayNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5236)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5236)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.

	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5240)	Returns the parent adapter of this adapter field.
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.

	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.12.2 TAdapterEndUserDisplayNameField Methods

The methods of the TAdapterEndUserDisplayNameField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 5236)	Returns the name used to access this adapter field with server-side script.
V	ImplGetValue (see page 5236)	Returns the value associated with this adapter field.

57.1.12.2.1 TAdapterEndUserDisplayNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'DisplayName'. This allows server-side script to access this adapter field using the parent adapter's DisplayName property.

57.1.12.2.2 TAdapterEndUserDisplayNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetDisplayNameValue. Server-side script accesses the result of this function using the

DisplayText property of this adapter field. Server-side script may use this value to display the end-user's name on a Web page.

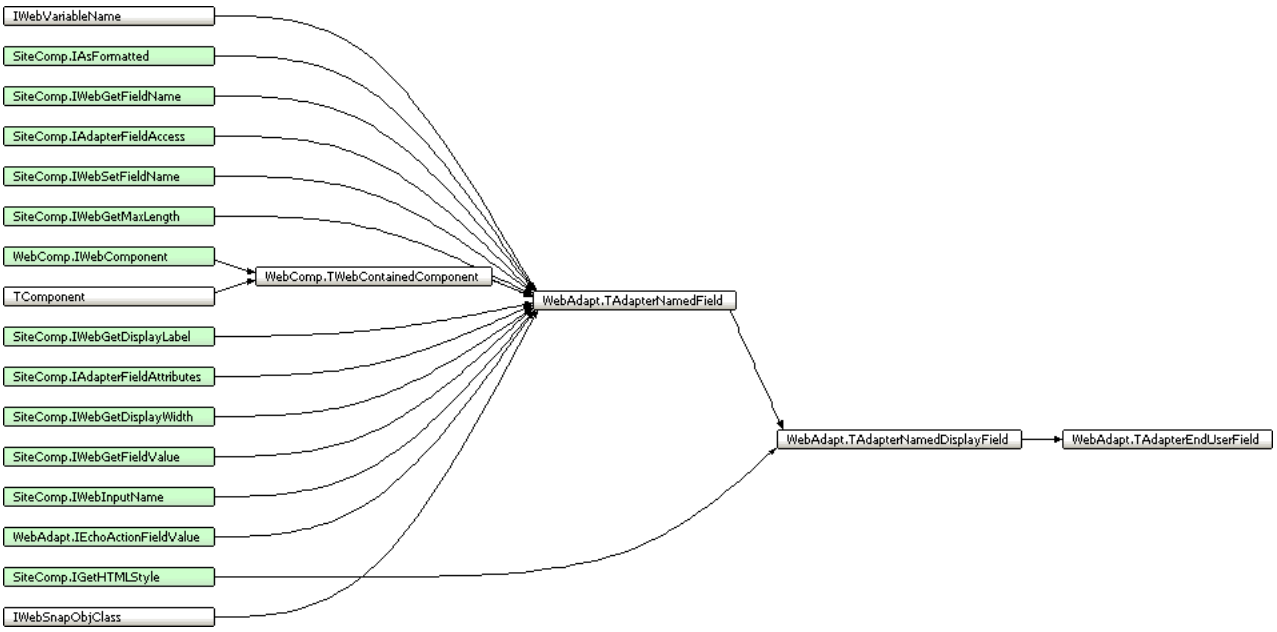
See Also

GetDisplayNameValue (see page 5494)

57.1.13 TAdapterEndUserField Class

TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.

Class Hierarchy



Pascal

```
TAdapterEndUserField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterEndUserField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterEndUserField implements functionality useful to an adapter field that can be a child of TEndUserAdapter (see page 5750) or TEndUserSessionAdapter (see page 5779).

See Also

TAdapterNamedDisplayField (see page 5331)

TBaseEndUserAdapter (see page 5485)

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

TAdapterEndUserLoggedInField (see page 5241)

TAdapterEndUserDisplayNameField (see page 5233)

57.1.13.1 TAdapterEndUserField Members

The following tables list the members exposed by TAdapterEndUserField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.

	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5240)	Returns the parent adapter of this adapter field.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.13.2 TAdapterEndUserField Properties

The properties of the TAdapterEndUserField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5240)	Returns the parent adapter of this adapter field.

57.1.13.2.1 TAdapterEndUserField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

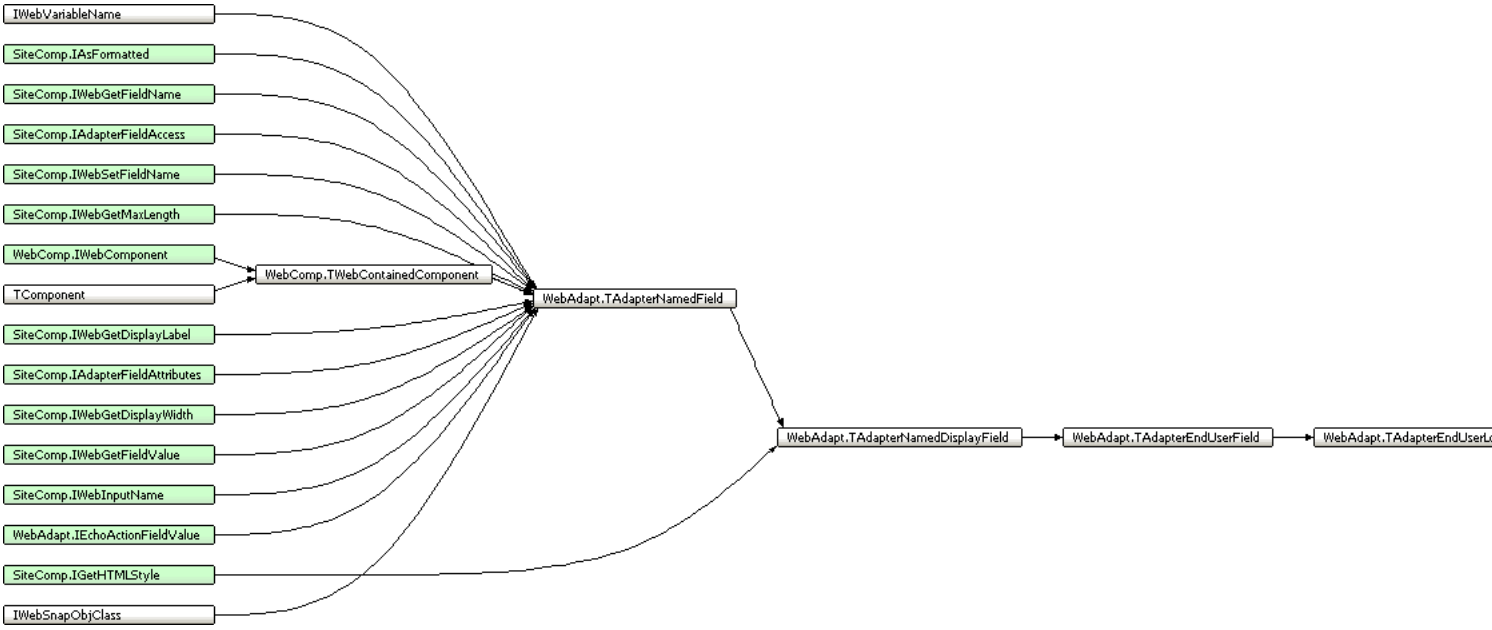
See Also

TBaseEndUserAdapter (see page 5485)

57.1.14 TAdapterEndUserLoggedInField Class

TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

Class Hierarchy



Pascal

```
TAdapterEndUserLoggedInField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserLoggedInField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserLoggedInField is added to TApplicationAdapter (see page 5427). The logged-in flag indicates whether the user is logged-in. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

See Also

TApplicationAdapter (see page 5427)

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

57.1.14.1 TAdapterEndUserLoggedInField Members

The following tables list the members exposed by TAdapterEndUserLoggedInField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5244)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5244)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.

	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5240)	Returns the parent adapter of this adapter field.
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.

	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.14.2 TAdapterEndUserLoggedInField Methods

The methods of the TAdapterEndUserLoggedInField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 5244)	Returns the name used to access this adapter field with server-side script.
V	ImplGetValue (see page 5244)	Returns the value associated with this adapter field.

57.1.14.2.1 TAdapterEndUserLoggedInField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'LoggedIn'. This allows server-side script to access this adapter field using the parent adapter's LoggedIn property.

57.1.14.2.2 TAdapterEndUserLoggedInField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetLoggedInValue. Server-side script accesses the result of this function using the Value (

see page 5357) property of this adapter field. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

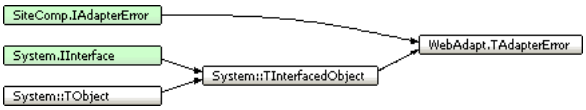
See Also

GetLoggedInValue (see page 5495)

57.1.15 TAdapterError Class

TAdapterError is the implementation of IAdapterError (see page 934).

Class Hierarchy



Pascal

```
TAdapterError = class(TInterfacedObject, IAdapterError);
```

C++

```
class TAdapterError : public TInterfacedObject, public IAdapterError;
```

File

WebAdapt

Description

TAdapterError is the implementation class for IAdapterError (see page 934).

See Also

IAdapterError (see page 934)

57.1.15.1 TAdapterError Members

The following tables list the members exposed by TAdapterError.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetErrorText (see page 5247)	Returns the value of the ErrorText property.
	GetField (see page 5247)	Returns the value of the Field property.
	GetID (see page 5248)	Returns the value of the ID property.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5247)	Create instantiates an instance of TAdapterError (see page 5245). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ErrorText (see page 936)	Returns text describing the error.
R	Field (see page 936)	Returns text describing the error.
R	ID (see page 936)	Returns error ID.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

57.1.15.2 TAdapterError.Create Constructor

Create instantiates an instance of TAdapterError (see page 5245). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterError(TAdapterErrorsList AErrors, int AIndex);
```

Description

```
constructor Create(AErrors: TAdapterErrorsListTAdapterErrorsList; AIndex: Integer); public;
```

Create instantiates an instance of TAdapterError (see page 5245). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

AErrors is the adapter errors list object that creates this TAdapterError (see page 5245) object.

AIndex is the index of this adapter error in the list that AErrors maintains.

57.1.15.3 TAdapterError Methods

The methods of the TAdapterError class are listed here.

Protected Methods

	Name	Description
	GetErrorText (see page 5247)	Returns the value of the ErrorText property.
	GetField (see page 5247)	Returns the value of the Field property.
	GetID (see page 5248)	Returns the value of the ID property.

57.1.15.3.1 TAdapterError.GetErrorText Method

Returns the value of the ErrorText property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall UnicodeString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText property.

See Also

ErrorText (see page 936)

57.1.15.3.2 TAdapterError.GetField Method

Returns the value of the Field property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field property.

See Also

Field (see page 936)

57.1.15.3.3 TAdapterError.GetID Method

Returns the value of the ID property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID property.

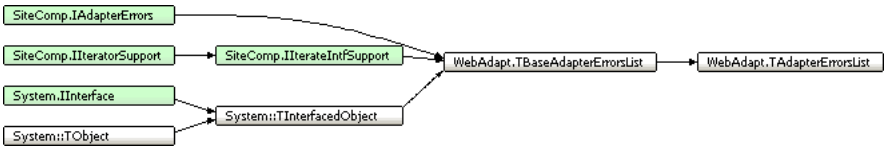
See Also

ID (see page 936)

57.1.16 TAdapterErrorsList Class

TAdapterErrorsList keeps track of errors that an adapter encounters.

Class Hierarchy



Pascal

```
TAdapterErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

TAdapterErrorsList acts as a repository for all the errors an adapter (or any of its subcomponents) encounters while processing an HTTP request message. Adapter components create an instance of this class to implement their Errors (see page 5475) property. Server-side scripts can use the Errors (see page 5475) property to access the errors in this list. The usual purpose of such a script is to display error messages to the end user.

To add new error objects to the list, call the AddError (see page 5252) method. To access the errors in the list, read its Errors (see page 5475) property. Errors (see page 5475) lists the IAdapterError (see page 934) interfaces of adapter error objects (TAdapterError (see page 5245)) that represent each error in the list. You can also access the individual properties of those error objects using the FieldNames (see page 5255), IDs (see page 5255), and Messages (see page 5255) properties of the TAdapterErrorsList instance.

TAdapterErrorsList implements the IAdapterErrorsList interface.

Note: Do not confuse TAdapterErrorsList with TAdapterErrorList. TAdapterErrorsList stores the errors that the WebSnap application encounters when using an adapter. TAdapterErrorList is a display component that displays the errors from a TAdapterErrorsList object.

See Also

TAdapterErrorList (see page 6365)

TCustomAdapter (see page 5519)

TAdapterError (see page 5245)

IAdapterErrors (see page 937)

IAdapterError (see page 934)

57.1.16.1 TAdapterErrorsList Members

The following tables list the members exposed by TAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	EndIterator (see page 5471)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 5471)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5471)	Returns a particular error.
	GetErrorCount (see page 5471)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5472)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5472)	Indicates whether the errors list contains objects associated with a particular adapter field.
	ImplClearErrors (see page 5253)	This is ImplClearErrors, a member of class TAdapterErrorsList.
	ImplDefineLabel (see page 5253)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
	ImplGetError (see page 5253)	This is ImplGetError, a member of class TAdapterErrorsList.
	ImplGetErrorCount (see page 5254)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
	ImplGetObjectErrors (see page 5254)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
	ImplHasObjectErrors (see page 5254)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.
	NextIteration (see page 5474)	Advances to the objects next item.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5474)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddError (see page 5252)	This is the overview for the AddError method overload.
	AddError (see page 5469)	This is the overview for the AddError method overload.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 938)	Clears the errors list.
	ClearErrors (see page 5470)	Clear error list.
	Create (see page 5251)	Creates and initializes an instance of TAdapterErrorsList (see page 5248).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	DefineLabel (see page 938)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 5470)	Define error label for field.
	Destroy (see page 5251)	Frees an instance of TAdapterErrorsList (see page 5248).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetCurrentIntf (see page 992)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 938)	Returns a particular error.
	GetErrorCount (see page 939)	Return the number of errors in the errors list.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 939)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 939)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 997)	Advances to the objects next item.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	ErrorCount (see page 940)	Returns the number of errors in the errors list.
	ErrorCount (see page 5474)	Number of error objects contained.
	Errors (see page 940)	Returns an error from the list.
	Errors (see page 5475)	Access to error objects.
	FieldNames (see page 5255)	Lists the names of the objects that encountered the errors in the list.
	IDs (see page 5255)	Lists the error codes of the errors in the list.
	Messages (see page 5255)	Lists the error messages for the errors in the list.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

57.1.16.2 TAdapterErrorsList.Create Constructor

Creates and initializes an instance of TAdapterErrorsList (see page 5248).

Pascal

```
constructor Create(AAdapter: TObject);
```

C++

```
__fastcall TAdapterErrorsList(TObject * AAdapter);
```

Description

Typically, WebSnap applications do not create their own instances of TAdapterErrorsList (see page 5248). Adapter objects instantiate this class to implement their Errors (see page 5475) property.

AAdapter is the adapter component that uses this object to implement its Errors (see page 5475) property.

See Also

Errors (see page 5559)

57.1.16.3 TAdapterErrorsList.Destroy Destructor

Frees an instance of TAdapterErrorsList (see page 5248).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterErrorsList();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the errors list is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

57.1.16.4 TAdapterErrorsList Methods

The methods of the TAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplClearErrors (see page 5253)	This is ImplClearErrors, a member of class TAdapterErrorsList.
	ImplDefineLabel (see page 5253)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
	ImplGetError (see page 5253)	This is ImplGetError, a member of class TAdapterErrorsList.
	ImplGetErrorCount (see page 5254)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
	ImplGetObjectErrors (see page 5254)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
	ImplHasObjectErrors (see page 5254)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

Public Methods

	Name	Description
	AddError (see page 5252)	This is the overview for the AddError method overload.

57.1.16.4.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TAdapterErrorsList.AddError (Exception, string) (see page 5252)	Adds an error to the list.
	TAdapterErrorsList.AddError (Exception, TObject) (see page 5252)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, string, Integer) (see page 5252)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, TObject, Integer) (see page 5252)	Adds an error to the list.

57.1.16.4.1.1 TAdapterErrorsList.AddError Method (Exception, TObject)

Adds an error to the list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); override;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
override;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); override;
procedure AddError(AException: Exception; AObject: TObject); override;
```

C++

```
virtual __fastcall AddError(Exception AException, const UnicodeString AObjectName = '');
virtual __fastcall AddError(const UnicodeString AMessage, const UnicodeString AObjectName =
'', int AID = 0);
virtual __fastcall AddError(const UnicodeString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the object's internal list. An error is described by three pieces of information:

An error message string. You can specify this string directly with the AMessage parameter, or using an Exception (see page 4015) object (with the AException parameter). If you specify an Exception (see page 4015) object, the object's Message property becomes the message string.

The name of an object (Variable) related to the error. You can specify this value as a string (with the AObjectName parameter) or as an object reference (with the AObject parameter). If you omit both these parameters, but supply an EAdapterFieldException (see page 5167) object as the AException parameter, TAdapterErrorsList (see page 5248) uses the FieldName property of that exception object. If you do not supply an AObject parameter, an AObjectName parameter, or an EAdapterFIeldException object as the AException parameter, TAdapterErrorsList (see page 5248) uses an empty string.

An error ID value. You can specify this value using the AID parameter. If you do not supply a value for AID but specify an EAdapterException (see page 5166) object as the AException parameter, TAdapterErrorsList (see page 5248) uses the ID property of that exception object. If you do not supply an AID parameter or an EAdapterException (see page 5166) object, TAdapterErrorsList (see page 5248) uses 0.

See Also

FieldName (see page 5169)

ID (see page 5167)

Messages (see page 5255)

FieldNames (see page 5255)

IDs (see page 5255)

57.1.16.4.2 TAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; override;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TAdapterErrorsList.

57.1.16.4.3 TAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterErrorsList.

57.1.16.4.4 TAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterErrorsList.

57.1.16.4.5 TAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TAdapterErrorsList.

57.1.16.4.6 TAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const UnicodeString AName);
```

Description

This is ImplGetObjectErrors, a member of class TAdapterErrorsList.

57.1.16.4.7 TAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const UnicodeString AName);
```

Description

This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

57.1.16.5 TAdapterErrorsList Properties

The properties of the TAdapterErrorsList class are listed here.

Public Properties

	Name	Description
	FieldNames (see page 5255)	Lists the names of the objects that encountered the errors in the list.
	IDs (see page 5255)	Lists the error codes of the errors in the list.
	Messages (see page 5255)	Lists the error messages for the errors in the list.

57.1.16.5.1 TAdapterErrorsList.FieldNames Property

Lists the names of the objects that encountered the errors in the list.

Pascal

```
property FieldNames [I: Integer]: string;
```

C++

```
__property UnicodeString FieldNames[int I];
```

Description

Use FieldNames to access the name of the object associated with a specified error. FieldNames values typically reflect the name of an adapter field that was the cause of the error. However, they can be any object reference for which there is a defined Variable that can be accessed from server-side script. If there is no object associated with an error, the value of FieldNames is an empty string.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of FieldNames for an error is set by the AddError (see page 5252) method when the error is added to the list.

See Also

AddError (see page 5252)

Field (see page 936)

FieldName (see page 5169)

57.1.16.5.2 TAdapterErrorsList.IDs Property

Lists the error codes of the errors in the list.

Pascal

```
property IDs [I: Integer]: integer;
```

C++

```
__property int IDs[int I];
```

Description

Use IDs to access the ID (error code) of a specified error in the list. If the error has no error code, the value of IDs is 0.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of IDs for an error is set by the AddError (see page 5252) method when the error is added to the list.

See Also

AddError (see page 5252)

ID (see page 936)

ID (see page 5167)

57.1.16.5.3 TAdapterErrorsList.Messages Property

Lists the error messages for the errors in the list.

Pascal

```
property Messages [I: Integer]: string;
```

C++

```
__property UnicodeString Messages[int I];
```

Description

Use Messages to access the error message for a specified error in the list.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of Messages for an error is set by the AddError (see page 5252) method when the error is added to the list.

See Also

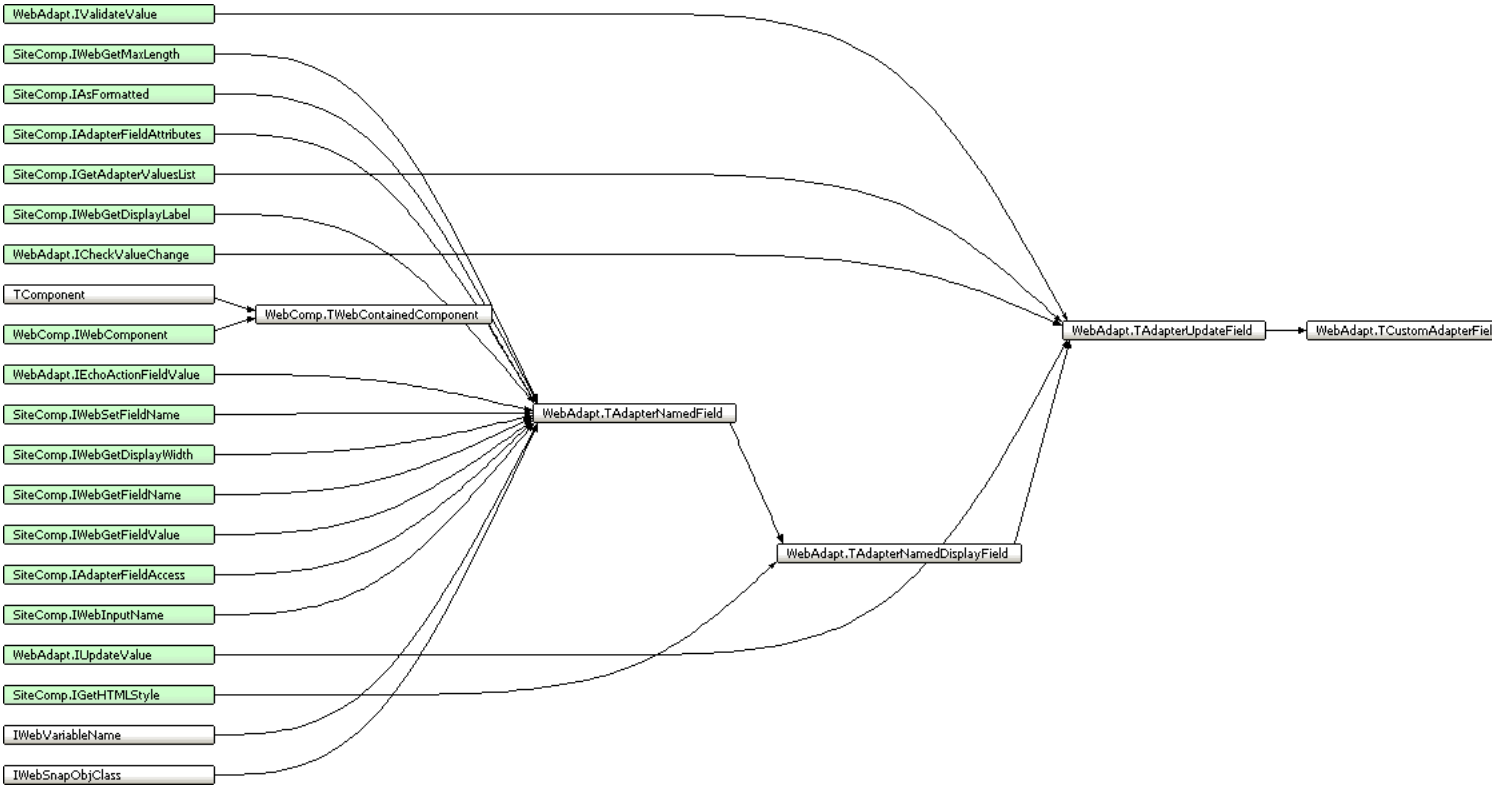
AddError (see page 5252)

ErrorText (see page 936)

57.1.17 TAdapterField Class

TAdapterField is an event handler based adapter field that provides a single value.

Class Hierarchy



Pascal

```
TAdapterField = class(TCustomAdapterField);
```

C++

```
class TAdapterField : public TCustomAdapterField;
```

File

WebAdapt

Description

TAdapterField publishes the properties in TCustomAdapterField (see page 5583). TCustomAdapterField (see page 5583) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterField to TAdapter (see page 5179) or other adapter components types when you need an adapter field that supports a single value using event handlers. Event handlers are used to provide the adapter field value(s), validate a value entered by the end-user, and update the adapter field with a value entered by the end-user.

There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. For example, TAdapterMultiValue field is used when the end-user can enter multiple values using a group of check boxes or a multiple selection select list. TAdapterFileField (see page 5270), TAdapterMemoField (see page 5312), TAdapterBooleanField (see page 5212) and TAdapterImageField (see page 5297) are other event driven adapter fields.

See Also

TAdapterUpdateField (see page 5407)

TCustomAdapterField (see page 5583)

TAdapter (see page 5179)

TAdapterMultiValueField (see page 5320)

TAdapterMemoField (see page 5312)

TAdapterFileField (see page 5270)

TAdapterBooleanField (see page 5212)

TAdapterImageField (see page 5297)

57.1.17.1 TAdapterField Members

The following tables list the members exposed by TAdapterField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5588)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.

	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
V	ImplGetValue (see page 5588)	Returns the value of this adapter field as a variant.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.

	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5589)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.

	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetDisplayText (see page 5264)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5264)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5264)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5265)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5261)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5261)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5261)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5262)	Identifies the rights required to update this field.
	ReadOnly (see page 5262)	Indicates whether this adapter field is read-only.
	Required (see page 5262)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5263)	Associates a list of values with the adapter field.
	ViewAccess (see page 5263)	Identifies the rights required to view this field.

57.1.17.2 TAdapterField Properties

The properties of the TAdapterField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5261)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5261)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5261)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5262)	Identifies the rights required to update this field.
	ReadOnly (see page 5262)	Indicates whether this adapter field is read-only.
	Required (see page 5262)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5263)	Associates a list of values with the adapter field.
	ViewAccess (see page 5263)	Identifies the rights required to view this field.

57.1.17.2.1 TAdapterField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.17.2.2 TAdapterField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.17.2.3 TAdapterField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.17.2.4 TAdapterField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.17.2.5 TAdapterField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.17.2.6 TAdapterField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5347)

57.1.17.2.7 TAdapterField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1014)

TStringsValuesList (see page 5843)

TDataSetValuesList

57.1.17.2.8 TAdapterField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.17.3 TAdapterField Events

The events of the TAdapterField class are listed here.

Published Events

	Name	Description
	OnGetDisplayText (see page 5264)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5264)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5264)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5265)	Event handler called to validate value(s) entered by the end-user.

57.1.17.3.1 TAdapterField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 5415)

57.1.17.3.2 TAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5588)

57.1.17.3.3 TAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

57.1.17.3.4 TAdapterField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

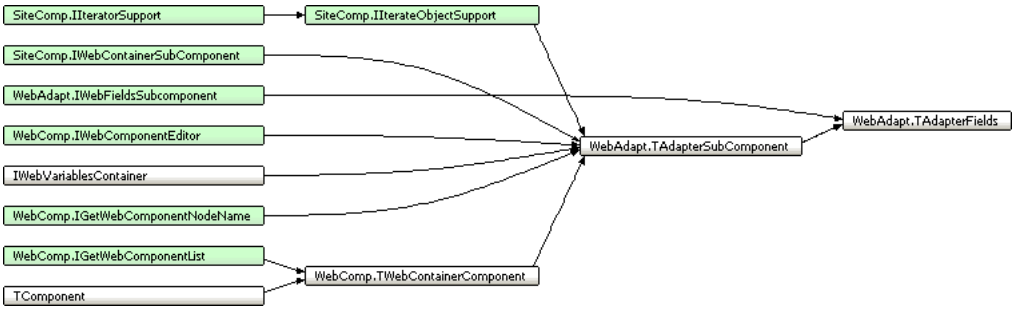
DoOnValidateValue (see page 5414)

IActionFieldValue

57.1.18 TAdapterFields Class

TAdapterFields manages the list of fields that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterFields = class(TAdapterSubComponent, IWebFieldsSubcomponent);
```

C++

```
class TAdapterFields : public TAdapterSubComponent, public IWebFieldsSubcomponent;
```

File

WebAdapt

Description

TAdapterFields is the type of the Data property on an adapter component. It manages a list that contains all the fields that belong to the adapter. The field objects (TAdapterNamedField (see page 5339) descendants) are available using the WebFields (see page 5270) property. TAdapterFields also includes support for associating the fields with variable names so that server-side script can use the field values.

TAdapterFields represents the list of explicitly assigned fields. Adapters that include a set of default fields use a TAdapterDefaultFields (see page 5226) object instead.

See Also

TCustomAdapter (see page 5519)

TAdapterActions (see page 5198)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

TAdapterNamedField (see page 5339)

57.1.18.1 TAdapterFields Members

The following tables list the members exposed by TAdapterFields.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5267)	Indicates whether the associated adapter supports a specified class of field component.
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5399)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5401)	Returns the value of the VariablesInUse (see page 5407) property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5268)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
V	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5269)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.

	SetDefaultWebComponents (see page 5404)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDefaultWebComponents (see page 6040)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5269)	Associates this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.
	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.
	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
	Destroy (see page 6038)	Frees an instance of TWebContainerComponent (see page 6036).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetParentComponent (see page 5400)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 5402)	Indicates whether this component has an associated adapter.
	NextIteration (see page 997)	Advances to the objects next item.
	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	WebFields (see page 5270)	Lists the field components that this object manages.

57.1.18.2 TAdapterFields Methods

The methods of the TAdapterFields class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5267)	Indicates whether the associated adapter supports a specified class of field component.
	ImplCanAddClass (see page 5268)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
	ImplGetWebComponentNodeName (see page 5269)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	SetWebParent (see page 5269)	Associates this component with an adapter.

57.1.18.2.1 TAdapterFields.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of field component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 5198) uses this method internally as part of its implementation for the CanAddClass (see page 5396) method. This allows the TAdapterFields (see page 5265) object to indicate, at design time, which types of components are valid as members of the WebFields (see page 5270) property list.

AParent is the instance of TAdapterFields (see page 5265) to which the child fields are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child field, false otherwise.

AdapterCanAddClass calls the CanAddFieldClass method of its associated adapter to determine whether it accepts the specified field class.

See Also

WebFields (see page 5270)

Adapter (see page 5406)

ImplCanAddClass (see page 5268)

CanAddFieldClass (see page 5527)

57.1.18.2.2 TAdapterFields.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5396) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5396) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterFields (see page 5265) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterFields (see page 5265) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter field class that the associated adapter recognizes and accepts.

AClass is a data entry field (a descendant of TAdapterUpdateField (see page 5407)).

See Also

CanAddClass (see page 5396)

AdapterCanAddClass (see page 5267)

TAdapterUpdateField (see page 5407)

57.1.18.2.3 TAdapterFields.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5401) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Fields'.

See Also

GetWebComponentNodeName (see page 5401)

57.1.18.2.4 TAdapterFields.SetWebParent Method

Associates this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

SetWebParent sets up the association between this component and its parent adapter.

Value is the adapter component that acts as a parent to this collection.

SetWebParent assigns Value to the protected Adapter (see page 5406) property and replaces that adapter's Data property with this TAdapterFields (see page 5265) instance.

See Also

Data (see page 5558)

Adapter (see page 5406)

57.1.18.3 TAdapterFields Properties

The properties of the TAdapterFields class are listed here.

Public Properties

	Name	Description
	WebFields (see page 5270)	Lists the field components that this object manages.

57.1.18.3.1 TAdapterFields.WebFields Property

Lists the field components that this object manages.

Pascal

```
property WebFields: TWebComponentList;
```

C++

```
__property TWebComponentList WebFields;
```

Description

Read (see page 3727) WebFields to access the list of field components. Each component in this list is a descendant of TAdapterNamedField (see page 5339).

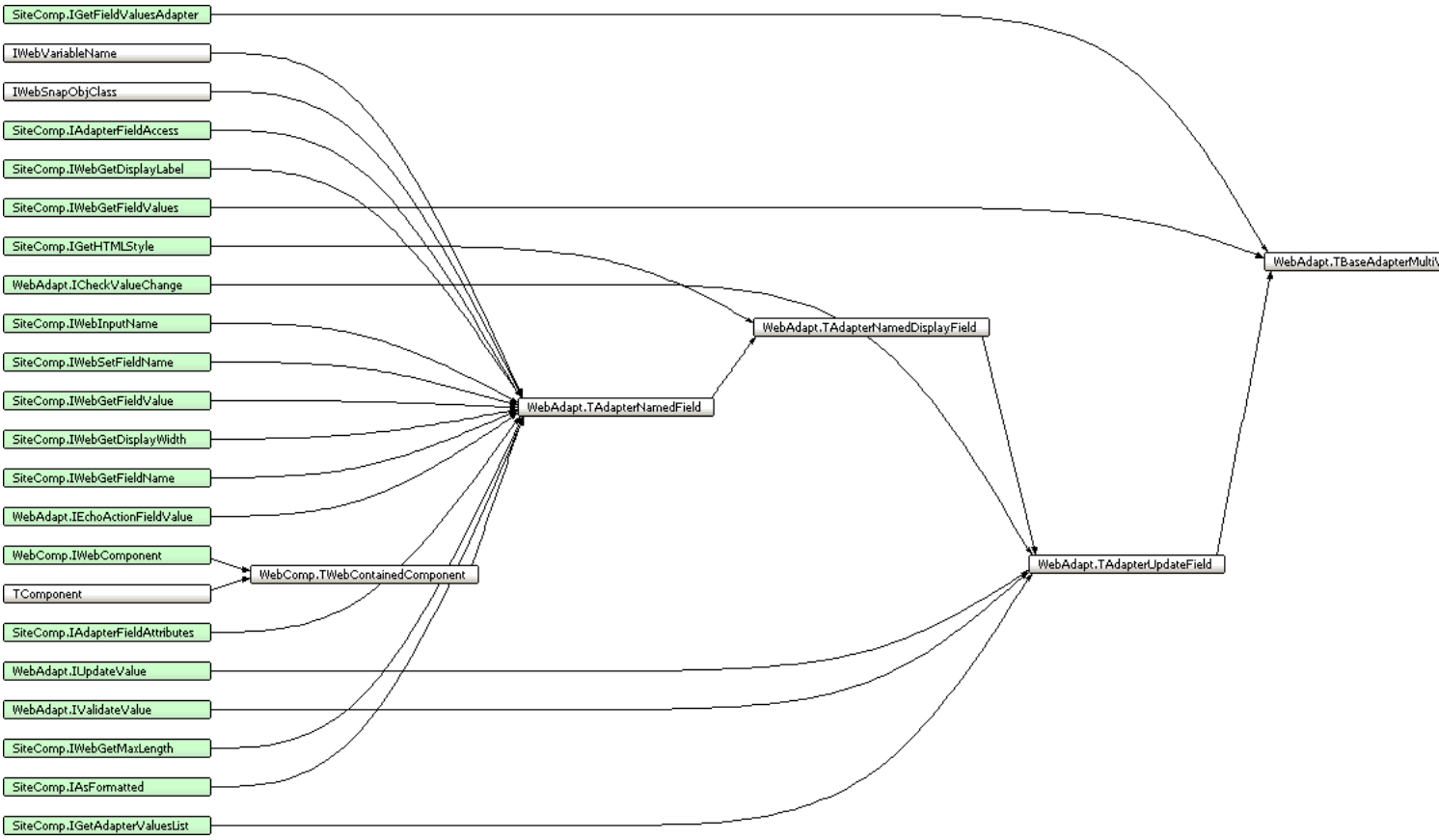
See Also

TAdapterNamedField (see page 5339)

57.1.19 TAdapterFileField Class

TAdapterFileField is an event handler based adapter field that can accept uploaded files.

Class Hierarchy



Pascal

```
TAdapterFileField = class(TCustomAdapterFileField);
```

C++

```
class TAdapterFileField : public TCustomAdapterFileField;
```

File

WebAdapt

Description

TAdapterFileField publishes the properties in TCustomAdapterFileField (see page 5590). TCustomAdapterFileField (see page 5590) calls event handlers to retrieve the adapter field's filenames, validate files uploaded by the end-user, and update the adapter field's files.

Add TAdapterFileField to TAdapter (see page 5179) or other adapter components types when you need an adapter field that supports files using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5256), TAdapterMultiValueField (see page 5320), TAdapterMemoField (see page 5312), TAdapterBooleanField (see page 5212) and TAdapterImageField (see page 5297).

See Also

TCustomAdapterFileField (see page 5590)

TAdapter (see page 5179)

TAdapterField (see page 5256)

TAdapterMemoField (see page 5312)

TAdapterMultiValueField (see page 5320)

TAdapterBooleanField (see page 5212)

TAdapterImageField (see page 5297)

57.1.19.1 TAdapterFileField Members

The following tables list the members exposed by TAdapterFileField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5595)	Validates files uploaded by the end-user.
	EndValueIterator (see page 5481)	Completes iteration.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5595)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
	GetFileNames (see page 5596)	Returns the count of files associated with this adapter field.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5596)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValueCount (see page 5482)	Returns the number of values associated with this adapter field.

	GetValueCount (see page 5596)	Returns the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetValues (see page 5482)	Returns one of the values associated with the adapter field.
	GetValues (see page 5597)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 5597)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5484)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5597)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.

	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 5480)	Constructor for TBaseAdapterMultiValueField (see page 5475).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 5480)	Destructor for TBaseAdapterMultiValueField (see page 5475).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.

	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FileCount (see page 5598)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 5598)	Returns a file associated with this adapter field.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetFileCount (see page 5278)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5279)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5279)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5279)	Validates values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5276)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5276)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5277)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5277)	Identifies the rights required to update this field.
	ReadOnly (see page 5277)	Indicates whether this adapter field is read-only.
	Required (see page 5278)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5278)	Identifies the rights required to view this field.

57.1.19.2 TAdapterFileField Properties

The properties of the TAdapterFileField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5276)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5276)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5277)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5277)	Identifies the rights required to update this field.
	ReadOnly (see page 5277)	Indicates whether this adapter field is read-only.
	Required (see page 5278)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5278)	Identifies the rights required to view this field.

57.1.19.2.1 TAdapterFileField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.19.2.2 TAdapterFileField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.19.2.3 TAdapterFileField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.19.2.4 TAdapterFileField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.19.2.5 TAdapterFileField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.19.2.6 TAdapterFileField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5347)

57.1.19.2.7 TAdapterFileField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.19.3 TAdapterFileField Events

The events of the TAdapterFileField class are listed here.

Published Events

	Name	Description
	OnGetFileCount (see page 5278)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5279)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5279)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5279)	Validates values entered by the end-user.

57.1.19.3.1 TAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 5596)

57.1.19.3.2 TAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 5596)

57.1.19.3.3 TAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue. Write an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5597)

TUpdateFileList (see page 5850)

57.1.19.3.4 TAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue. Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

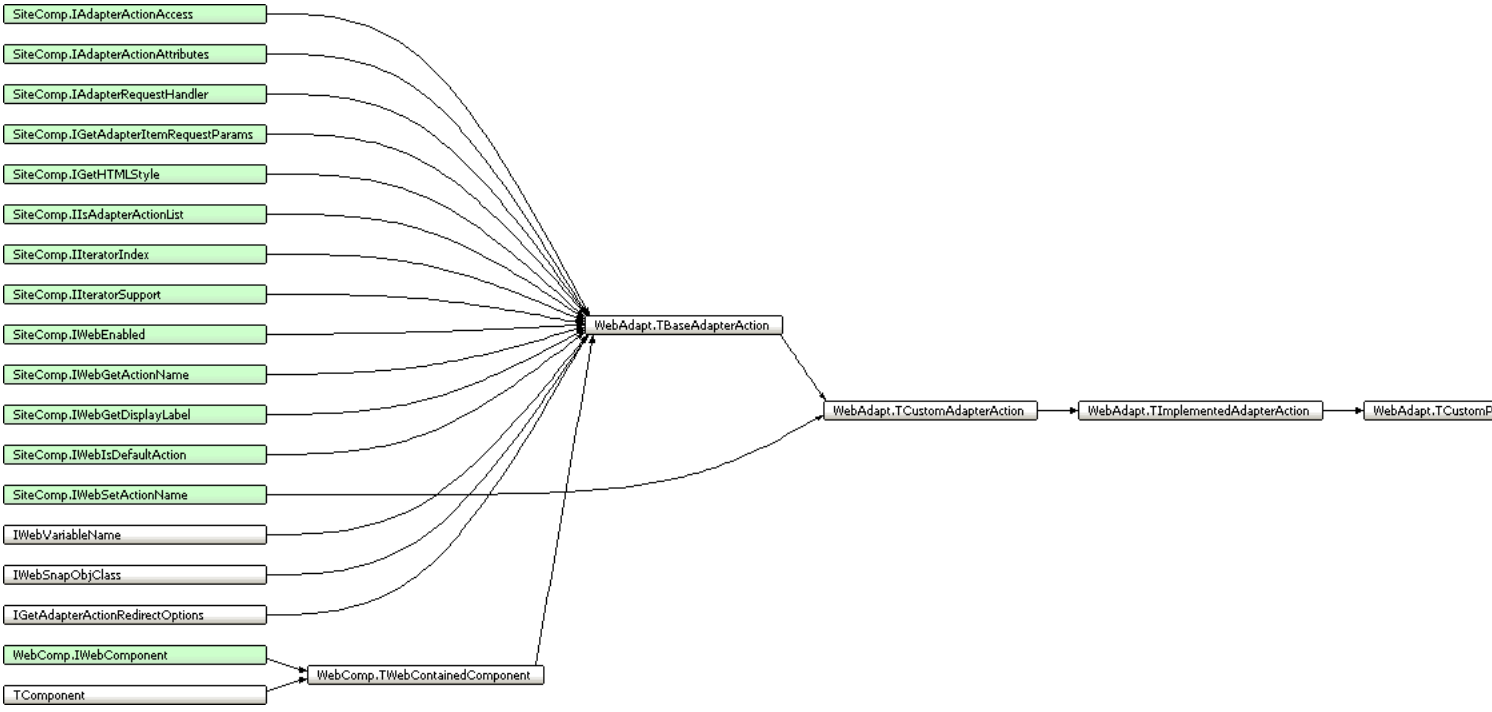
DoOnValidateValue (see page 5595)

TUpdateFileList (see page 5850)

57.1.20 TAdapterGotoPageAction Class

TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).

Class Hierarchy



Pascal

```
TAdapterGotoPageAction = class(TCustomAdapterGotoPageAction);
```

C++

```
class TAdapterGotoPageAction : public TCustomAdapterGotoPageAction;
```

File

WebAdapt

Description

TAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TAdapterGotoPageAction changes the current page in the parent adapter. TAdapterGotoPageAction publishes the properties of TCustomAdapterGotoPageAction (see page 5600). Add TAdapterGotoPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5733) such as TPagedAdapter (see page 5830).

See Also

TCustomAdapterGotoPageAction (see page 5600)

TDefaultFieldsPagedAdapter (see page 5733)

TPagedAdapter (see page 5830)

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

57.1.20.1 TAdapterGotoPageAction Members

The following tables list the members exposed by TAdapterGotoPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultActionName (see page 5605)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5605)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
V	GetItemCount (see page 5606)	Returns the number of commands defined by this adapter action.

	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5606)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5606)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetDisplayLabel (see page 5607)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIsAdapterActionList (see page 5607)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
V	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	ImplIsDefaultAction (see page 5608)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
V	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5608)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.

	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.
R	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
R	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5286)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5286)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5286)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5287)	Called before the adapter action response is generated.
	OnGetPageName (see page 5287)	Provides the display labels for each goto page command.
	OnGetParams (see page 5288)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ExecuteAccess (see page 5285)	Defines the rights required to execute this action.
	RedirectOptions (see page 5285)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.20.2 TAdapterGotoPageAction Properties

The properties of the TAdapterGotoPageAction class are listed here.

Published Properties

	Name	Description
	ExecuteAccess (see page 5285)	Defines the rights required to execute this action.
	RedirectOptions (see page 5285)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.20.2.1 TAdapterGotoPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6759)

hasexecuteaccess (see page 5539)

57.1.20.2.2 TAdapterGotoPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

57.1.20.3 TAdapterGotoPageAction Events

The events of the TAdapterGotoPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5286)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5286)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5286)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5287)	Called before the adapter action response is generated.
	OnGetPageName (see page 5287)	Provides the display labels for each goto page command.
	OnGetParams (see page 5288)	Specifies name/value pairs needed to execute this adapter action.

57.1.20.3.1 TAdapterGotoPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.20.3.2 TAdapterGotoPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.20.3.3 TAdapterGotoPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.20.3.4 TAdapterGotoPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.20.3.5 TAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

See Also

ImplGetDisplayLabel (see page 5607)

57.1.20.3.6 TAdapterGotoPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

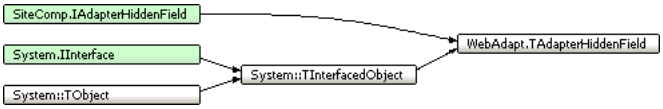
Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.21 TAdapterHiddenField Class

Class Hierarchy



Pascal

```
TAdapterHiddenField = class(TInterfacedObject, IAdapterHiddenField);
```

C++

```
class TAdapterHiddenField : public TInterfacedObject, public IAdapterHiddenField;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterHiddenField.

57.1.21.1 TAdapterHiddenField Members

The following tables list the members exposed by TAdapterHiddenField.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.

	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetName (see page 5290)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 5290)	This is GetValue, a member of class TAdapterHiddenField.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5290)	This is Create, a member of class TAdapterHiddenField.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Name (see page 945)	This is Name, a member of class IAdapterHiddenField.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

	Value (see page 945)	This is Value, a member of class IAdapterHiddenField.

57.1.21.2 TAdapterHiddenField.Create Constructor

Pascal

```
constructor Create(AHiddenFields: TAdapterHiddenFieldsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterHiddenField(TAdapterHiddenFieldsList AHiddenFields, int AIndex);
```

Description

This is Create, a member of class TAdapterHiddenField.

57.1.21.3 TAdapterHiddenField Methods

The methods of the TAdapterHiddenField class are listed here.

Protected Methods

	Name	Description
	GetName (see page 5290)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 5290)	This is GetValue, a member of class TAdapterHiddenField.

57.1.21.3.1 TAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall UnicodeString GetName();
```

Description

This is GetName, a member of class TAdapterHiddenField.

57.1.21.3.2 TAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall UnicodeString GetValue();
```

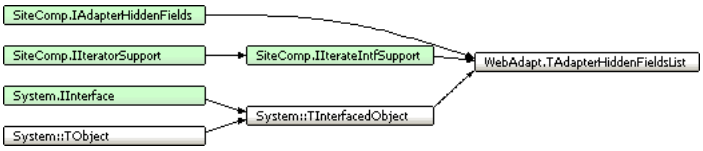
Description

This is GetValue, a member of class TAdapterHiddenField.

57.1.22 TAdapterHiddenFieldsList Class

TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.

Class Hierarchy



Pascal

```
TAdapterHiddenFieldsList = class(TInterfacedObject, IAdapterHiddenFields, IIterateIntfSupport);
```

C++

```
class TAdapterHiddenFieldsList : public TInterfacedObject, public IAdapterHiddenFields, public
IIterateIntfSupport;
```

File

WebAdapt

Description

Adapter components create instances of TAdapterHiddenFieldsList to store the name/value pairs they include as hidden fields in HTML forms. Each entry in the list has a field name and a value for that field. The adapter uses the methods of TAdapterHiddenFieldsList to add name/value pairs to the list, based on its hidden field options. When an adapter page producer generates a form using the adapter, it includes hidden fields for every name/value pair in the list. Those hidden fields are then included in any request that originates from the resulting HTML form, so that the adapter and its subcomponents can access those values to respond to the request.

For example, dataset adapters that have a Mode property that indicates whether the form is used for read-only browsing, inserting a new record, editing an existing record, or generating a query. The adapter includes the current mode as a hidden field. When the WebSnap application receives a message from the HTML form with that hidden field, the adapter can reestablish the correct mode for processing the request.

See Also

TCustomAdapter (see page 5519)

TCustomAdapter (see page 5519)

TCustomAdapter (see page 5519)

TCustomAdapter (see page 5519)

TAdapterPageProducer (see page 1076)

TCustomDataSetAdapter

57.1.22.1 TAdapterHiddenFieldsList Members

The following tables list the members exposed by TAdapterHiddenFieldsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.

	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	EndIterator (see page 5295)	This is EndIterator, a member of class TAdapterHiddenFieldsList.
	GetCurrentIntf (see page 5295)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.
	GetField (see page 5295)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 5295)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 5295)	Advances to the objects next item.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5296)	This is StartIterator, a member of class TAdapterHiddenFieldsList.

Protected Properties

	Name	Description
R	FieldCount (see page 5296)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
R	Fields (see page 5296)	This is Fields, a member of class TAdapterHiddenFieldsList.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	Add (see page 5294)	Adds a name/value pair to the list.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5294)	Clears the list of all the name/value pairs it currently holds.
	Create (see page 5293)	Creates and initializes a TAdapterHiddenFieldsList (see page 5290) object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 5293)	Frees an instance of TAdapterHiddenFieldsList (see page 5290).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.

	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

57.1.22.2 TAdapterHiddenFieldsList.Create Constructor

Creates and initializes a TAdapterHiddenFieldsList (see page 5290) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterHiddenFieldsList();
```

Description

Typically, applications have no need to create TAdapterHiddenFieldsList (see page 5290) instances. The adapter component creates and maintains its own internal TAdapterHiddenFieldsList (see page 5290) objects, which it uses to store the hidden fields it needs to add to HTML forms.

See Also

listhiddenfields (see page 5550)

ListHiddenRecordFields (see page 5550)

57.1.22.3 TAdapterHiddenFieldsList.Destroy Destructor

Frees an instance of TAdapterHiddenFieldsList (see page 5290).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterHiddenFieldsList();
```

Description

Do not use the Destroy method directly to free a TAdapterHiddenFieldsList (see page 5290) instance. Instead, use the Free method, which checks that the TAdapterHiddenFieldsList (see page 5290) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

57.1.22.4 TAdapterHiddenFieldsList Methods

The methods of the TAdapterHiddenFieldsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 5295)	This is EndIterator, a member of class TAdapterHiddenFieldsList.
	GetCurrentIntf (see page 5295)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.
	GetField (see page 5295)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 5295)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 5295)	Advances to the objects next item.
	StartIterator (see page 5296)	This is StartIterator, a member of class TAdapterHiddenFieldsList.

Public Methods

	Name	Description
	Add (see page 5294)	Adds a name/value pair to the list.
	Clear (see page 5294)	Clears the list of all the name/value pairs it currently holds.

57.1.22.4.1 TAdapterHiddenFieldsList.Add Method

Adds a name/value pair to the list.

Pascal

```
procedure Add(const AName: string; const AValue: string);
```

C++

```
__fastcall Add(const UnicodeString AName, const UnicodeString AValue);
```

Description

Call Add to add a new name/value pair to the list of hidden fields that the adapter maintains.

AName is the name of the field.

AValue is the value of the field.

See Also

Clear (see page 5294)

57.1.22.4.2 TAdapterHiddenFieldsList.Clear Method

Clears the list of all the name/value pairs it currently holds.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the TAdapterHiddenFieldsList (see page 5290) object. Clear discards any name/value pairs that have been added previously by a call to the Add (see page 5294) method.

See Also

Add (see page 5294)

57.1.22.4.3 TAdapterHiddenFieldsList.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TAdapterHiddenFieldsList.

57.1.22.4.4 TAdapterHiddenFieldsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.

57.1.22.4.5 TAdapterHiddenFieldsList.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class TAdapterHiddenFieldsList.

57.1.22.4.6 TAdapterHiddenFieldsList.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class TAdapterHiddenFieldsList.

57.1.22.4.7 TAdapterHiddenFieldsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 998)

enditerator (see page 997)

57.1.22.4.8 TAdapterHiddenFieldsList.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TAdapterHiddenFieldsList.

57.1.22.5 TAdapterHiddenFieldsList Properties

The properties of the TAdapterHiddenFieldsList class are listed here.

Protected Properties

	Name	Description
R	FieldCount (see page 5296)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
R	Fields (see page 5296)	This is Fields, a member of class TAdapterHiddenFieldsList.

57.1.22.5.1 TAdapterHiddenFieldsList.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class TAdapterHiddenFieldsList.

57.1.22.5.2 TAdapterHiddenFieldsList.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

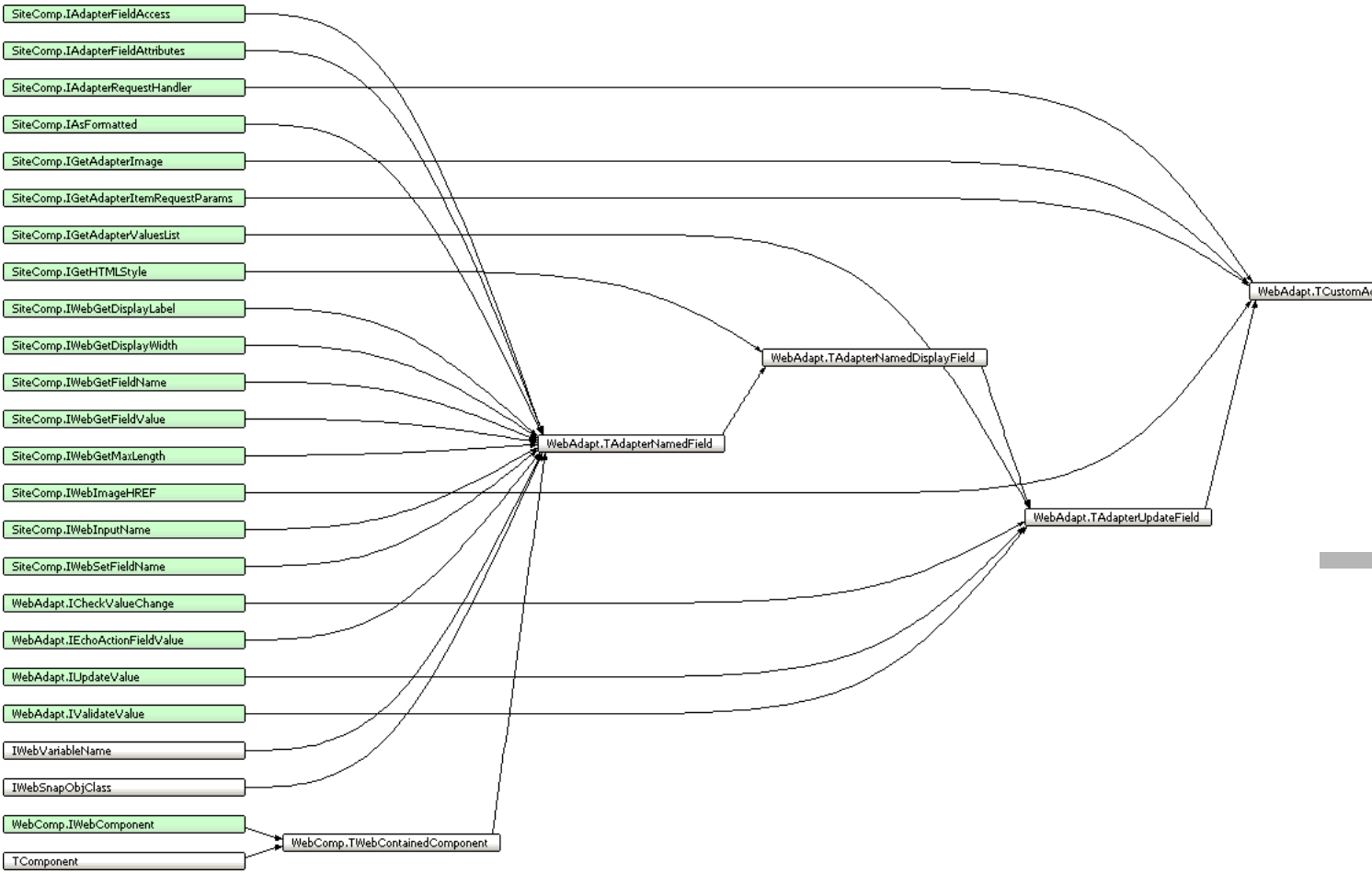
Description

This is Fields, a member of class TAdapterHiddenFieldsList.

57.1.23 TAdapterImageField Class

TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.

Class Hierarchy



Pascal

```
TAdapterImageField = class(TCustomAdapterImageField);
```

C++

```
class TAdapterImageField : public TCustomAdapterImageField;
```

File

WebAdapt

Description

TAdapterImageField publishes the properties in TCustomAdapterImageField (see page 5609). TCustomAdapterImageField (see page 5609) calls event handlers to retrieve the adapter field's image file name, image file, and image file URL. It also defines event handler that are called to validate files uploaded by the end-user, and update the adapter field's image file.

Add TAdapterFileField (see page 5270) to TAdapter (see page 5179) or other adapter components types when you need an adapter field that supports an image using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5256), TAdapterMultiValueField (see page 5320), TAdapterMemoField (see page 5312), TAdapterBooleanField (see page 5212) and TAdapterFileField (see page 5270).

See Also

TCustomAdapterImageField (see page 5609)

TAdapter (see page 5179)

TAdapterField (see page 5256)

TAdapterMemoField (see page 5312)

TAdapterMultiValueField (see page 5320)

TAdapterBooleanField (see page 5212)

TAdapterFileField (see page 5270)

57.1.23.1 TAdapterImageField Members

The following tables list the members exposed by TAdapterImageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 5614)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5614)	Validates files uploaded by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 5615)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 5615)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.

	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5616)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5616)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HandleRequest (see page 5616)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplCheckValueChange (see page 5617)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5617)	Creates AdapterRequest and AdapterResponse objects.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
V	ImplGetValue (see page 5618)	Returns the name of the image file associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.

	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 5618)	Handles a request for the image file associated with this adapter field.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5619)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 5619)	Returns a URL that can be used to access the image file associated with this adapter field.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 5619)	Responds to a request for an image file by providing a stream containing the image.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 5620)	Returns a URL that can be used to access the image file associated with this adapter field.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).

	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterImage (see page 965)	This is GetAdapterImage, a member of class IGetAdapterImage.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.
	WebImageHREF (see page 1040)	Retrieves a URL for an external image file.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetHREF (see page 5306)	Provides the URL of an external image file associated with this adapter field.

	OnGetImage (see page 5306)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5306)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5307)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5307)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5307)	Used to validate files uploaded by the end user.

Published Properties

	Name	Description
	DisplayLabel (see page 5302)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5303)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5303)	Identifies the name of this field when using server-side script.
	HREF (see page 5303)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 5304)	Identifies the rights required to update this field.
	ReadOnly (see page 5304)	Indicates whether this adapter field is read-only.
	Required (see page 5304)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5305)	Associates a list of values with the adapter field.
	ViewAccess (see page 5305)	Identifies the rights required to view this field.

57.1.23.2 TAdapterImageField Properties

The properties of the TAdapterImageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5302)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5303)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5303)	Identifies the name of this field when using server-side script.
	HREF (see page 5303)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 5304)	Identifies the rights required to update this field.
	ReadOnly (see page 5304)	Indicates whether this adapter field is read-only.
	Required (see page 5304)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5305)	Associates a list of values with the adapter field.
	ViewAccess (see page 5305)	Identifies the rights required to view this field.

57.1.23.2.1 TAdapterImageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.23.2.2 TAdapterImageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.23.2.3 TAdapterImageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.23.2.4 TAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property UnicodeString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF event handler.

See Also

ImplWebImageHREF (see page 5619)

OnGetHREF (see page 5621)

OnGetImage (see page 5622)

OnGetImageName (see page 5622)

57.1.23.2.5 TAdapterImageField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.23.2.6 TAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.23.2.7 TAdapterImageField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5347)

57.1.23.2.8 TAdapterImageField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1014)

TStringsValuesList (see page 5843)

TDataSetValuesList

57.1.23.2.9 TAdapterImageField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.23.3 TAdapterImageField Events

The events of the TAdapterImageField class are listed here.

Published Events

	Name	Description
	OnGetHREF (see page 5306)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5306)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5306)	Returns the name of the image file associated with this adapter field.

	OnGetParams (see page 5307)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5307)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5307)	Used to validate files uploaded by the end user.

57.1.23.3.1 TAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When OnGetHREF is called, the HREF parameter will contain the value of the HREF property. To dynamically generate an image file for this adapter file, implement the OnGetImage event handler instead.

See Also

ImplWebImageHREF (see page 5619)

HREF (see page 5620)

OnGetImage (see page 5622)

57.1.23.3.2 TAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF event handler or use the HREF property to specify the URL of an external image.

See Also

RenderAdapterImage (see page 5619)

OnGetParams (see page 5622)

OnGetHREF (see page 5621)

HREF (see page 5620)

57.1.23.3.3 TAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 5618)

57.1.23.3.4 TAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 5615)

OnGetImage (see page 5622)

57.1.23.3.5 TAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue. This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5619)

TAbstractWebRequestFile

57.1.23.3.6 TAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

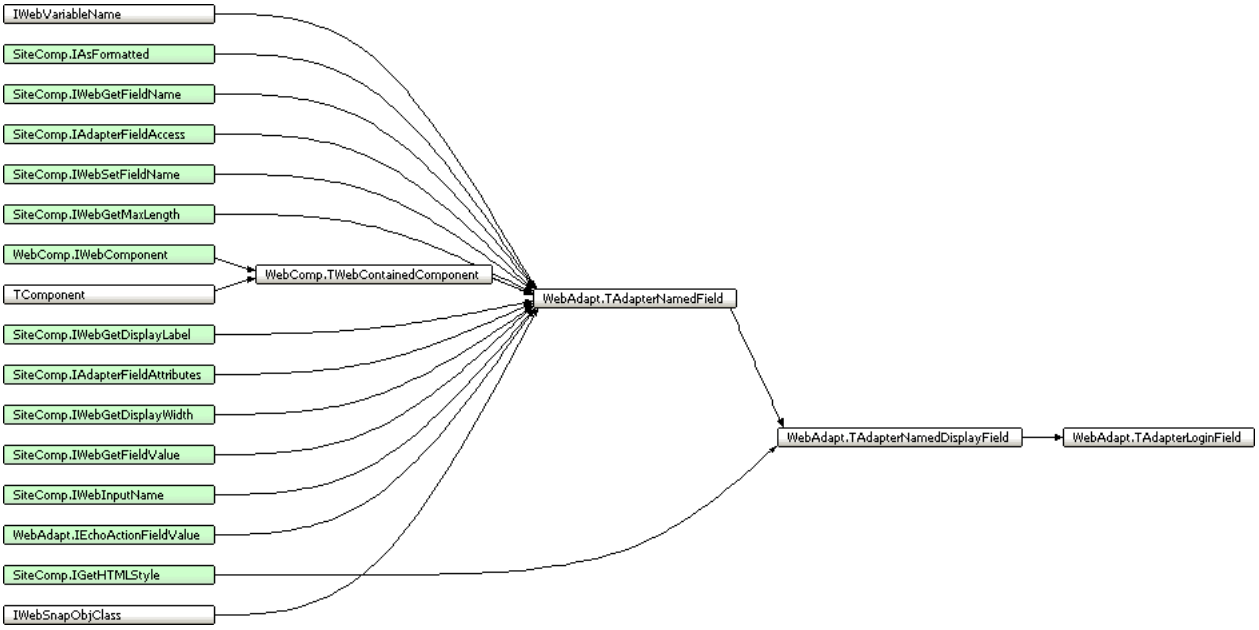
DoOnValidateValue (see page 5614)

TAbstractWebRequestFile

57.1.24 TAdapterLoginField Class

TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5810) component.

Class Hierarchy



Pascal

```
TAdapterLoginField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterLoginField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterLoginField implements functionality that is used by an adapter field that is a child of TLoginFormAdapter (see page 5810).

See Also

TLoginFormAdapter (see page 5810)

TAdapterNamedDisplayField (see page 5331)

TAdapterUserNameField (see page 5422)

TAdapterPasswordField (see page 5379)

57.1.24.1 TAdapterLoginField Members

The following tables list the members exposed by TAdapterLoginField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.

	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.

V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 5311)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.24.2 TAdapterLoginField Properties

The properties of the TAdapterLoginField class are listed here.

Public Properties

	Name	Description
R	Adapter (see page 5311)	Returns the parent adapter of this adapter field.

57.1.24.2.1 TAdapterLoginField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter. GetAdapter (see page 5344) implements this property.

See Also

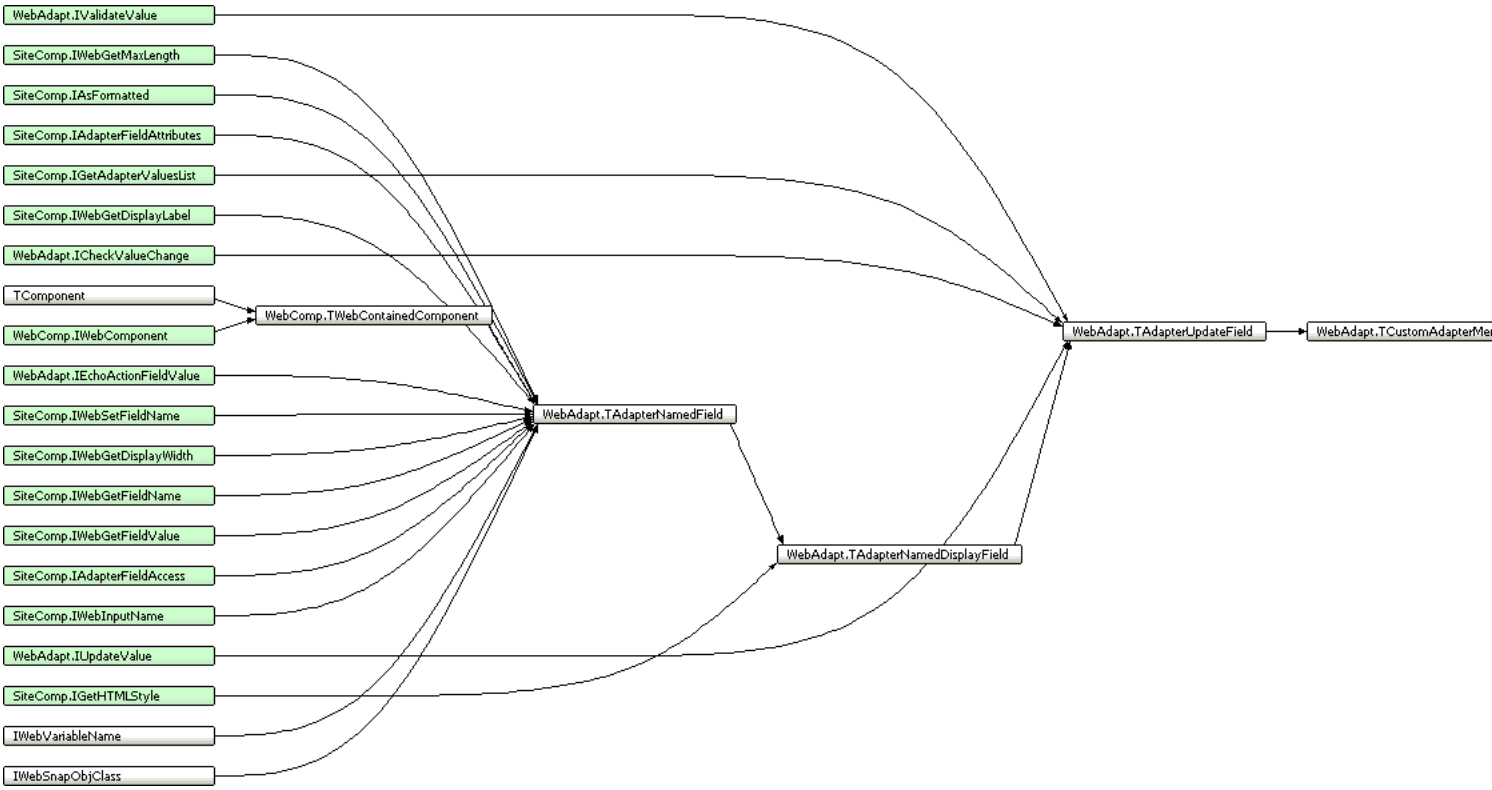
TCustomLoginFormAdapter (see page 5687)

GetAdapter

57.1.25 TAdapterMemoField Class

TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.

Class Hierarchy



Pascal

```
TAdapterMemoField = class(TCustomAdapterMemoField);
```

C++

```
class TAdapterMemoField : public TCustomAdapterMemoField;
```

File

WebAdapt

Description

TAdapterMemoField publishes the properties in TCustomAdapterMemoField (see page 5624). TCustomAdapterMemoField (see page 5624) calls event handlers to retrieve the adapter field's text value, validate a value entered end-user, and update the

adapter field's value.

Add TAdapterMemoField to TAdapter (see page 5179) or other adapter components types when you need an adapter field that supports a multi-line text value. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5256), TAdapterMultiValueField (see page 5320), TAdapterFileField (see page 5270), TAdapterBooleanField (see page 5212) and TAdapterImageField (see page 5297).

See Also

TCustomAdapterMemoField (see page 5624)

TAdapter (see page 5179)

TAdapterField (see page 5256)

TAdapterFileField (see page 5270)

TAdapterMultiValueField (see page 5320)

TAdapterBooleanField (see page 5212)

TAdapterImageField (see page 5297)

57.1.25.1 TAdapterMemoField Members

The following tables list the members exposed by TAdapterMemoField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5628)	Validates a value entered by the end user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5628)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.

	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetInputStyleType (see page 5629)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 5629)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetValue (see page 5630)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5630)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.

	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.

	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 5319)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5319)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5320)	Called to update the adapter field's value with a file uploaded by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5317)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5317)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5317)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5318)	Identifies the rights required to update this field.
	ReadOnly (see page 5318)	Indicates whether this adapter field is read-only.
	Required (see page 5318)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5319)	Identifies the rights required to view this field.

57.1.25.2 TAdapterMemoField Properties

The properties of the TAdapterMemoField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5317)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5317)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5317)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5318)	Identifies the rights required to update this field.
	ReadOnly (see page 5318)	Indicates whether this adapter field is read-only.

	Required (see page 5318)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5319)	Identifies the rights required to view this field.

57.1.25.2.1 TAdapterMemoField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.25.2.2 TAdapterMemoField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.25.2.3 TAdapterMemoField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.25.2.4 TAdapterMemoField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.25.2.5 TAdapterMemoField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.25.2.6 TAdapterMemoField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5347)

57.1.25.2.7 TAdapterMemoField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.25.3 TAdapterMemoField Events

The events of the TAdapterMemoField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 5319)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5319)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5320)	Called to update the adapter field's value with a file uploaded by the end-user.

57.1.25.3.1 TAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5630)

57.1.25.3.2 TAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5630)

57.1.25.3.3 TAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

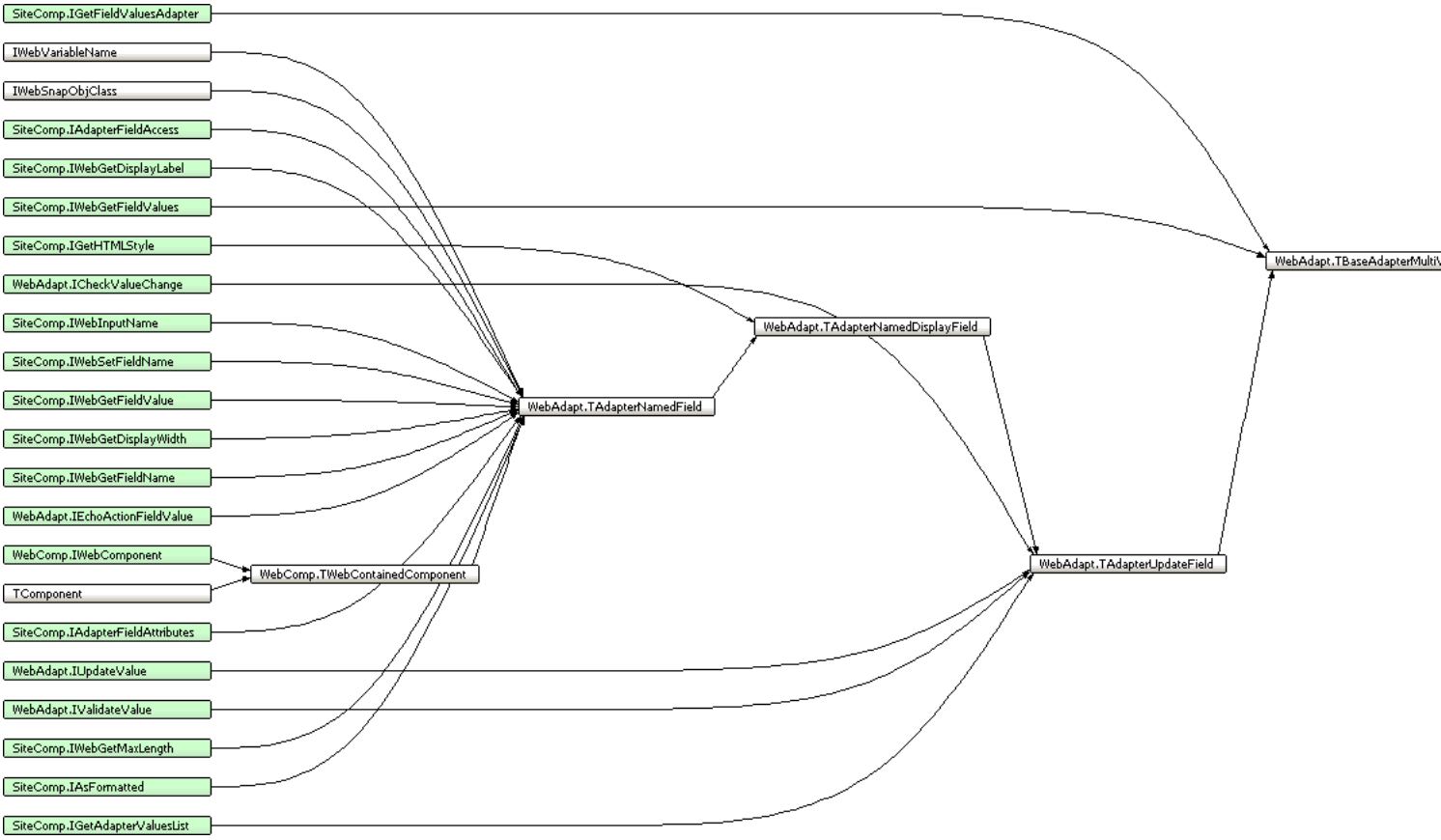
See Also

ImplUpdateValue (see page 5630)

57.1.26 TAdapterMultiValueField Class

TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.

Class Hierarchy



Pascal

```
TAdapterMultiValueField = class(TCustomAdapterMultiValueField);
```

C++

```
class TAdapterMultiValueField : public TCustomAdapterMultiValueField;
```

File

WebAdapt

Description

TAdapterMultiValueField publishes the properties in TCustomAdapterMultiValueField (see page 5632). TCustomAdapterMultiValueField (see page 5632) calls event handlers to retrieve the adapter field's values, validate values entered by the end-user, and update the adapter field's values.

Add TAdapterMultiValueField to TAdapter (see page 5179) or other adapter components types when you need an adapter field that supports a multiple values using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5256), TAdapterFileField (see page 5270), TAdapterMemoField (see page 5312), TAdapterBooleanField (see page 5212) and TAdapterImageField (see page 5297).

See Also

TCustomAdapterMultiValueField (see page 5632)

TAdapter (see page 5179)

TAdapterField (see page 5256)

TAdapterMemoField (see page 5312)

TAdapterFileField (see page 5270)

TAdapterBooleanField (see page 5212)

TAdapterImageField (see page 5297)

57.1.26.1 TAdapterMultiValueField Members

The following tables list the members exposed by TAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5637)	Validates values entered by the end-user.
	EndValueIterator (see page 5481)	Completes iteration.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5637)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5637)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.

	GetValueCount (see page 5482)	Returns the number of values associated with this adapter field.
	GetValueCount (see page 5638)	Returns the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetValues (see page 5482)	Returns one of the values associated with the adapter field.
	GetValues (see page 5638)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5484)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	ImplHasValue (see page 5639)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5639)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 5480)	Constructor for TBaseAdapterMultiValueField (see page 5475).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 5480)	Destructor for TBaseAdapterMultiValueField (see page 5475).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.

	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 5639)	Returns the number of values associated with this adapter field.
R	Values (see page 5640)	Returns one of the values associated with the adapter field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValueCount (see page 5329)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5329)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5330)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5330)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5330)	Validates field values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5326)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5326)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5327)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5327)	Identifies the rights required to update this field.
	ReadOnly (see page 5327)	Indicates whether this adapter field is read-only.
	Required (see page 5328)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5328)	Associates a list of values with the adapter field.

	ViewAccess (see page 5328)	Identifies the rights required to view this field.

57.1.26.2 TAdapterMultiValueField Properties

The properties of the TAdapterMultiValueField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5326)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5326)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5327)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5327)	Identifies the rights required to update this field.
	ReadOnly (see page 5327)	Indicates whether this adapter field is read-only.
	Required (see page 5328)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5328)	Associates a list of values with the adapter field.
	ViewAccess (see page 5328)	Identifies the rights required to view this field.

57.1.26.2.1 TAdapterMultiValueField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.26.2.2 TAdapterMultiValueField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.26.2.3 TAdapterMultiValueField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.26.2.4 TAdapterMultiValueField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.26.2.5 TAdapterMultiValueField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.26.2.6 TAdapterMultiValueField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5347)

57.1.26.2.7 TAdapterMultiValueField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1014)

TStringsValuesList (see page 5843)

TDataSetValuesList

57.1.26.2.8 TAdapterMultiValueField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.26.3 TAdapterMultiValueField Events

The events of the TAdapterMultiValueField class are listed here.

Published Events

	Name	Description
	OnGetValueCount (see page 5329)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5329)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5330)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5330)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5330)	Validates field values entered by the end-user.

57.1.26.3.1 TAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues to provide the individual values.

See Also

GetValueCount (see page 5638)

OnGetValues (see page 5641)

57.1.26.3.2 TAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount to provide the count of values associated with this adapter field.

See Also

GetValues (see page 5638)

OnGetValueCount (see page 5640)

57.1.26.3.3 TAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue will search for a matching value the Values and ValueCount properties.

See Also

ImplHasValue (see page 5639)

ValueCount (see page 5639)

Values (see page 5640)

57.1.26.3.4 TAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 5642)

ImplUpdateValue (see page 5639)

57.1.26.3.5 TAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

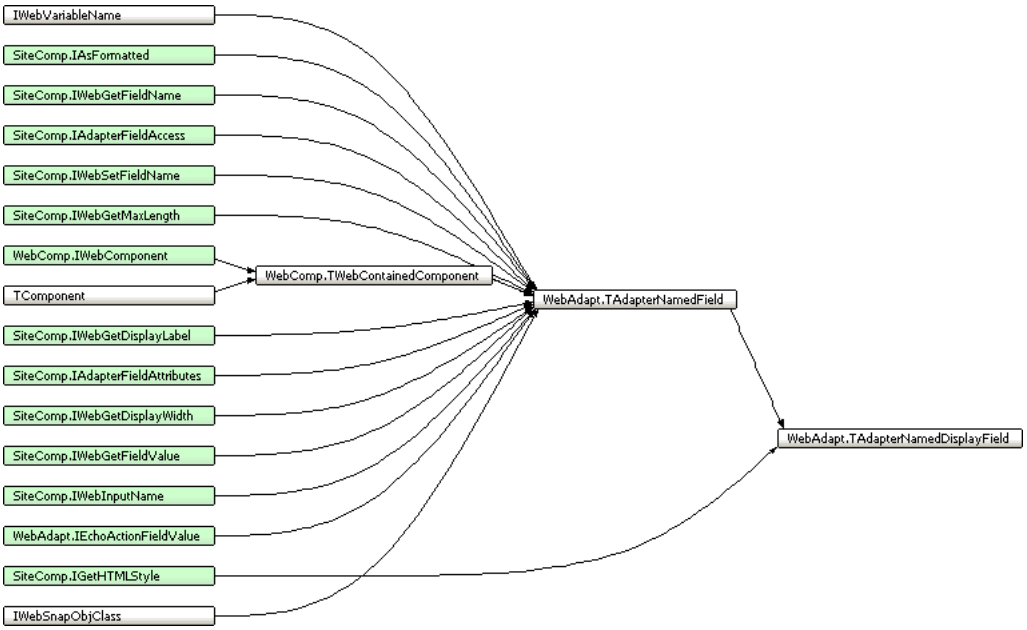
See Also

OnValidateValues (see page 5642)

57.1.27 TAdapterNamedDisplayField Class

The AdapterNamedDisplayField is a base class for adapter fields.

Class Hierarchy



Pascal

```
TAdapterNamedDisplayField = class(TAdapterNamedField, IGetHTMLStyle);
```

C++

```
class TAdapterNamedDisplayField : public TAdapterNamedField, public IGetHTMLStyle;
```

File

WebAdapt

Description

TAdapterNameDisplayField implements the IGetHTMLStyle (see page 973) interface used to inquire how this adapter field should be displayed in an HTML form. For example, IGetHTMLStyle (see page 973)'s GetInputStyle (see page 5335) method may indicate that this field should be displayed using a select list. TAdapterNamedField (see page 5339) also declares properties that may be modified at design-time to control how the adapter field should be displayed. For example, the DisplayLabel (see page 5338) property is used to suggest display label for this adapter field.

See Also

TAdapterNamedField (see page 5339)

TCustomAdapter (see page 5519)

TAdapter (see page 5179)

IGetHTMLStyle (see page 973)

57.1.27.1 TAdapterNamedDisplayField Members

The following tables list the members exposed by TAdapterNamedDisplayField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.

	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.

	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.27.2 TAdapterNamedDisplayField.Create Constructor

Creates an instance of TAdapterNamedDisplayField (see page 5331).

Pascal

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

C++

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

Description

Do not call Create. It will be called automatically when the TAdapterNamedDisplayField (see page 5331) object is created

57.1.27.3 TAdapterNamedDisplayField Methods

The methods of the TAdapterNamedDisplayField class are listed here.

Protected Methods

	Name	Description
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.

57.1.27.3.1 TAdapterNamedDisplayField.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter field.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle (see page 973)'s GetDisplayStyle by translating the result value of GetDisplayStyleType (see page 5335) into a string.

See Also

GetDisplayStyle (see page 974)

GetDisplayStyleType (see page 5335)

57.1.27.3.2 TAdapterNamedDisplayField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmldText, indicating that this adapter field should be displayed by simple text. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 5334)

57.1.27.3.3 TAdapterNamedDisplayField.GetInputStyle Method

Returns a string value to suggest an HTML input style for this adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle (see page 973)'s GetInputType method by translating the result value of GetInputStyleType (see page 5335) into a string.

See Also

GetInputStyle (see page 974)

GetInputStyleType (see page 5335)

57.1.27.3.4 TAdapterNamedDisplayField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; virtual;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType returns htmliTextInput, indicating that this adapter field should be edited by a single line input field. Descendant classes may override this method to suggest a different HTML input type.

See Also

GetInputStyle (see page 5335)

57.1.27.3.5 TAdapterNamedDisplayField.GetViewMode Method

Returns a string value to suggest a view mode for this adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle (see page 973)'s GetViewMode method by translating the result value of GetViewModeType (see page 5336) into a string.

See Also

GetViewMode (see page 975)

GetViewModeType (see page 5336)

57.1.27.3.6 TAdapterNamedDisplayField.GetViewModeType Method

Suggests a view mode for this adapter field.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const UnicodeString
AAdapterMode);
```

Description

GetViewModeType returns htmlvmDisplay if the ReadOnly (see page 5339) property is true. Otherwise, htmlvmInput is returned.

See Also

ReadOnly (see page 5339)

GetViewMode (see page 5336)

57.1.27.3.7 TAdapterNamedDisplayField.ImplGetDisplayLabel Method

Implements interface access to the DisplayLabel (see page 5338) property.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5338) property.

See Also

DisplayLabel (see page 5338)

57.1.27.3.8 TAdapterNamedDisplayField.ImplGetDisplayWidth Method

Implements interface access to the DisplayWidth (see page 5338) property.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 5338) property.

See Also

DisplayWidth (see page 5338)

57.1.27.3.9 TAdapterNamedDisplayField.ImplGetRequired Method

Implements interface access to the Required (see page 5339) property.

Pascal

```
function ImplGetRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns the value of the Required (see page 5339) property.

See Also

Required (see page 5339)

57.1.27.3.10 TAdapterNamedDisplayField.SetDisplayLabel Method

SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.

Pascal

```
procedure SetDisplayLabel(const AValue: string);
```

C++

```
__fastcall SetDisplayLabel(const UnicodeString AValue);
```

Description

Use SetDisplayLabel to set the value of the DisplayLabel (see page 5338) property to AValue.

See Also

DisplayLabel (see page 5338)

57.1.27.4 TAdapterNamedDisplayField Properties

The properties of the TAdapterNamedDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.

57.1.27.4.1 TAdapterNamedDisplayField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.27.4.2 TAdapterNamedDisplayField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this

adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.27.4.3 TAdapterNamedDisplayField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode (see page 5336) which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.27.4.4 TAdapterNamedDisplayField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

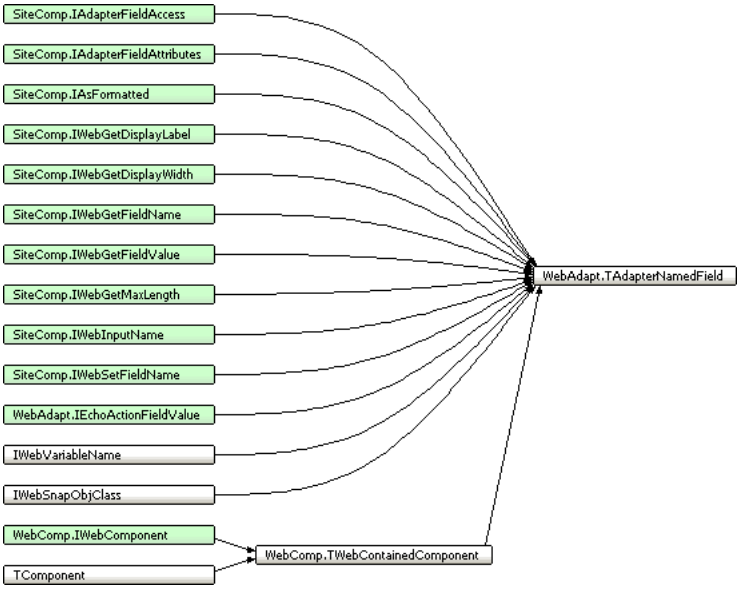
See Also

GetRequired (see page 5347)

57.1.28 TAdapterNamedField Class

TAdapterNamedField is a based class for adapter field components.

Class Hierarchy



Pascal

```
TAdapterNamedField = class(TWebContainedComponent, IWebVariableName, IWebInputName,
IWebGetFieldValue, IEchoActionFieldValue, IAsFormatted, IWebGetFieldName, IWebSetFieldName,
IWebGetDisplayWidth, IWebGetDisplayLabel, IWebGetMaxLength, IWebSnapObjClass,
IAdapterFieldAccess, IAdapterFieldAttributes);
```

C++

```
class TAdapterNamedField : public TWebContainedComponent, public IWebVariableName, public
IWebInputName, public IWebGetFieldValue, public IEchoActionFieldValue, public IAsFormatted,
public IWebGetFieldName, public IWebSetFieldName, public IWebGetDisplayWidth, public
IWebGetDisplayLabel, public IWebGetMaxLength, public IWebSnapObjClass, public
IAdapterFieldAccess, public IAdapterFieldAttributes;
```

File

WebAdapt

Description

TAdapterNamedField implements interfaces used to access an adapter field's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter field class.

See Also

TAdapterNamedDisplayField (see page 5331)

TAdapter (see page 5179)

57.1.28.1 TAdapterNamedField Members

The following tables list the members exposed by TAdapterNamedField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
R	Adapter (see page 5356)	Returns the parent adapter component.

R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.28.2 TAdapterNamedField Methods

The methods of the TAdapterNamedField class are listed here.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.

	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.

57.1.28.2.1 TAdapterNamedField.AsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function AsFormatted: string;
```

C++

```
__fastcall UnicodeString AsFormatted();
```

Description

This function returns a text that is typically used when generating a read-only representation of the adapter field value. The value returned by this function can be accessed with server-side script using the DisplayText property.

See Also

ImplAsFormatted (see page 5349)

57.1.28.2.2 TAdapterNamedField.CheckModifyAccess Method

Checks end-user rights to modify the adapter field value.

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This procedure is called to verify that the current user has rights to modify this field. An exception will be raised if access is denied.

See Also

ModifyAccess (see page 5357)

HasModifyAccess (see page 5349)

57.1.28.2.3 TAdapterNamedField.CheckViewAccess Method

Checks end-user rights to view the adapter field value.

Pascal

```
procedure CheckViewAccess;
```

C++

```
__fastcall CheckViewAccess();
```

Description

This procedure is called to verify that the current user has rights to view this field. An exception will be raised if access is denied.

See Also

ViewAccess (see page 5358)

HasModifyAccess (see page 5349)

57.1.28.2.4 TAdapterNamedField.FullyQualifyInputName Method

Indicates whether the HTML input names must be qualified with the name of the adapter.

Pascal

```
function FullyQualifyInputName: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FullyQualifyInputName();
```

Description

This functions returns true if, when generating an HTML input referencing this adapter field, the name of the HTML input must be qualified with the name of the adapter component. Input (see page 3895) names must be fully qualified when a single HTML form is used to input values for multiple adapters.

See Also

MasterAdapter

57.1.28.2.5 TAdapterNamedField.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

57.1.28.2.6 TAdapterNamedField.GetDefaultDisplayLabel Method

Provides the default display label for this adapter field.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

When the display label for the field matches this value then the display label property does not need to be stored in the form file.

57.1.28.2.7 TAdapterNamedField.GetDefaultFieldName Method

Gets the default name used to reference this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

This function returns the default field name for the field. This name will be used when a field name has not be set using ImplSetFieldName (see page 5354). GetDefaultFieldName returns the name of the adapter field component. Descendants may override this function.

See Also

ImplGetVariableName (see page 5352)

ImplSetFieldName (see page 5354)

57.1.28.2.8 TAdapterNamedField.GetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall UnicodeString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display the value of this adapter field. The value

returned by this function can be accessed with server-side script using the DisplayLabel property. GetDisplayLabel calls ImplGetDisplayLabel (see page 5349).

See Also

ImplGetDisplayLabel (see page 5349)

57.1.28.2.9 TAdapterNamedField.GetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

This function returns the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. The value returned by this function can be accessed with server-side script using the DisplayWidth property. GetDisplayWidth calls ImplGetDisplayWidth (see page 5350).

See Also

ImplGetDisplayWidth (see page 5350)

57.1.28.2.10 TAdapterNamedField.GetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

EchoActionFieldValue (see page 5356) calls ImplEchoActionFieldValue.

See Also

EchoActionFieldValue (see page 5356)

57.1.28.2.11 TAdapterNamedField.GetInputName Method

Identifies the name to use when generating an HTML input for this adapter field.

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall UnicodeString GetInputName();
```

Description

This functions returns the name used for an HTML input associated with this adapter field. This value is accessed with server-side script using the InputName property. GetInputName calls ImplGetInputName (see page 5350).

See Also

ImplGetInputName (see page 5350)

57.1.28.2.12 TAdapterNamedField.GetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function GetMaxLength: Integer;
```

C++

```
__fastcall int GetMaxLength();
```

Description

The value returned by this function can be accessed with server-side script using the MaxLength property. Server-side script may use this value to define HTML elements that limit the number of characters that can be entered by the end-user. GetMaxLength calls ImplGetMaxLength (see page 5351).

See Also

ImplGetMaxLength (see page 5351)

57.1.28.2.13 TAdapterNamedField.GetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This function returns true or false to indicate whether a value must be entered by the end user before submitting a form. The value returned by this function can be accessed with server-side script using the Required property. Typically, this function is called while executing an adapter action that validates field values. The adapter action component will raise an exception if the value of the submitted field is empty. GetRequired calls ImplGetRequired (see page 5351).

See Also

ImplGetRequired (see page 5351)

57.1.28.2.14 TAdapterNamedField.GetValue Method

Gets the value of this adapter field as a Variant (see page 3430).

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

GetValue returns the result of ImplGetValue (see page 5351).

See Also

Value (see page 5357)

ImplGetValue (see page 5351)

57.1.28.2.15 TAdapterNamedField.GetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter field with server-side script. GetVariableName calls ImplGetVariableName (see page 5352).

See Also

ImplGetVariableName (see page 5352)

57.1.28.2.16 TAdapterNamedField.GetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The value returned by this function can be accessed with server-side script using the Visible property. GetVisible calls ImplGetVisible (see page 5352).

See Also

ImplGetVisible (see page 5352)

57.1.28.2.17 TAdapterNamedField.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

57.1.28.2.18 TAdapterNamedField.HasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanModify property. HasModifyAccess calls ImplHasModifyAccess (see page 5353).

See Also

ImplHasModifyAccess (see page 5353)

57.1.28.2.19 TAdapterNamedField.HasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 5353).

See Also

ImplHasViewAccess (see page 5353)

57.1.28.2.20 TAdapterNamedField.ImplAsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function ImplAsFormatted: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted is called by AsFormatted (see page 5343).

See Also

AsFormatted (see page 5343)

57.1.28.2.21 TAdapterNamedField.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel is called by GetDisplayLabel (see page 5345). ImplGetDisplayLabel returns the result of GetDefaultDisplayLabel (see page 5345). Descendants may override this function.

See Also

GetDisplayLabel (see page 5345)

GetDefaultDisplayLabel (see page 5345)

57.1.28.2.22 TAdapterNamedField.ImplGetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function ImplGetDisplayWidth: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

This function is called by GetDisplayWidth (see page 5346) to retrieve the suggested display width for this adapter field. ImplGetDisplayWidth returns –1 to indicate an undefined display width. Descendants may override this function.

See Also

GetDisplayWidth (see page 5346)

57.1.28.2.23 TAdapterNamedField.ImplGetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

This function is called by GetEchoActionFieldValue (see page 5346).

See Also

EchoActionFieldValue (see page 5356)

GetEchoActionFieldValue (see page 5346)

57.1.28.2.24 TAdapterNamedField.ImplGetInputName Method

Identifies the name to use when generating the HTML input for this adapter field.

Pascal

```
function ImplGetInputName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetInputName();
```

Description

When an action request is received, the name/value pairs in the HTTP are searched for adapter field values. For each adapter field value found, an IActionFieldValue (see page 5890) implementation is created. In order for the HTTP request to be interpreted properly, the HTML input fields must have names that correspond to the adapter fields they represent.

See Also

GetInputName (see page 5346)

IActionFieldValue

57.1.28.2.25 TAdapterNamedField.ImplGetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function ImplGetMaxLength: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetMaxLength();
```

Description

ImplGetMaxLength is called by GetMaxLength (see page 5347). ImplGetMaxLength returns –1 to indicate an undefined maximum length. Descendants may override this function.

See Also

GetMaxLength (see page 5347)

57.1.28.2.26 TAdapterNamedField.ImplGetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function ImplGetRequired: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns False. Descendants may override this function.

See Also

GetRequired (see page 5347)

57.1.28.2.27 TAdapterNamedField.ImplGetValue Method

Gets the value of this adapter field as a Variant (see page 3430).

Pascal

```
function ImplGetValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

This function is called by GetValue (see page 5347). ImplGetValue returns Unassigned. Descendants override this function.

See Also

Value (see page 5357)

GetValue (see page 5347)

57.1.28.2.28 TAdapterNamedField.ImplGetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 5348). GetVariableName (see page 5348) returns the field name set by ImplSetFieldName (see page 5354) unless this name is blank. If the field name is blank, the result of GetDefaultFieldName (see page 5345) is returned.

See Also

GetVariableName (see page 5348)

ImplSetFieldName (see page 5354)

GetDefaultFieldName (see page 5345)

57.1.28.2.29 TAdapterNamedField.ImplGetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

This function returns true to indicate that the field should be displayed to the end user. Descendants may override this method.

See Also

GetVisible (see page 5348)

57.1.28.2.30 TAdapterNamedField.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 5349) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 5357) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 5357) property value.

See Also

hasmodifyaccess (see page 5539)

ModifyAccess (see page 5357)

HasModifyAccess (see page 5349)

57.1.28.2.31 TAdapterNamedField.ImplHasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This function is called by HasViewAccess (see page 5349) to check the end-user's right to view the adapter field. If the ViewAccess (see page 5358) property is not blank, this function calls the EndUserAdapter's HasViewRights function, passing the ViewAccess (see page 5358) property value.

See Also

HasViewAccess (see page 5539)

ViewAccess (see page 5358)

HasViewAccess (see page 5349)

57.1.28.2.32 TAdapterNamedField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplEchoActionFieldValue is called by SetEchoActionFieldValue (see page 5354). ImplEchoActionFieldValue is a placeholder. Descendants may override this method.

See Also

EchoActionFieldValue (see page 5356)

SetEchoActionFieldValue (see page 5354)

57.1.28.2.33 TAdapterNamedField.ImplSetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); virtual;
```

C++

```
virtual __fastcall ImplSetFieldName(const UnicodeString Value);
```

Description

ImplSetFieldName is called by SetFieldName (see page 5355). ImplSetFieldName stores the name in an instance variable. This method may be overridden by descendants.

See Also

SetFieldName (see page 5355)

57.1.28.2.34 TAdapterNamedField.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TAdapterNamedField (see page 5339) clears its cached reference to the parent adapter.

57.1.28.2.35 TAdapterNamedField.SetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

SetEchoActionFieldValue calls ImplSetActionFieldValue.

See Also

EchoActionFieldValue (see page 5356)

ImplSetEchoActionFieldValue (see page 5353)

57.1.28.2.36 TAdapterNamedField.SetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure SetFieldName(const AName: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString AName);
```

Description

SetFieldName is typically called at design time to set the field name of a new adapter field. SetFieldName calls ImplSetFieldName (see page 5354).

See Also

ImplSetFieldName (see page 5354)

57.1.28.3 TAdapterNamedField Properties

The properties of the TAdapterNamedField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5356)	Returns the parent adapter component.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.

57.1.28.3.1 TAdapterNamedField.ActionOrigValue Property

Retrieve the original value of this adapter field.

Pascal

```
property ActionOrigValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionOrigValue;
```

Description

Some adapter fields will include their original value as a hidden field in an HTML form. The original value can be compared with

the current value to verify that the current value hasn't been changed by another user between the time that the HTML form was generated and the HTML form was submitted. Use this property when executing an adapter action to retrieve the original value for this adapter field at the time that the HTML form was generated.

See Also

FieldFlags

57.1.28.3.2 TAdapterNamedField.ActionValue Property

Retrieve the value for this adapter field entered by the end-user.

Pascal

```
property ActionValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionValue;
```

Description

The ActionValue property retrieves the value(s) entered by the end-user and passed in the HTTP request when an HTML form is submitted. When executing an adapter action, use this property to retrieve the value(s) entered by the end-user.

57.1.28.3.3 TAdapterNamedField.Adapter Property

Returns the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Use this property to access the parent adapter component. The type of this class is TComponent because adapter components are not required to descend from a common adapter base class. Typically the adapter component is accessed through interfaces.

57.1.28.3.4 TAdapterNamedField.EchoActionFieldValue Property

Used to echo the value(s) entered by the end-user when generating an HTML form.

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

```
__property Boolean EchoActionFieldValue;
```

Description

The EchoActionFieldValue property is used to alter the source of the adapter fields value. When this flag is set, the adapter field retrieves it's value from the HTTP request. When this flag is clear, the adapter field retrieves it's current value. Typically, this flag is set after an error occurs while processing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValue (see page 5354)

57.1.28.3.5 TAdapterNamedField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.28.3.6 TAdapterNamedField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6759)

hasmodifyaccess (see page 5539)

57.1.28.3.7 TAdapterNamedField.Value Property

Provides the value of an adapter field as a variant.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

The Value property returns the value of an adapter field as a Variant (see page 3430). Server-side script can access this value using the Value property. Note that descendant adapter fields support multiple values per field, files and images. For these fields, the Value property will not correctly represent the value of the adapter field.

See Also

GetValueCount (see page 5415)

GetValues (see page 5415)

57.1.28.3.8 TAdapterNamedField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

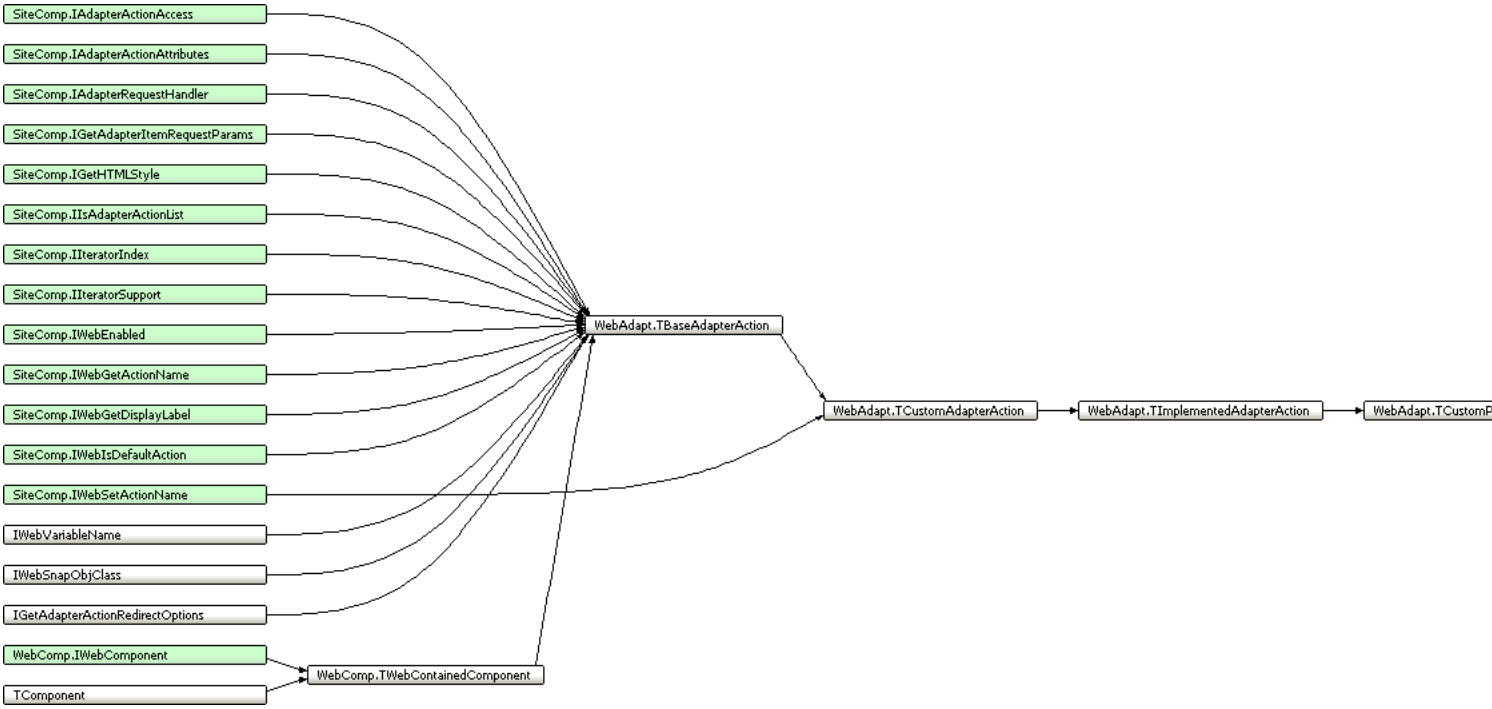
TWebUserList (see page 6759)

HasViewAccess (see page 5539)

57.1.29 TAdapterNextPageAction Class

TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5733).

Class Hierarchy



Pascal

```
TAdapterNextPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterNextPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterNextPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5733) such as TPagedAdapter (see page 5830).

See Also

TCustomAdapterPageAction (see page 5642)

TDefaultFieldsPagedAdapter (see page 5733)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

TPagedAdapter (see page 5830)

TDefaultFieldsPagedAdapter (see page 5733)

57.1.29.1 TAdapterNextPageAction Members

The following tables list the members exposed by TAdapterNextPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5363)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5363)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5647)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.

	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetNewPage (see page 5363)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5648)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5648)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 5649)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5649)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.

	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.
R	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
R	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5365)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5366)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5366)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5366)	Called before the adapter action response is generated.
	OnGetParams (see page 5367)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5364)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5364)	Defines the rights required to execute this action.
	RedirectOptions (see page 5365)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.29.2 TAdapterNextPageAction Methods

The methods of the TAdapterNextPageAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5363)	Gets the name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5363)	Provides the default display label for this adapter action.
	GetNewPage (see page 5363)	Indicates the page to display when this action is executed.

57.1.29.2.1 TAdapterNextPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "NextPage".

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.29.2.2 TAdapterNextPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns ">>". This displays as '>>' in the browser.

See Also

DisplayLabel (see page 5462)

57.1.29.2.3 TAdapterNextPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 5647)

RecordCount (see page 5747)

PageSize (see page 5747)

CurrentPage (see page 5746)

57.1.29.3 TAdapterNextPageAction Properties

The properties of the TAdapterNextPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5364)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5364)	Defines the rights required to execute this action.
	RedirectOptions (see page 5365)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.29.3.1 TAdapterNextPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.29.3.2 TAdapterNextPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6759)

hasexecuteaccess (see page 5539)

57.1.29.3.3 TAdapterNextPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

57.1.29.4 TAdapterNextPageAction Events

The events of the TAdapterNextPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5365)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5366)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5366)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5366)	Called before the adapter action response is generated.
	OnGetParams (see page 5367)	Specifies name/value pairs needed to execute this adapter action.

57.1.29.4.1 TAdapterNextPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.29.4.2 TAdapterNextPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.29.4.3 TAdapterNextPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.29.4.4 TAdapterNextPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.29.4.5 TAdapterNextPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

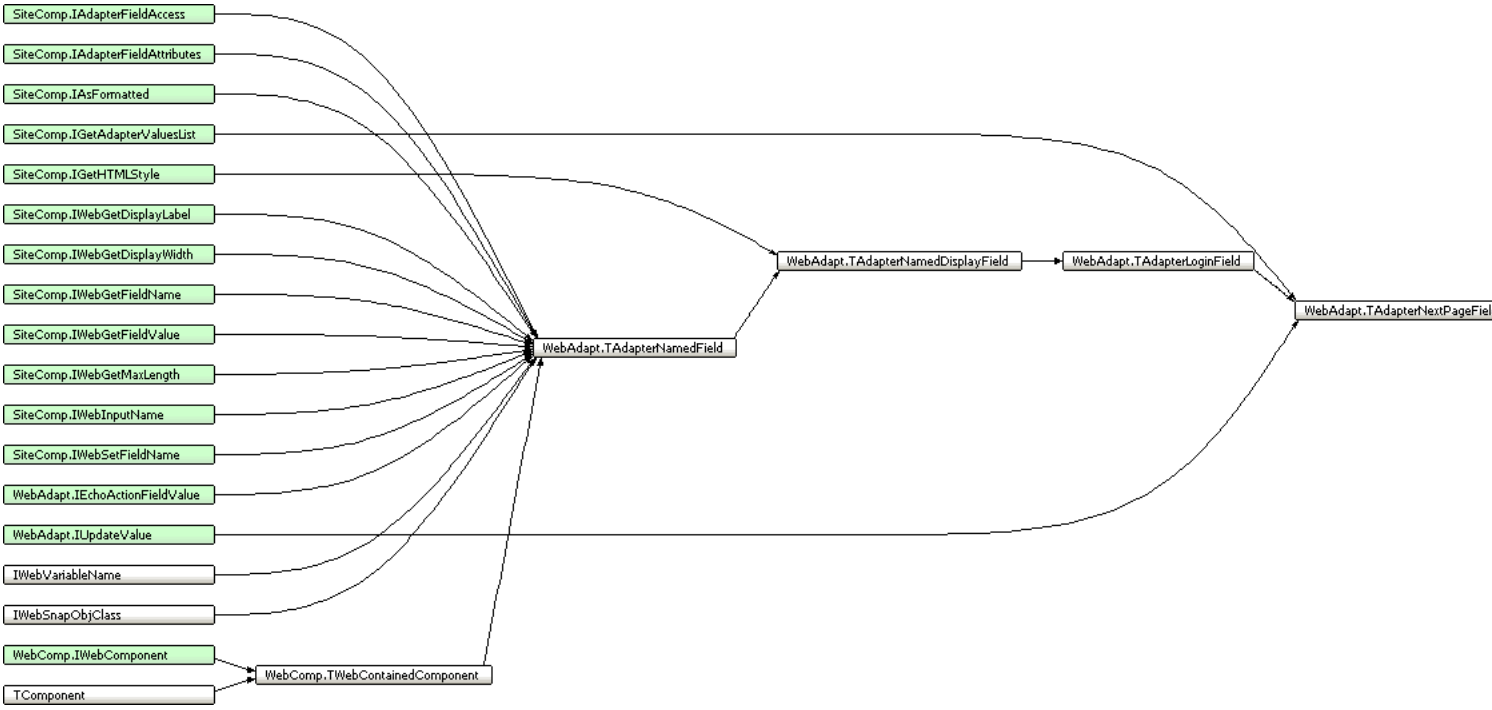
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.30 TAdapterNextPageField Class

The TAdapterNextPageField navigates to the next page of data.

Class Hierarchy



Pascal

```
TAdapterNextPageField = class(TAdapterLoginField, IGetAdapterValuesList, IUpdateValue);
```

C++

```
class TAdapterNextPageField : public TAdapterLoginField, public IGetAdapterValuesList, public
IUpdateValue;
```

File

WebAdapt

Description

TAdapterNextPageField is one of the components that is used by TPagedAdapter (see page 5830) along with the TAdapterNextPageAction (see page 5358) to navigate to the next page of data.

The TPagedAdapter (see page 5830) PageSize property indicates the number of rows displayed on a page. When the PageSize is not 0, the TPagedAdapter (see page 5830) component automatically adds built-in actions for moving between pages. These actions are TAdapterGotoPageAction (see page 5280), TAdapterPrevPageAction (see page 5384), and TAdapterNextPageAction (see page 5358). The TAdapterPageProducer can be used to generate HTML hyperlinks or buttons for these actions.

See Also

TPagedAdapter (see page 5830)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

TCustomAdapterNextPageAction

TCustomAdapterPageAction (see page 5642)

57.1.30.1 TAdapterNextPageField Members

The following tables list the members exposed by TAdapterNextPageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5372)	Returns a values list adapter containing the names of the published pages.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5372)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5372)	Returns the echo flag for this adapter field.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5373)	Returns the page name to be displayed after login.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5373)	Sets the echo flag for this adapter field.

	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5374)	Updates the parent adapter's next page property with the value entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 5371)	Frees an instance of TAdapterNextPageField (see page 5367).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 5311)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5374)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5374)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5375)	Identifies the name of this field when using server-side script.

57.1.30.2 TAdapterNextPageField.Destroy Destructor

Frees an instance of TAdapterNextPageField (see page 5367).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterNextPageField();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TAdapterNextPageField (see page 5367).

57.1.30.3 TAdapterNextPageField Methods

The methods of the TAdapterNextPageField class are listed here.

Protected Methods

	Name	Description
	GetAdapterValuesList (see page 5372)	Returns a values list adapter containing the names of the published pages.
V	GetInputStyleType (see page 5372)	Suggests an HTML input type for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5372)	Returns the echo flag for this adapter field.
V	ImplGetValue (see page 5373)	Returns the page name to be displayed after login.

	ImplSetEchoActionFieldValue (see page 5373)	Sets the echo flag for this adapter field.
	UpdateValue (see page 5374)	Updates the parent adapter's next page property with the value entered by the end-user.

57.1.30.3.1 TAdapterNextPageField.GetAdapterValuesList Method

Returns a values list adapter containing the names of the published pages.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList returns a reference to TLogicalPageList (see page 5805). This function will be called when server-side script references the ValuesList property of this adapter field. Typically, server-side script uses the ValuesList property to populate an HTML select list with the names of published pages. The end-user may select one of these pages to go to after login.

See Also

IValuesListAdapter (see page 1014)

57.1.30.3.2 TAdapterNextPageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType. GetInputStyleType returns htmliSelect indicating that a single selection list HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5335)

TAdapterInputHTMLElementType

57.1.30.3.3 TAdapterNextPageField.ImplGetEchoActionFieldValue Method

Returns the echo flag for this adapter field.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

ImplGetEchoActionFieldValue returns the value of an instance variable.

See Also

ImplGetEchoActionFieldValue (see page 5350)

EchoActionFieldValue (see page 5356)

ImplGetValue (see page 5373)

ImplSetEchoActionFieldValue (see page 5373)

57.1.30.3.4 TAdapterNextPageField.ImplGetValue Method

Returns the page name to be displayed after login.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

If the EchoActionFieldValue (see page 5356) property is true, then the page name selected or entered by end-user is returned. If the EchoActionFieldValue (see page 5356) property is false then the parent adapter's NextPage property is returned. The value returned by ImplGetValue is used when server-side script references the EditText property of this adapter field while generating the login form.

See Also

EchoActionFieldValue (see page 5356)

ActionValue (see page 5356)

NextPage (see page 5699)

57.1.30.3.5 TAdapterNextPageField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue sets an internal variable to true. The echo flag is typically set by an adapter action when an error occurs while processing an HTTP request. The purpose of the echo flag is to preserve the value entered by the end-user. When the echo flag is true, ImplGetValue (see page 5373) returns the value entered by the end-user rather than the value of the adapter field.

See Also

ImplSetEchoActionFieldValue (see page 5353)

EchoActionFieldValue (see page 5356)

ImplGetValue (see page 5373)

ImplGetEchoActionFieldValue (see page 5372)

57.1.30.3.6 TAdapterNextPageField.UpdateValue Method

Updates the parent adapter's next page property with the value entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

Typically a single selection list HTML element is displayed on the login form. The list contains all of the published pages. The HTTP request will contain the page name selected by the end-user.

See Also

NextPage (see page 5699)

IActionRequest

57.1.30.4 TAdapterNextPageField Properties

The properties of the TAdapterNextPageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5374)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5374)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5375)	Identifies the name of this field when using server-side script.

57.1.30.4.1 TAdapterNextPageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.30.4.2 TAdapterNextPageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.30.4.3 TAdapterNextPageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

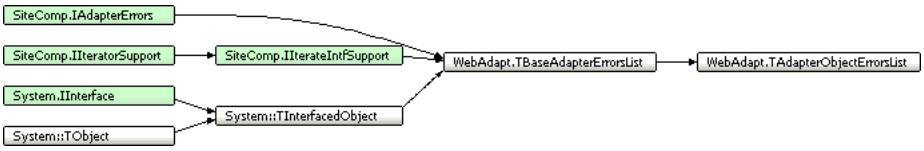
```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.31 TAdapterObjectErrorsList Class

Class Hierarchy



Pascal

```
TAdapterObjectErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterObjectErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterObjectErrorsList.

57.1.31.1 TAdapterObjectErrorsList Members

The following tables list the members exposed by TAdapterObjectErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	EndIterator (see page 5471)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 5471)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5471)	Returns a particular error.
	GetErrorCount (see page 5471)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5472)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5472)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 5472)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 5378)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
V	ImplGetError (see page 5378)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
V	ImplGetErrorCount (see page 5378)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.
V	ImplGetObjectErrors (see page 5473)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 5473)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5474)	Advances to the objects next item.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5474)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AddError (see page 5469)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 938)	Clears the errors list.
	ClearErrors (see page 5470)	Clear error list.
	Create (see page 5377)	This is Create, a member of class TAdapterObjectErrorsList.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.

	DefineLabel (see page 938)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 5470)	Define error label for field.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetCurrentIntf (see page 992)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 938)	Returns a particular error.
	GetErrorCount (see page 939)	Return the number of errors in the errors list.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 939)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 939)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 997)	Advances to the objects next item.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ErrorCount (see page 940)	Returns the number of errors in the errors list.
R	ErrorCount (see page 5474)	Number of error objects contained.
R	Errors (see page 940)	Returns an error from the list.
R	Errors (see page 5475)	Access to error objects.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

57.1.31.2 TAdapterObjectErrorsList.Create Constructor

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; const AName: string);
```

C++

```
__fastcall TAdapterObjectErrorsList(TAdapterErrorsList AErrors, const UnicodeString AName);
```

Description

This is Create, a member of class TAdapterObjectErrorsList.

57.1.31.3 TAdapterObjectErrorsList Methods

The methods of the TAdapterObjectErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplDefineLabel (see page 5378)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
	ImplGetError (see page 5378)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
	ImplGetErrorCount (see page 5378)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

57.1.31.3.1 TAdapterObjectErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.

57.1.31.3.2 TAdapterObjectErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterObjectErrorsList.

57.1.31.3.3 TAdapterObjectErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

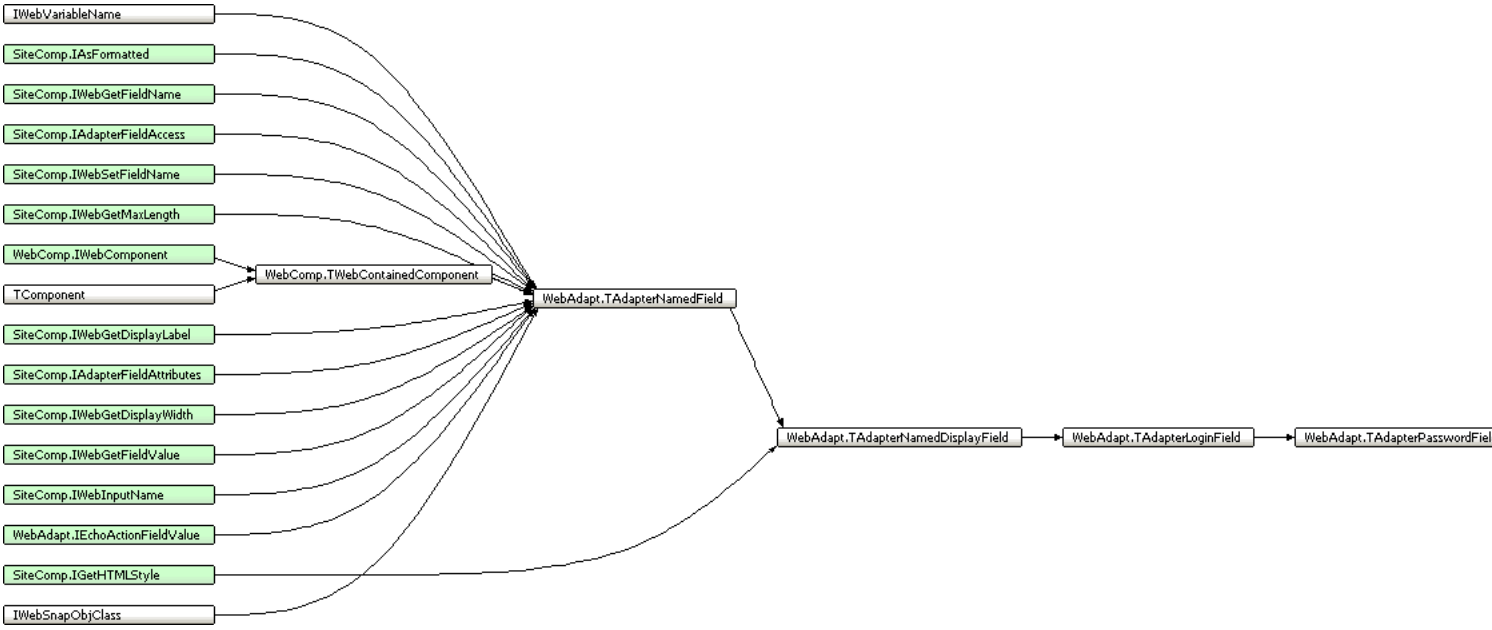
Description

This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

57.1.32 TAdapterPasswordField Class

TAdapterPasswordField is an adapter field used to access the password entered by the end-user.

Class Hierarchy



Pascal

```
TAdapterPasswordField = class(TAdapterLoginField);
```

C++

```
class TAdapterPasswordField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterPasswordField is added to TLoginFormAdapter (see page 5810). TAdapterUserNameField (see page 5422), TAdapterPasswordField, TLoginFormAdapterLoginAction (see page 5821) and TAdapterNextPageField (see page 5367) are used with TLoginFormAdapter (see page 5810) to generate an HTML login form. When the HTML form is submitted, TAdapterPasswordField is used to extract the end user's password from the HTTP request.

See Also

TAdapterLoginField (see page 5308)

TLoginFormAdapter (see page 5810)

TAdapterUserNameField (see page 5422)

TAdapterNextPageField (see page 5367)

TLoginFormAdapterLoginAction (see page 5821)

57.1.32.1 TAdapterPasswordField Members

The following tables list the members exposed by TAdapterPasswordField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5382)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5383)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.

	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.

R	Adapter (see page 5311)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5383)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5383)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5384)	Identifies the name of this field when using server-side script.

57.1.32.2 TAdapterPasswordField Methods

The methods of the TAdapterPasswordField class are listed here.

Protected Methods

	Name	Description
V	GetInputStyleType (see page 5382)	Suggests an HTML input type for this adapter field.
V	ImplGetValue (see page 5383)	Returns the value associated with this adapter field.

57.1.32.2.1 TAdapterPasswordField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliPassword indicating that a password input HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5335)

TAdapterInputHTMLElementType

57.1.32.2.2 TAdapterPasswordField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 5357) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's password field will be blank. When the login form is submitted, the ActionValue (see page 5356) property can be used to access the password entered by the end-user.

See Also

ActionValue (see page 5356)

57.1.32.3 TAdapterPasswordField Properties

The properties of the TAdapterPasswordField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5383)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5383)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5384)	Identifies the name of this field when using server-side script.

57.1.32.3.1 TAdapterPasswordField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.32.3.2 TAdapterPasswordField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.32.3.3 TAdapterPasswordField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

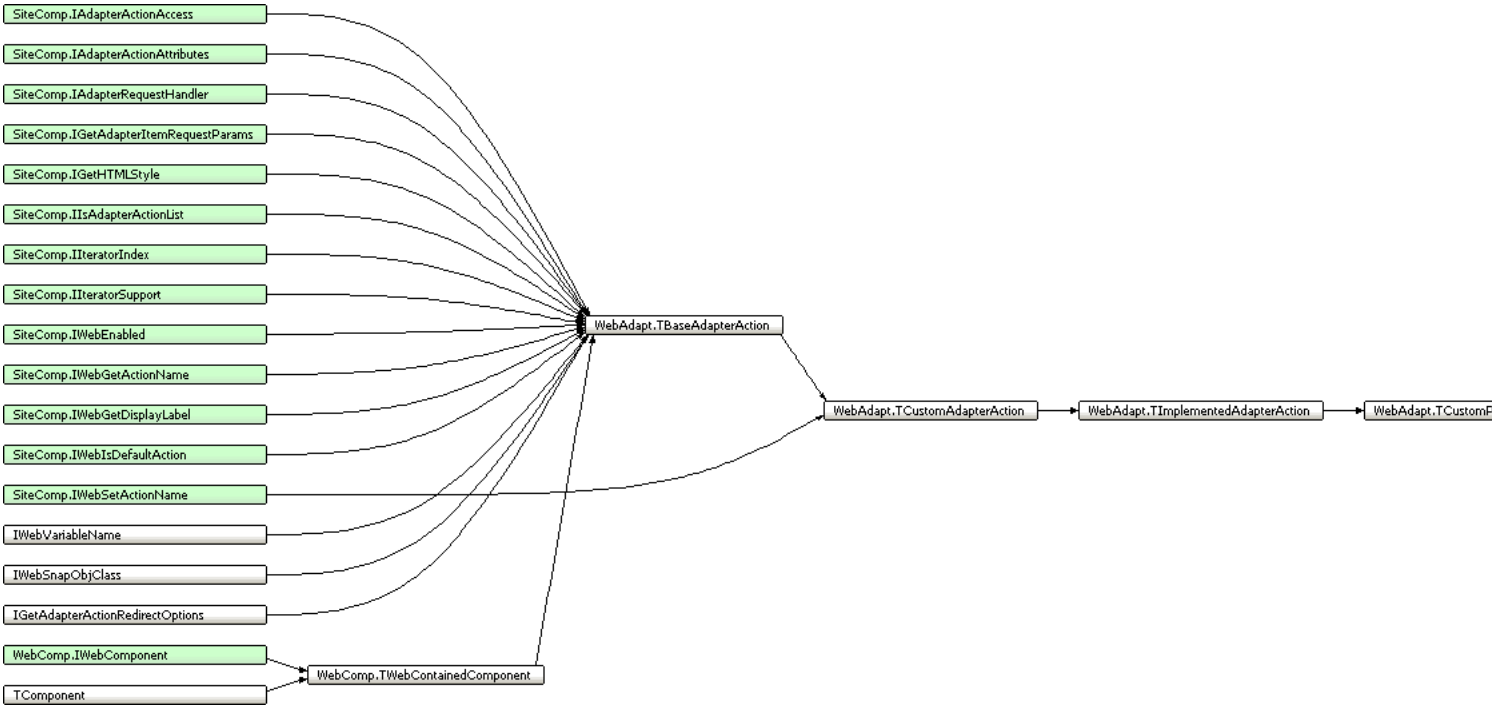
Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.33 TAdapterPrevPageAction Class

TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5733).

Class Hierarchy



Pascal

```
TAdapterPrevPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterPrevPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterPrevPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5733) such as TPagedAdapter (see page 5830).

See Also

TCustomAdapterPageAction (see page 5642)

TDefaultFieldsPagedAdapter (see page 5733)

TAdapterNextPageAction (see page 5358)

TAdapterGotoPageAction (see page 5280)

TPagedAdapter (see page 5830)

TDefaultFieldsPagedAdapter (see page 5733)

57.1.33.1 TAdapterPrevPageAction Members

The following tables list the members exposed by TAdapterPrevPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5389)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5389)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5647)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
V	GetNewPage (see page 5390)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5648)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5648)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.

	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 5649)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5649)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5392)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5392)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5392)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5393)	Called before the adapter action response is generated.
	OnGetParams (see page 5393)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5390)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5391)	Defines the rights required to execute this action.
	RedirectOptions (see page 5391)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.33.2 TAdapterPrevPageAction Methods

The methods of the TAdapterPrevPageAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5389)	Gets the name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5389)	Provides the default display label for this adapter action.
	GetNewPage (see page 5390)	Indicates the page to display when this action is executed.

57.1.33.2.1 TAdapterPrevPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "PrevPage".

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.33.2.2 TAdapterPrevPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns "<<". This displays as '<<' in the browser.

See Also

DisplayLabel (see page 5462)

57.1.33.2.3 TAdapterPrevPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 5647)

RecordCount (see page 5747)

PageSize (see page 5747)

CurrentPage (see page 5746)

57.1.33.3 TAdapterPrevPageAction Properties

The properties of the TAdapterPrevPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5390)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5391)	Defines the rights required to execute this action.
	RedirectOptions (see page 5391)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.33.3.1 TAdapterPrevPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side

script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.33.3.2 TAdapterPrevPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6759)

hasexecuteaccess (see page 5539)

57.1.33.3.3 TAdapterPrevPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

57.1.33.4 TAdapterPrevPageAction Events

The events of the TAdapterPrevPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5392)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5392)	Called after the adapter action response is generated.

	OnBeforeExecute (see page 5392)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5393)	Called before the adapter action response is generated.
	OnGetParams (see page 5393)	Specifies name/value pairs needed to execute this adapter action.

57.1.33.4.1 TAdapterPrevPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.33.4.2 TAdapterPrevPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.33.4.3 TAdapterPrevPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.33.4.4 TAdapterPrevPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.33.4.5 TAdapterPrevPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

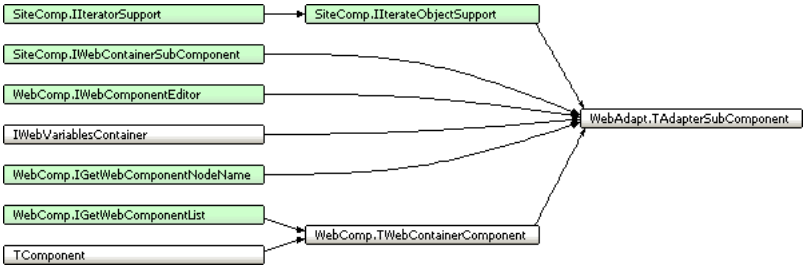
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.34 TAdapterSubComponent Class

TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterSubComponent = class(TWebContainerComponent, IWebContainerSubComponent,
IWebComponentEditor, IWebVariablesContainer, IGetWebComponentNodeName, IIterateObjectSupport);
```

C++

```
class TAdapterSubComponent : public TWebContainerComponent, public IWebContainerSubComponent,
public IWebComponentEditor, public IWebVariablesContainer, public IGetWebComponentNodeName,
public IIterateObjectSupport;
```

File

WebAdapt

Description

Adapter (see page 5406) components use a TAdapterSubComponent descendant to represent the list of actions they can execute or the list of fields whose values they support. Each TAdapterSubComponent is specialized for a specific type of object (actions or fields), and represents either the set of explicitly assigned actions or fields, or the set that the adapter contains by default.

TAdapterSubComponent introduces support for:

Associating the collection with an adapter component.

Assigning variable names to the child items in the collection. These are the names of variables that server-side script can use to refer to the actions or fields.

Letting server-side script use an enumerator to iterate over all the child items in the collection.

Working with the WebSnap designers at design time.

See Also

TAdapterActions (see page 5198)

TAdapterFields (see page 5265)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

57.1.34.1 TAdapterSubComponent Members

The following tables list the members exposed by TAdapterSubComponent.

Protected Methods

	Name	Description
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5399)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5401)	Returns the value of the VariablesInUse (see page 5407) property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5402)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
V	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5403)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5404)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5405)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
V	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
V	Destroy (see page 6038)	Frees an instance of TWebContainerComponent (see page 6036).
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5400)	Returns the associated adapter.
V	HasParent (see page 5402)	Indicates whether this component has an associated adapter.
V	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.

	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.

57.1.34.2 TAdapterSubComponent Methods

The methods of the TAdapterSubComponent class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 5396)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5397)	Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5397)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5398)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5398)	Returns the Web module that contains a specified item from this list.
	GetCurrentObject (see page 5399)	Returns the current field or action from the list when an enumerator is iterating over child items.
	GetDefaultWebComponents (see page 5399)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5400)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5401)	Indicates the number of items in this list that can be accessed from server-side script.
	GetVariablesInUse (see page 5401)	Returns the value of the VariablesInUse (see page 5407) property.
	GetWebComponentNodeName (see page 5401)	Returns a string that indicates the type of items in this collection.
	ImplCanAddClass (see page 5402)	Provides the underlying implementation for the CanAddClass (see page 5396) method.
	ImplGetWebComponentNodeName (see page 5403)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.
	NextIteration (see page 5403)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5404)	Notifies the associated adapter when the list of items changes.
	SetDefaultWebComponents (see page 5404)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5405)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5406)	Makes the first child item with a variable name the current object.

Public Methods

	Name	Description
	GetParentComponent (see page 5400)	Returns the associated adapter.
	HasParent (see page 5402)	Indicates whether this component has an associated adapter.
	ReadState (see page 5404)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5405)	Sets the Adapter (see page 5406) property to be the same as this component's parent.

57.1.34.2.1 TAdapterSubComponent.CanAddClass Method

Indicates whether an object of a specified type can be added to this collection.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to this list. This method allows the IDE to limit the

actions or fields in the collection to those that are appropriate for the associated adapter.

AParent is the TAdapterSubComponent (see page 5394) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

CanAddClass returns true if the TAdapterSubComponent (see page 5394) descendant can include a child of type AClass, false otherwise.

CanAddClass calls the virtual ImplCanAddClass (see page 5402) method to determine whether a class of Web item is valid.

See Also

ImplCanAddClass (see page 5402)

Adapter (see page 5406)

57.1.34.2.2 TAdapterSubComponent.EndIterator Method

Stops the TAdapterSubComponent (see page 5394) object from iterating over the Web items it lists.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

EndIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called when the enumeration stops. In TAdapterSubComponent (see page 5394), EndIterator does nothing. This method exists only as a backstop for the standard call made from server-side script.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5406) or NextIteration (see page 5403) method. TAdapterSubComponent (see page 5394) does not use the OwnerData parameter.

See Also

StartIterator (see page 5406)

NextIteration (see page 5403)

GetCurrentObject (see page 5399)

57.1.34.2.3 TAdapterSubComponent.FindNamedVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

FindNamedVariable searches the list of child items that have variable names (the VariablesInUse (see page 5407) property) to find one with the specified name. The associated adapter component calls this method from its FindNamedField or FindNamedAction method to look up a field or action with a specified name.

AName is a variable name. Typically, this comes from server-side script that is looking for a child of an adapter with this name.

FindNamedVariable returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 5407)

FindVariable (see page 5398)

GetVariable (see page 5400)

GetVariableCount (see page 5401)

FindVariableContainer (see page 5398)

FindNamedField (see page 5531)

FindNamedAction (see page 5531)

57.1.34.2.4 TAdapterSubComponent.FindVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

FindVariable method is called from server-side script to locate a field or action with a specified name. This method is identical to the FindNamedVariable (see page 5397) method, except that it is not virtual. Instead, FindVariable calls the virtual FindNamedVariable (see page 5397) method.

AName is a variable name.

FindNamedVariable (see page 5397) returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable (see page 5397) returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 5407)

FindNamedVariable (see page 5397)

GetVariable (see page 5400)

GetVariableCount (see page 5401)

FindVariableContainer (see page 5398)

57.1.34.2.5 TAdapterSubComponent.FindVariableContainer Method

Returns the Web module that contains a specified item from this list.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer searches the list of child items that have variable names (the VariablesInUse (see page 5407) property) to find one with the specified name. It then returns the Web module that contains the component it located.

AName is the name of the action or field to locate.

FindVariableContainer returns the Web module that containse the specified item. If there is no item in the list with the name AName, FindVariableContainer returns nil (Delphi) or NULL (C++). If FindVariableContainer includes an item with the specified name, but is unable to determine what Web module contains that item, FindVariableContainer raises an EAdapterException (see page 5166) exception.

See Also

VariablesInUse (see page 5407)

FindNamedVariable (see page 5397)

GetVariable (see page 5400)

GetVariableCount (see page 5401)

EAdapterException (see page 5166)

57.1.34.2.6 TAdapterSubComponent.GetCurrentObject Method

Returns the current field or action from the list when an enumerator is iterating over child items.

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

GetCurrentObject is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method returns the current child item.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5406) or NextIteration (see page 5403) method. TAdapterSubComponent (see page 5394) does not use the OwnerData parameter.

See Also

StartIterator (see page 5406)

NextIteration (see page 5403)

EndIterator (see page 5397)

57.1.34.2.7 TAdapterSubComponent.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 5394), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5220) and TAdapterDefaultFields (see page 5226)) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6039)

TAdapterDefaultActions (see page 5220)

TAdapterDefaultFields (see page 5226)

57.1.34.2.8 TAdapterSubComponent.GetParentComponent Method

Returns the associated adapter.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TAdapterSubComponent (see page 5394) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the Adapter (see page 5406) property (the adapter that uses this component to represent the set of its fields or actions).

See Also

GetParentComponent

HasParent (see page 5402)

Adapter (see page 5406)

57.1.34.2.9 TAdapterSubComponent.GetVariable Method

Returns a child item from this list, given its position in the list of child items with variable names.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

FindNamedVariable (see page 5397) indexes into the list of child items that have variable names (the VariablesInUse (see page 5407) property) and returns the item in the specified position.

I is the index of the child item, where 0 is the first item that can be reference from server-side script, 1 is the second item, and so on. The GetVariableCount (see page 5401) method returns an upper limit for this index.

See Also

VariablesInUse (see page 5407)

FindNamedVariable (see page 5397)

FindVariable (see page 5398)

GetVariableCount (see page 5401)

57.1.34.2.10 TAdapterSubComponent.GetVariableCount Method

Indicates the number of items in this list that can be accessed from server-side script.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount returns the number of child items that have variable names. This is the number of items in the VariablesInUse (see page 5407) proprety. Use GetVariableCount as an upper bound to the I parameter of the GetVariable (see page 5400) method.

See Also

VariablesInUse (see page 5407)

GetVariable (see page 5400)

57.1.34.2.11 TAdapterSubComponent.GetVariablesInUse Method

Returns the value of the VariablesInUse (see page 5407) property.

Pascal

```
function GetVariablesInUse: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse (see page 5407) property.

In TAdapterSubComponent (see page 5394), GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5407)

GetWebComponents (see page 6039)

57.1.34.2.12 TAdapterSubComponent.GetWebComponentNodeName Method

Returns a string that indicates the type of items in this collection.

Pascal

```
function GetWebComponentNodeName: string;
```

C++

```
__fastcall UnicodeString GetWebComponentNodeName();
```

Description

GetWebComponentNodeName returns the label for the type of items in the collection. For example, if the TAdapterSubComponent (see page 5394) descendant holds the fields belonging to an adapter, GetWebComponentNodeName returns the string 'Fields'. If the TAdapterSubComponent (see page 5394) descendant holds the actions belonging to an adapter, GetWebComponentNodeName returns the string 'Actions'.

GetWebComponentNodeName delegates the task of supplying a name to the virtual ImplGetWebComponentNodeName (see page 5403) method.

See Also

ImplGetWebComponentNodeName (see page 5403)

57.1.34.2.13 TAdapterSubComponent.HasParent Method

Indicates whether this component has an associated adapter.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TAdapterSubComponent (see page 5394) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the Adapter (see page 5406) property is not nil (Delphi) or NULL (C++).

If the TAdapterSubComponent (see page 5394) object has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 5400)

Adapter (see page 5406)

57.1.34.2.14 TAdapterSubComponent.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5396) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5396) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterSubComponent (see page 5394) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterSubComponent (see page 5394) descendant can include a child of type AClass,

false otherwise.

In TAdapterSubComponent (see page 5394), ImplCanAddClass always returns false, disallowing any child items. Descendant classes override this method to conditionally allow those fields or actions that are appropriate to the associated adapter.

See Also

CanAddClass (see page 5396)

Adapter (see page 5406)

57.1.34.2.15 TAdapterSubComponent.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5401) method.

Pascal

```
function ImplGetWebComponentNodeName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5401) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the name of the type of items in this list.

In TAdapterSubComponent (see page 5394), ImplGetWebComponentNodeName returns an empty string. Descendant classes created for specific types of elements override this method to return an appropriate string such as 'Fields' or 'Actions'.

See Also

GetWebComponentNodeName (see page 5401)

57.1.34.2.16 TAdapterSubComponent.NextIteration Method

Changes the current object to the next item that has a variable name.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

NextIteration is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to move to the next item in the list. After a call to NextIteration, the GetCurrentObject (see page 5399) method returns the next item in the VariablesInUse (see page 5407) list.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5406) or NextIteration method. TAdapterSubComponent (see page 5394) does not use the OwnerData parameter.

NextIteration returns true if it successfully moves to a new item, false if there was no addition item to which it could move.

See Also

StartIterator (see page 5406)

EndIterator (see page 5397)

GetCurrentObject (see page 5399)

VariablesInUse (see page 5407)

57.1.34.2.17 TAdapterSubComponent.NotifyAdapterChange Method

Notifies the associated adapter when the list of items changes.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

TAdapterSubComponent (see page 5394) descendants call NotifyAdapterChange internally when the items in this list change. NotifyAdapterChange passes this notification on to the associated adapter so that it can make any necessary adjustments.

See Also

Adapter (see page 5406)

NotifyAdapterChange (see page 5551)

57.1.34.2.18 TAdapterSubComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TAdapterSubComponent (see page 5394) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the Adapter (see page 5406) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 5400)

Adapter (see page 5406)

57.1.34.2.19 TAdapterSubComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 5399) supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 5394), GetDefaultWebComponents (see page 5399) always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5220) and TAdapterDefaultFields (see page 5226)) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent (see page 6036), SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5220) and TAdapterDefaultFields (see page 5226)) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5399)

IsDefaultField (see page 6015)

57.1.34.2.20 TAdapterSubComponent.SetParentComponent Method

Sets the Adapter (see page 5406) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TAdapterSubComponent (see page 5394) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the Adapter (see page 5406) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 5402)

Adapter (see page 5406)

57.1.34.2.21 TAdapterSubComponent.SetWebParent Method

Introduces a method to associate this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

TAdapterSubComponent (see page 5394) introduces SetWebParent as a virtual method for setting the read-only Adapter (

see page 5406) property.

Value is the adapter component that acts as a parent to this collection.

In TAdapterSubComponent (see page 5394), SetWebParent does nothing. Descendant classes override this method to set the Adapter (see page 5406) property and populate the list with actions or fields appropriate to that adapter.

See Also

Adapter (see page 5406)

57.1.34.2.22 TAdapterSubComponent.StartIterator Method

Makes the first child item with a variable name the current object.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

StartIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to begin iterating over the items in the list. After a call to StartIterator, the GetCurrentObject (see page 5399) method returns the first item in the VariablesInUse (see page 5407) list.

OwnerData is a pointer to object-specific information returned that should be passed to the NextIteration (see page 5403) or EndIterator (see page 5397) method. TAdapterSubComponent (see page 5394) does not use the OwnerData parameter.

StartIterator returns true if it successfully moves to the first item, false if the VariablesInUse (see page 5407) property does not list any items.

See Also

NextIteration (see page 5403)

EndIterator (see page 5397)

GetCurrentObject (see page 5399)

VariablesInUse (see page 5407)

57.1.34.3 TAdapterSubComponent Properties

The properties of the TAdapterSubComponent class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5406)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5407)	Lists those fields or actions in the collection for which variables are defined.

57.1.34.3.1 TAdapterSubComponent.Adapter Property

Indicates the adapter that uses this object to manage its fields or actions.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Read (see page 3727) Adapter to access the adapter component that uses this collection. This TAdapterSubComponent (see page 5394) object is the value of the adapter's Data or Actions property.

See Also

Actions (see page 5558)

Data (see page 5558)

Actions (see page 5732)

Data (see page 5733)

57.1.34.3.2 TAdapterSubComponent.VariablesInUse Property

Lists those fields or actions in the collection for which variables are defined.

Pascal

```
property VariablesInUse: TWebComponentList;
```

C++

```
__property TWebComponentList VariablesInUse;
```

Description

Read (see page 3727) VariablesInUse to access the child items (fields or actions) that can be addressed from server-side script.

TAdapterSubComponent (see page 5394) implements VariablesInUse to include all of the fields or actions that have been added to the list. Descendant classes can override this property to provide a different set of items, such as the default fields or actions of the associated adapter.

See Also

FindNamedVariable (see page 5397)

FindVariable (see page 5398)

GetVariable (see page 5400)

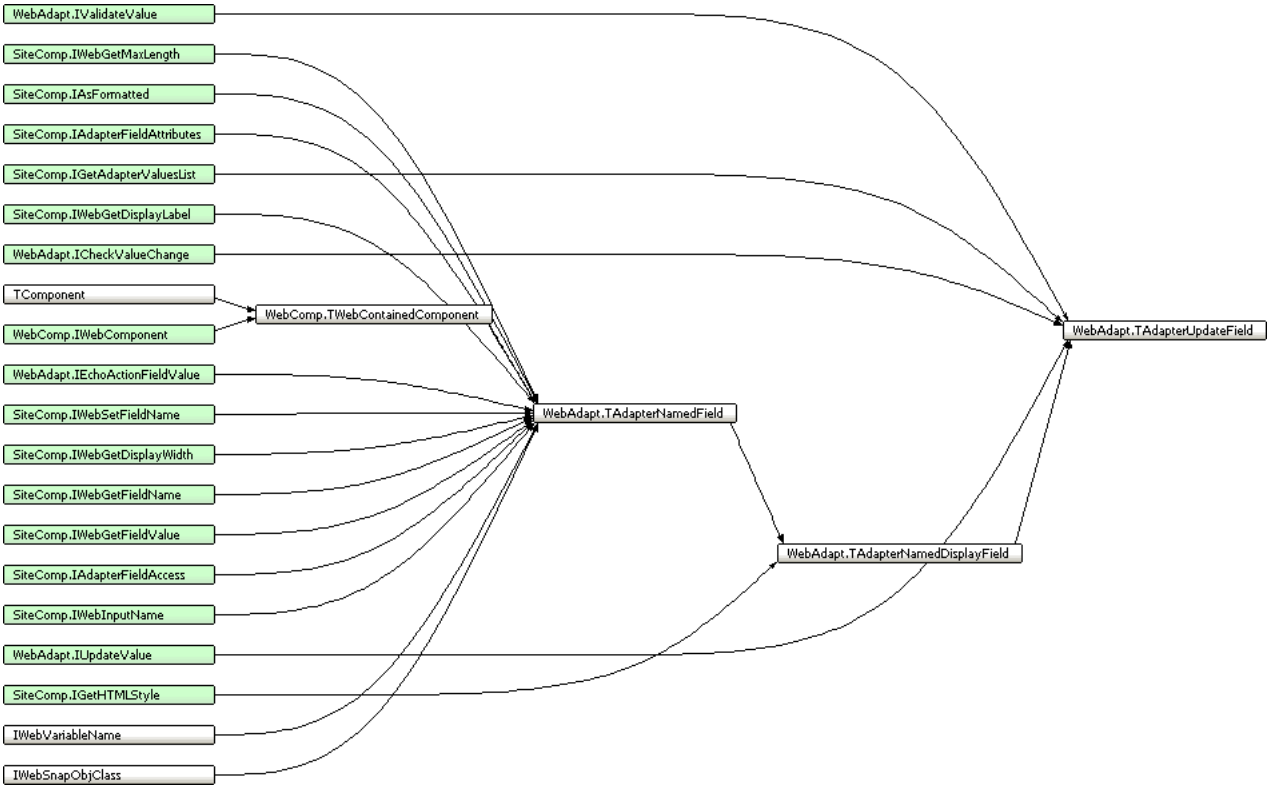
GetVariableCount (see page 5401)

FindVariableContainer (see page 5398)

57.1.35 TAdapterUpdateField Class

TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.

Class Hierarchy



Pascal

```
TAdapterUpdateField = class(TAdapterNamedDisplayField, IGetAdapterValuesList, IUpdateValue,
IValidateValue, ICheckValueChange);
```

C++

```
class TAdapterUpdateField : public TAdapterNamedDisplayField, public IGetAdapterValuesList,
public IUpdateValue, public IValidateValue, public ICheckValueChange;
```

File

WebAdapt

Description

TAdapterUpdateField implements the interfaces necessary to support field validation and updating. Descendant classes support different types of data that can be entered by the user (for example, image upload, file upload, single value, multiple value, memo and Boolean).

See Also

TAdapterNamedDisplayField (see page 5331)

TCustomAdapterField (see page 5583)

TAdapter (see page 5179)

57.1.35.1 TAdapterUpdateField Members

The following tables list the members exposed by TAdapterUpdateField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.

	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.35.2 TAdapterUpdateField.Create Constructor

Creates an instance of TAdapterUpdateField (see page 5407).

Pascal

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

C++

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

Description

Do not call TAdapterUpdateField (see page 5407). It will be called automatically when the TAdapterUpdateField (see page 5407) object is created.

See Also

Create_d

57.1.35.3 TAdapterUpdateField Methods

The methods of the TAdapterUpdateField class are listed here.

Protected Methods

	Name	Description
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
V	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
V	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
V	Notification (see page 5419)	Clears references to a deleted component.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

57.1.35.3.1 TAdapterUpdateField.CheckValueChange Method

Checks the value(s) entered by the end-user against the value associated with this adapter field.

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

CheckValueChange is called to determine whether the value(s) entered by the user is different from the value(s) provided by the adapter field. CheckValueChange calls the virtual function ImplCheckValueChange (see page 5416).

See Also

ImplCheckValueChange (see page 5416)

57.1.35.3.2 TAdapterUpdateField.ConvertValue Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue is called to retrieve, as a variant, the value entered by the user for this adapter field. ConvertValue using index 0 to retrieve a value from the IActionFieldValue (see page 5890) parameter. Typically, IActionFieldValue (see page 5890) will contain a string value. Descendants may override this function to caste the value into another type.

See Also

ConvertValue (see page 5580)

57.1.35.3.3 TAdapterUpdateField.ConvertValueCount Method

Retrieves the number of values entered by the end-user.

Pascal

```
function ConvertValueCount(AValue: IActionFieldValue): Integer; virtual;
```

C++

```
virtual __fastcall int ConvertValueCount(IActionFieldValue AValue);
```

Description

ConvertValueCount is called to retrieve the number of values entered by the user. ConvertValueCount returns 1. Descendants may override this method.

57.1.35.3.4 TAdapterUpdateField.ConvertValues Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValues(AValue: IActionFieldValue; Index: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValues(IActionFieldValue AValue, int Index);
```

Description

ConvertValues is called to retrieve, as a variant, a particular value entered by the end-user for this adapter field.

57.1.35.3.5 TAdapterUpdateField.DoOnValidateValue Method

Validates field values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); virtual;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue is called when the field value entered by the end-user is to be validated. DoOnValidateValue calls the OnValidateValue (see page 5421) event handler. DoOnValidateValue may be overridden by descendants.

See Also

OnValidateValue (see page 5421)

ImplValidateValue (see page 5418)

57.1.35.3.6 TAdapterUpdateField.GetAdapterValuesList Method

Provides an interface to a component that can be used to generate a list of possible values for this adapter field.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList provides an interface to a component that can be used to generate a list of possible values for this adapter field. It does this by returning the value of the ValuesList (see page 5420) property.

See Also

IValuesListAdapter (see page 1014)

57.1.35.3.7 TAdapterUpdateField.GetInputStyleType Method

Indicates the preferred HTML element to use when editing this adapter field's value.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType returns htmliSelect if the ValuesList (see page 5420) property is not nil (Delphi) or NULL (C++). Otherwise, htmliTextInput is returned.

See Also

TAdapterInputHTMLElementType

ValuesList (see page 5420)

57.1.35.3.8 TAdapterUpdateField.GetValueCount Method

Provides the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns the number of values provided by this adapter field. GetValueCount returns 1. Descendant classes override this method.

See Also

TAdapterMultiValueField (see page 5320)

GetValueCount

57.1.35.3.9 TAdapterUpdateField.GetValues Method

Returns one of the values provided by this adapter field.

Pascal

```
function GetValues(I: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns one of the values provided by this adapter field. GetValues calls GetValue (see page 5347). Descendant classes override this method.

See Also

GetValues

TAdapterMultiValueField (see page 5320)

57.1.35.3.10 TAdapterUpdateField.ImplAsFormatted Method

Returns a string representation of the adapter field's value that is appropriate for a read-only display.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted calls the OnGetDisplayText (see page 5421) event handler, if one is defined. Otherwise, ImplAsFormatted returns the adapter field's value converted to a string. Server-side script accesses the value returned by this function using the AsDisplayText property.

See Also

OnGetDisplayText (see page 5421)

GetValue (see page 5347)

57.1.35.3.11 TAdapterUpdateField.ImplCheckValueChange Method

Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange compares the number and content of values associated with the adapter field with the number and content of values entered by the end-user.

See Also

CheckValueChange (see page 5412)

57.1.35.3.12 TAdapterUpdateField.ImplGetDisplayLabel Method

Gets the preferred display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5338) property.

See Also

DisplayLabel (see page 5338)

57.1.35.3.13 TAdapterUpdateField.ImplGetDisplayWidth Method

Returns the preferred display width for the HTML element used to edit or display this adapter field's value.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 5338) property.

See Also

DisplayWidth (see page 5338)

57.1.35.3.14 TAdapterUpdateField.ImplGetEchoActionFieldValue Method

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

See Also

ImplSetEchoActionFieldValue (see page 5417)

57.1.35.3.15 TAdapterUpdateField.ImplGetVariableName Method

Gets the name used to access this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

ImplGetVariableName returns the value of the FieldName (see page 5357) property.

See Also

FieldName (see page 5357)

57.1.35.3.16 TAdapterUpdateField.ImplSetEchoActionFieldValue Method

Sets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue is typically called when an error occurs during the execution of an adapter action. It allows the page generated after the error to display the values entered by the user rather than the action adapter field values.

57.1.35.3.17 TAdapterUpdateField.ImplSetFieldName Method

Sets the name used to access this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); override;
```

C++

```
virtual __fastcall ImplSetFieldName(const UnicodeString Value);
```

Description

ImplSetVariableName sets the value of the FieldName (see page 5357) property.

See Also

FieldName (see page 5357)

57.1.35.3.18 TAdapterUpdateField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue is a placeholder. Descendants override this method.

See Also

UpdateValue (see page 5419)

57.1.35.3.19 TAdapterUpdateField.ImplValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ImplValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplValidateValue calls the virtual function DoOnValidateValue (see page 5414) after extracting an IActionFieldValue (see page 5890) that contains the value(s) entered by the user. The AFieldIndex parameter identifies that location of the IActionFieldValue (see page 5890) in the AActionRequest.

See Also

ValidateValue (see page 5419)

DoOnValidateValue (see page 5414)

IActionRequest

57.1.35.3.20 TAdapterUpdateField.IsMultiValueField Method

Indicates whether this adapter field can support multiple values.

Pascal

```
function IsMultiValueField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsMultiValueField();
```

Description

IsMultiValueField return true to indicate that this adapter field can provide multiple values. IsMultiValueField returns false. Descendants may override this function.

See Also

GetValueCount (see page 5415)

GetValues (see page 5415)

TAdapterMultiValueField (see page 5320)

57.1.35.3.21 TAdapterUpdateField.Notification Method

Clears references to a deleted component.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted. TAdapterUpdateField (see page 5407) clears the ValuesList (see page 5420) property when the associated component is deleted.

57.1.35.3.22 TAdapterUpdateField.UpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

UpdateValue calls the virtual method ImplUpdateValue (see page 5418).

See Also

ImplUpdateValue (see page 5418)

57.1.35.3.23 TAdapterUpdateField.ValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ValidateValue calls the virtual function ImplValidateValue (see page 5418).

See Also

ImplValidateValue (see page 5418)

IActionRequest

57.1.35.4 TAdapterUpdateField Properties

The properties of the TAdapterUpdateField class are listed here.

Public Properties

	Name	Description
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.

57.1.35.4.1 TAdapterUpdateField.Value Property

Retrieves the value associated with this adapter field.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

GetValue (see page 5347) calls the GetValue (see page 5347) method to retrieve the value of the adapter field as a variant.

See Also

GetValue (see page 5347)

57.1.35.4.2 TAdapterUpdateField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1014)

TStringsValuesList (see page 5843)

TDataSetValuesList

57.1.35.5 TAdapterUpdateField Events

The events of the TAdapterUpdateField class are listed here.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

57.1.35.5.1 TAdapterUpdateField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 5415)

57.1.35.5.2 TAdapterUpdateField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue (see page 5414) to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

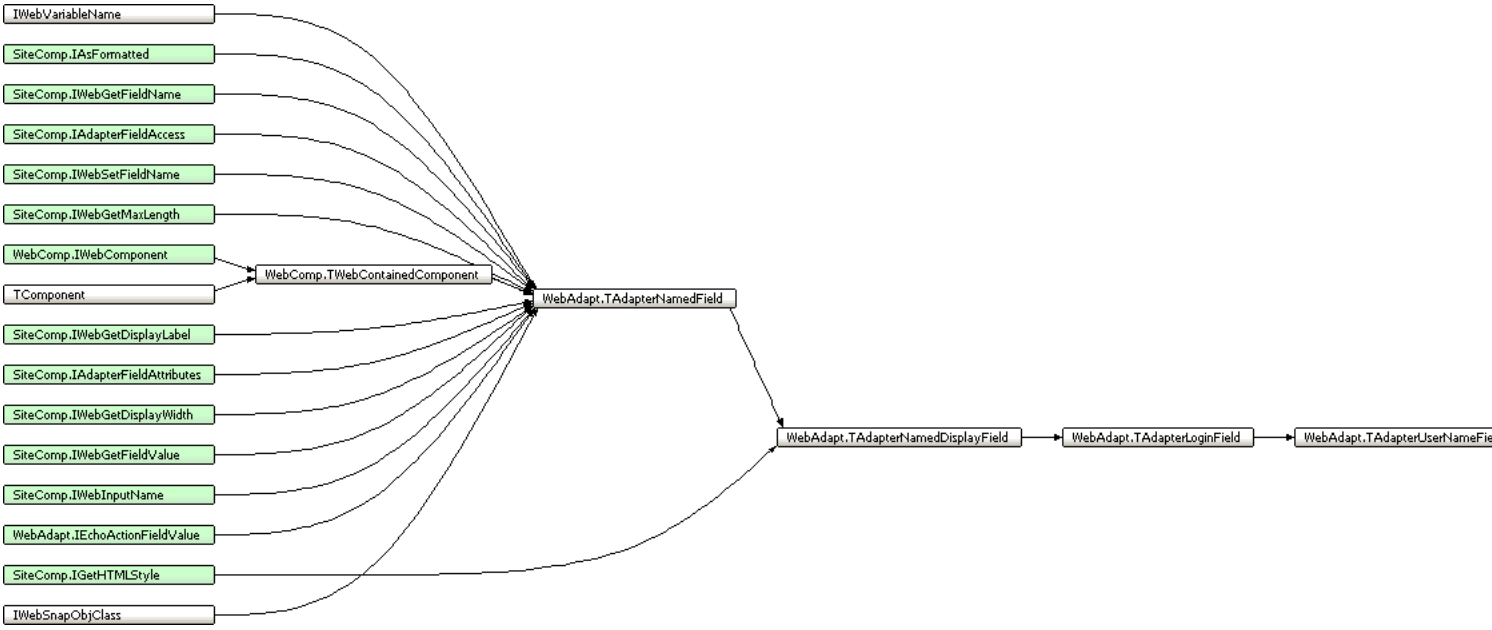
DoOnValidateValue (see page 5414)

IActionFieldValue

57.1.36 TAdapterUserNameField Class

TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.

Class Hierarchy



Pascal

```
TAdapterUserNameField = class(TAdapterLoginField);
```

C++

```
class TAdapterUserNameField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterUserNameField is added to TLoginFormAdapter (see page 5810). TAdapterUserNameField, TAdapterPasswordField (see page 5379), TLoginFormAdapterLoginAction (see page 5821) and TAdapterNextPageField (see page 5367) are used with TLoginFormAdapter (see page 5810) to generate an HTML login form. When the HTML form is submitted, TAdapterUserNameField is used to extract the end user's name from the HTTP request.

See Also

TAdapterLoginField (see page 5308)

TLoginFormAdapter (see page 5810)

TAdapterPasswordField (see page 5379)

TAdapterNextPageField (see page 5367)

TLoginFormAdapterLoginAction (see page 5821)

57.1.36.1 TAdapterUserNameField Members

The following tables list the members exposed by TAdapterUserNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5425)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.

	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.

R	Adapter (see page 5311)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5426)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5426)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5426)	Identifies the name of this field when using server-side script.

57.1.36.2 TAdapterUserNameField Methods

The methods of the TAdapterUserNameField class are listed here.

Protected Methods

	Name	Description
V	ImplGetValue (see page 5425)	Returns the value associated with this adapter field.

57.1.36.2.1 TAdapterUserNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 5357) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's user name field will be blank. When the login form is submitted, the ActionValue (see page 5356) property can be used to access the user name entered by the end-user.

See Also

ActionValue (see page 5356)

57.1.36.3 TAdapterUserNameField Properties

The properties of the TAdapterUserNameField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5426)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5426)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5426)	Identifies the name of this field when using server-side script.

57.1.36.3.1 TAdapterUserNameField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5345)

57.1.36.3.2 TAdapterUserNameField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5346)

57.1.36.3.3 TAdapterUserNameField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

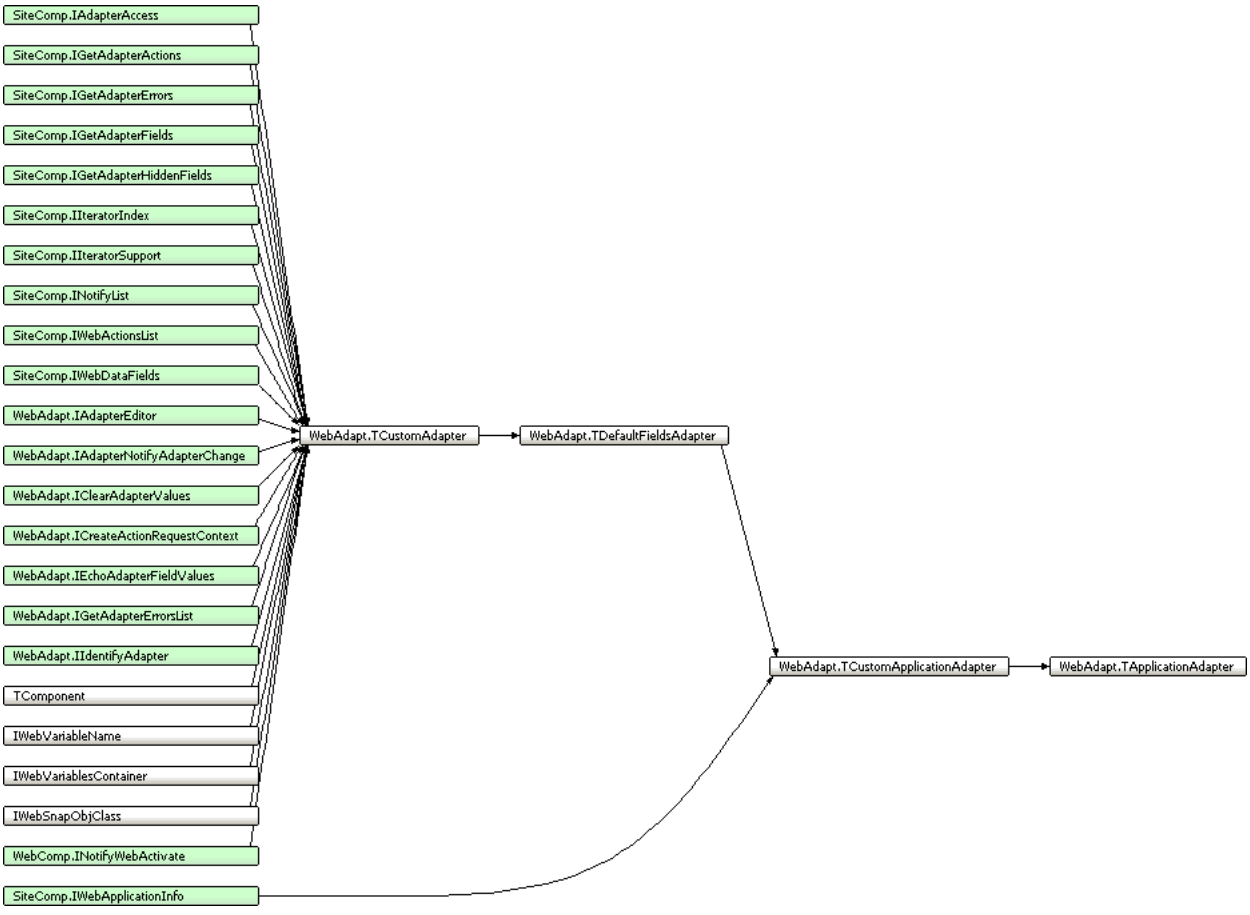
Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

57.1.37 TApplicationAdapter Class

TApplicationAdapter is a container for field and action components that are available through the Application script variable.

Class Hierarchy



Pascal

```
TApplicationAdapter = class(TCustomApplicationAdapter);
```

C++

```
class TApplicationAdapter : public TCustomApplicationAdapter;
```

File

WebAdapt

Description

By default, TApplicationAdapter contains a single field called Title that gets its value from the ApplicationTitle (see page 5432) property. The TApplicationAdapter can be customized by adding additional field and action components. When writing server-side script, use the Application variable to access fields and actions of the ApplicationAdapter. For example, <%= Application.Title %> writes the title of the application.

See Also

TAdapter (see page 5179)

TApplicationAdapter

57.1.37.1 TApplicationAdapter Members

The following tables list the members exposed by TApplicationAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.

	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationTitle method.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 5655)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 5655)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5656)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 5656)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetFieldsList (see page 5656)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.

	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.

	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetApplicationAdapter (see page 1021)	Returns the value of the ApplicationAdapter (see page 1021) property.
	GetApplicationTitle (see page 1021)	Returns the value of the ApplicationTitle (see page 1022) property.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	ApplicationAdapter (see page 1021)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5433)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5433)	Occurs after the adapter action response is generated.

	OnBeforeExecuteAction (see page 5434)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5434)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5435)	Supplies name/value pairs needed to execute an adapter action.

Published Properties

	Name	Description
	Actions (see page 5432)	Container for adapter actions components.
	ApplicationTitle (see page 5432)	Defines the title of the application.
	Data (see page 5433)	Container for adapter field components.

57.1.37.2 TApplicationAdapter Properties

The properties of the TApplicationAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5432)	Container for adapter actions components.
	ApplicationTitle (see page 5432)	Defines the title of the application.
	Data (see page 5433)	Container for adapter field components.

57.1.37.2.1 TApplicationAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.37.2.2 TApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;__property UnicodeString ApplicationTitle;
```

Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

57.1.37.2.3 TApplicationAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.37.3 TApplicationAdapter Events

The events of the TApplicationAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5433)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5433)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5434)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5434)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5435)	Supplies name/value pairs needed to execute an adapter action.

57.1.37.3.1 TApplicationAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.37.3.2 TApplicationAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.37.3.3 TApplicationAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.37.3.4 TApplicationAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.37.3.5 TApplicationAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

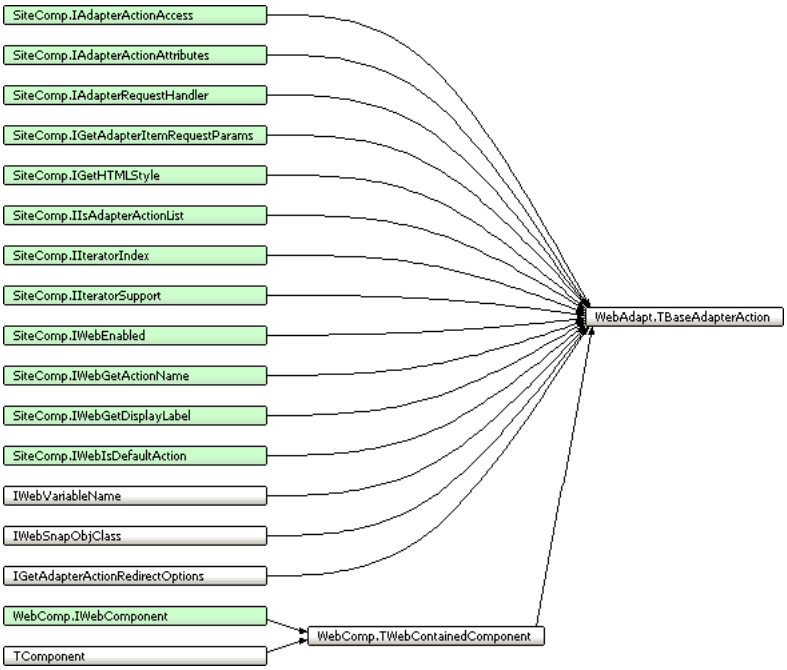
See Also

OnGetParams (see page 5466)

57.1.38 TBaseAdapterAction Class

TBaseAdapterAction is a based class for adapter action components.

Class Hierarchy



Pascal

```
TBaseAdapterAction = class(TWebContainedComponent, IWebVariableName, IWebEnabled,
IGetAdapterItemRequestParams, IWebGetActionName, IWebGetDisplayLabel, IWebSnapObjClass,
IAdapterActionAccess, IAdapterRequestHandler, IGetAdapterActionRedirectOptions, IGetHTMLStyle,
IAdapterActionAttributes, IIteratorSupport, IIteratorIndex, IIsAdapterActionList,
IWebIsDefaultAction);
```

C++

```
class TBaseAdapterAction : public TWebContainedComponent, public IWebVariableName, public
IWebEnabled, public IGetAdapterItemRequestParams, public IWebGetActionName, public
IWebGetDisplayLabel, public IWebSnapObjClass, public IAdapterActionAccess, public
IAdapterRequestHandler, public IGetAdapterActionRedirectOptions, public IGetHTMLStyle, public
IAdapterActionAttributes, public IIteratorSupport, public IIteratorIndex, public
IIsAdapterActionList, public IWebIsDefaultAction;
```

File

WebAdapt

Description

TBaseAdapterAction implements interfaces used to access an adapter action's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter action class.

See Also

TCustomAdapterAction (see page 5565)

TAdapter (see page 5179)

57.1.38.1 TBaseAdapterAction Members

The following tables list the members exposed by TBaseAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.

	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
V	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
V	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.

	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.38.2 TBaseAdapterAction Methods

The methods of the TBaseAdapterAction class are listed here.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.

	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.

	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
V	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.38.2.1 TBaseAdapterAction.CheckExecuteAccess Method

Checks end-user rights to execute this adapter action.

Pascal

```
procedure CheckExecuteAccess;
```

C++

```
__fastcall CheckExecuteAccess();
```

Description

This procedure is called to verify that the current user has rights to execute this action. An exception will be raised if access is denied.

See Also

ExecuteAccess (see page 5463)

HasExecuteAccess (see page 5451)

ExecuteActionRequest (see page 5443)

57.1.38.2.2 TBaseAdapterAction.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls this function before calling HandleRequest (see page 5451). The result of this function is stored in the WebContext (see page 5977) variable. During the course of executing this action, the object returned by CreateRequestContext will be used to access the action parameters and field values that were passed in the HTTP request to execute this action.

See Also

ImplCreateRequestContext (see page 5452)

AdapterRequest (see page 5946)

TAdapterDispatcher (see page 6122)

HandleRequest (see page 5451)

57.1.38.2.3 TBaseAdapterAction.DoAfterExecuteActionRequest Method

Calls event handlers after the adapter action is executed.

Pascal

```
procedure DoAfterExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoAfterExecuteActionRequest is called by ExecuteActionRequest (see page 5443). DoAfterExecuteActionRrequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnAfterExecuteAction event handler is called first. If it does not handle the action request then the OnAfterExecute (see page 5465) event handler is called.

See Also

OnAfterExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

DoAfterExecuteActionRequest

ExecuteActionRequest (see page 5443)

57.1.38.2.4 TBaseAdapterAction.DoAfterGetExecuteActionResponse Method

Calls event handlers after the adapter action response is formulated.

Pascal

```
procedure DoAfterGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoAfterGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5446). The parent adapter's OnAfterGetActionResponse event handler is called first. If it does not handle the action response then the OnAfterGetResponse (see page 5465) event handler is called.

See Also

OnAfterGetResponse (see page 5465)

OnAfterGetActionResponse (see page 5562)

DoBeforeGetExecuteActionResponse (see page 5442)

GetExecuteActionResponse (see page 5446)

57.1.38.2.5 TBaseAdapterAction.DoBeforeExecuteActionRequest Method

Calls event handlers before the adapter action is executed.

Pascal

```
procedure DoBeforeExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoBeforeExecuteActionRequest is called by ExecuteActionRequest (see page 5443). DoBeforeExecuteActionRequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnBeforeExecuteAction event handler is called first. If it does not handle the action request then the OnBeforeExecute (see page 5465) event handler is called. If either event handler handles the request then the IActionRequest (see page 5890)'s Handled property is set to true.

See Also

OnBeforeExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

DoAfterExecuteActionRequest (see page 5441)

ExecuteActionRequest (see page 5443)

57.1.38.2.6 TBaseAdapterAction.DoBeforeGetExecuteActionResponse Method

Calls event handlers before the adapter action response is formulated.

Pascal

```
procedure DoBeforeGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoBeforeGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5446). The parent adapter's OnBeforeGetActionResponse event handler is called first. If it does not handle the action response then the OnBeforeGetResponse (see page 5466) event handler is called. If either event handler handles the action response then the IActionResponse's Handled property is set to true.

See Also

OnBeforeGetResponse (see page 5466)

OnBeforeGetActionResponse (see page 5563)

DoAfterGetExecuteActionResponse (see page 5441)

GetExecuteActionResponse (see page 5446)

57.1.38.2.7 TBaseAdapterAction.EndIterator Method

Implements IIteratorSupport (see page 996)'s EndIterator method.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

IIteratorSupport (see page 996) is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 5280) is an example of an adapter action that supports multiple commands. EndIterator calls ImplNextIteration (see page 5458).

See Also

IIteratorSupport (see page 996)

IIteratorIndex (see page 994)

enditerator (see page 997)

ImplEndIterator (see page 5452)

57.1.38.2.8 TBaseAdapterAction.ExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall ExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ExecuteActionRequest is called by ImplHandleRequest (see page 5456) when a request to execute this adapter action is found in the HTTP request. ExecuteActionRequest first calls CheckExecuteAccess (see page 5440). CheckExecuteAccess (see page 5440) raises an exception if the end-user does not have execute rights. ExecuteActionRequest then calls, in turn, DoBeforeExecuteActionRequest (see page 5442), ImplExecuteActionRequest (see page 5452), and DoAfterExecuteActionRequest (see page 5441).

See Also

DoBeforeExecuteActionRequest (see page 5442)

DoAfterExecuteActionRequest (see page 5441)

CheckExecuteAccess (see page 5440)

ImplExecuteActionRequest (see page 5452)

ImplHandleRequest (see page 5456)

57.1.38.2.9 TBaseAdapterAction.GetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

GetActionName is called to retrieve the name used to access this adapter field with server-side script. GetActionName calls ImplGetActionName (see page 5453).

See Also

ActionName (see page 5462)

ImplGetActionName (see page 5453)

57.1.38.2.10 TBaseAdapterAction.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

See Also

Adapter (see page 5462)

57.1.38.2.11 TBaseAdapterAction.GetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
Params);
```

Description

The name/value pairs provided by this method will written to the HTTP response when the HTML page is generated and passed back in the HTTP request when this adapter action is to be executed. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted.

See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.38.2.12 TBaseAdapterAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 5453)

57.1.38.2.13 TBaseAdapterAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

When the display label for the action matches this value then the display label property does not need to be stored in the form file.

57.1.38.2.14 TBaseAdapterAction.GetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall UnicodeString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display a button or hyperlink for this adapter action. The value returned by this function can be accessed with server-side script using the DisplayLabel (see page 5462) property. GetDisplayLabel calls ImplGetDisplayLabel (see page 5453).

See Also

DisplayLabel (see page 5462)

ImplGetDisplayLabel (see page 5453)

57.1.38.2.15 TBaseAdapterAction.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle (see page 973)'s GetDisplayStyle method by translating the result value of GetDisplayStyleType (see page 5446) into a string.

See Also

GetDisplayStyle (see page 974)

GetDisplayStyleType (see page 5446)

57.1.38.2.16 TBaseAdapterAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaButton, indicating that this adapter action should be displayed using a submit button. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 5445)

57.1.38.2.17 TBaseAdapterAction.GetErrors Method

Pascal

```
function GetErrors: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetErrors();
```

Description

This is GetErrors, a member of class TBaseAdapterAction.

57.1.38.2.18 TBaseAdapterAction.GetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure GetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall GetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

GetExecuteActionResponse is called by ImplHandleRequest (see page 5456) after executing an adapter action specified by the HTTP request. GetExecuteActionResponse calls, in turn, DoBeforeGetExecuteActionResponse (see page 5442), ImplGetExecuteActionResponse (see page 5454), and DoAfterGetExecuteActionResponse (see page 5441).

See Also

IActionRequest

IActionResponse

DoBeforeGetExecuteActionResponse (see page 5442)

DoAfterGetExecuteActionResponse (see page 5441)

ImplGetExecuteActionResponse (see page 5454)

ImplHandleRequest (see page 5456)

57.1.38.2.19 TBaseAdapterAction.GetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function GetFailureOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetFailureOptions();
```

Description

Set RedirectOptions (see page 5464) when an HTTP redirect should be performed after execution of this adapter action fails. GetFailureOptions returns the result of ImplGetFailureOptions (see page 5454).

See Also

ImplGetFailureOptions (see page 5454)

GetSuccessOptions (see page 5449)

RedirectOptions (see page 5464)

57.1.38.2.20 TBaseAdapterAction.GetInputStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle (see page 973)'s GetInputType method by returning the result of GetDisplayStyle (see page 5445). This method must be implemented as part of IGetHTMLStyle (see page 973). GetInputStyle is treated as a

synonym of GetDisplayStyle (see page 5445).

See Also

GetInputStyle (see page 974)

GetDisplayStyle (see page 5445)

57.1.38.2.21 TBaseAdapterAction.GetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 5280) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 5830). GetIsAdapterActionList calls ImplGetIsAdapterActionList (see page 5454).

See Also

ImplGetIsAdapterActionList (see page 5454)

TAdapterGotoPageAction (see page 5280)

57.1.38.2.22 TBaseAdapterAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 5280) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 5830). GetItemCount returns 1. GetItemCount may be overridden by a descendant class.

See Also

ItemCount (see page 5463)

ItemIndex (see page 5464)

TAdapterGotoPageAction (see page 5280)

GetIsAdapterActionList (see page 5448)

57.1.38.2.23 TBaseAdapterAction.GetIteratorIndex Method

Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

IIteratorIndex (see page 994) and IIteratorSupport (see page 996) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. GetIteratorIndex returns the index of the current command. GetIteratorIndex calls ImplGetIteratorIndex (see page 5455).

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

GetIteratorIndex (see page 995)

ImplGetIteratorIndex (see page 5455)

57.1.38.2.24 TBaseAdapterAction.GetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function GetSuccessOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetSuccessOptions();
```

Description

Set RedirectOptions (see page 5464) when an HTTP redirect should be performed after executing an action. GetSuccessOptions returns the result of ImplGetSuccessOptions (see page 5455).

See Also

ImplGetSuccessOptions (see page 5455)

GetFailureOptions (see page 5447)

RedirectOptions (see page 5464)

57.1.38.2.25 TBaseAdapterAction.GetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter action with server-side script. GetVariableName calls ImplGetVariableName (see page 5455).

See Also

ImplGetVariableName (see page 5455)

57.1.38.2.26 TBaseAdapterAction.GetViewMode Method

Returns a string value to suggest a view mode for this adapter action.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle (see page 973)'s GetViewMode method by translating the result value of GetViewModeType (see page 5450) into a string.

See Also

GetViewMode (see page 975)

GetViewModeType (see page 5450)

57.1.38.2.27 TBaseAdapterAction.GetViewModeType Method

Suggests a view mode for this adapter action.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const UnicodeString
AAdapterMode);
```

Description

GetViewModeType returns htmlvmInput.

See Also

ReadOnly (see page 5339)

GetViewMode (see page 5336)

57.1.38.2.28 TBaseAdapterAction.GetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The state of the parent adapter may dictate whether a button or hyperlink for a particular adapter action should not be shown. For example, TDataSetAdapterApplyRowAction can be configured to return false when the parent TDataSetAdapter is in browse

mode. GetVisible calls ImplGetVisible (see page 5456). Server-side script can access the value returned by GetVisible using the Visible property of an adapter action object.

See Also

ImplGetVisible (see page 5456)

57.1.38.2.29 TBaseAdapterAction.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

57.1.38.2.30 TBaseAdapterAction.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response. HandleRequest calls ImplHandleRequest (see page 5456).

See Also

CreateRequestContext (see page 5440)

57.1.38.2.31 TBaseAdapterAction.HasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanExecute property. HasExecuteAccess calls ImplHasExecuteAccess (see page 5456).

See Also

CheckExecuteAccess (see page 5440)

ImplHasExecuteAccess (see page 5456)

57.1.38.2.32 TBaseAdapterAction.ImplCreateRequestContext Method

Creates an object that describes the action parameters and field values.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext is called by CreateRequestContext (see page 5440). ImplCreateRequestContext returns the result of TCustomAdapter (see page 5519)'s CreateActionRequestContext method.

See Also

CreateRequestContext (see page 5440)

CreateActionRequestContext (see page 5528)

57.1.38.2.33 TBaseAdapterAction.ImplEndIterator Method

Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer);
```

C++

```
__fastcall ImplEndIterator(void * OwnerData);
```

Description

ImplEndIterator is called by EndIterator (see page 5443). ImplEndIterator resets internal variables.

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

EndIterator (see page 5443)

57.1.38.2.34 TBaseAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 5443). ImplExecuteActionRequest must be

overridden by a descendant class.

See Also

IActionRequest

IActionResponse

ExecuteActionRequest (see page 5443)

57.1.38.2.35 TBaseAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetActionName();
```

Description

This function is called by GetActionName (see page 5444). ImplGetActionName returns the result of GetDefaultActionName (see page 5445). Descendants may override this function.

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.38.2.36 TBaseAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5444). ImplGetAdapterItemRequestParams calls the OnGetActionParams event handler of the parent adapter and then calls the OnGetParams (see page 5466) event handler.

See Also

GetAdapterItemRequestParams (see page 5444)

OnGetActionParams (see page 5564)

OnGetParams (see page 5466)

57.1.38.2.37 TBaseAdapterAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5462) property.

See Also

DisplayLabel (see page 5462)

57.1.38.2.38 TBaseAdapterAction.ImplGetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure ImplGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplGetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5446). ImplGetExecuteActionResponse sets properties of IActionResponse to identify the response to this action request.

See Also

GetExecuteActionResponse (see page 5446)

57.1.38.2.39 TBaseAdapterAction.ImplGetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function ImplGetFailureOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetFailureOptions();
```

Description

ImplGetFailureOptions returns an empty set.

See Also

GetFailureOptions (see page 5447)

57.1.38.2.40 TBaseAdapterAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 5448). ImplGetIsAdapterActionList returns false. ImplGetIsAdapterActionList may be overridden by descendant classes.

See Also

GetIsAdapterActionList (see page 5448)

ImplGetIsAdapterActionList (see page 5607)

57.1.38.2.41 TBaseAdapterAction.ImplGetIteratorIndex Method

Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex is called by GetIteratorIndex (see page 5448). ImplGetIteratorIndex returns the value of an internal iteration index.

See Also

GetIteratorIndex (see page 995)

GetIteratorIndex (see page 5448)

57.1.38.2.42 TBaseAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 5464) property.

See Also

GetSuccessOptions (see page 5449)

RedirectOptions (see page 5464)

57.1.38.2.43 TBaseAdapterAction.ImplGetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetVariableName: string;
```

C++

```
__fastcall UnicodeString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 5449). GetVariableName (see page 5449) returns the result of GetActionName (see page 5444).

See Also

GetVariableName (see page 5449)

GetActionName (see page 5444)

57.1.38.2.44 TBaseAdapterAction.ImplGetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

ImplGetVisible is called by GetVisible (see page 5450). ImplGetVisible returns true. ImplGetVisible may be overridden by descendant components.

See Also

GetVisible (see page 5450)

57.1.38.2.45 TBaseAdapterAction.ImplHandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 5451). ImplHandleRequest calls ExecuteActionRequest (see page 5443) followed by GetExecuteActionResponse (see page 5446).

See Also

ExecuteActionRequest (see page 5443)

GetExecuteActionResponse (see page 5446)

57.1.38.2.46 TBaseAdapterAction.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 5451) to check the end-user's right to execute the adapter action. First, the parent adapter's HasExecuteAccess (see page 5451) method is called. If the adapter allows execute access and the ExecuteAccess (see page 5463) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ExecuteAccess (see page 5463) property value.

See Also

HasExecuteRights (see page 5496)

ExecuteAccess (see page 5463)

HasExecuteAccess (see page 5451)

hasexecuteaccess (see page 5539)

57.1.38.2.47 TBaseAdapterAction.ImplInIterator Method

Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

ImplInIterator is called by InIterator (see page 5459). ImplInIterator returns the value of an internal flag.

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

InIterator (see page 995)

ImplInIterator

57.1.38.2.48 TBaseAdapterAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5459). ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 5459)

ImplIsDefaultAction (see page 5608)

57.1.38.2.49 TBaseAdapterAction.ImplNextIteration Method

Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration is called by NextIteration (see page 5460). ImplNextIteration increments an internal iteration index. true is returned if the iteration index is within the bounds defined by ItemCount (see page 5463).

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

NextIteration (see page 5460)

57.1.38.2.50 TBaseAdapterAction.ImplStartIterator Method

Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator is called by StartIterator (see page 5461). ImplStartIterator initializes the internal iteration index. ImplStartIterator returns true because it is assumed that there is at least one command associated with this adapter action.

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

startiterator (see page 998)

StartIterator (see page 5461)

57.1.38.2.51 TBaseAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 5461). ImplWebEnabled returns true. Descendant classes may override this method.

See Also

WebEnabled (see page 5461)

57.1.38.2.52 TBaseAdapterAction.InIterator Method

Implements IIteratorIndex (see page 994)'s InIterator method.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

IIteratorIndex (see page 994) and IIteratorSupport (see page 996) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. InIterator calls ImplInIterator (see page 5457).

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

InIterator (see page 995)

ImplInIterator (see page 5457)

57.1.38.2.53 TBaseAdapterAction.IsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

Some adapter components have built in support for specific types of adapter actions. These actions are often added to the adapter component by default unless the developer explicitly adds one or more adapter actions. IsDefaultAction is called by the adapter while adding default actions to determine whether this adapter action should be added. ADisplay identifies characteristics of the adapter component. IsDefaultAction calls ImplIsDefaultAction (see page 5457).

See Also

ImplIsDefaultAction (see page 5457)

57.1.38.2.54 TBaseAdapterAction.NextIteration Method

Implements IIteratorSupport (see page 996)'s NextIteration method.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

IIteratorSupport (see page 996) is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 5280) is an example of an adapter action that supports multiple commands. NextIterations returns the result of ImplNextIteration (see page 5458).

See Also

IIteratorSupport (see page 996)

IIteratorIndex (see page 994)

ImplNextIteration (see page 5458)

NextIteration (see page 997)

57.1.38.2.55 TBaseAdapterAction.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TBaseAdapterAction (see page 5435) clears its cached reference to the parent adapter.

57.1.38.2.56 TBaseAdapterAction.SetDisplayLabel Method

Set the value of the DisplayLabel (see page 5462) property.

Pascal

```
procedure SetDisplayLabel(const Value: string);
```

C++

```
__fastcall SetDisplayLabel(const UnicodeString Value);
```

Description

SetDisplayLabel sets the value of the DisplayLabel (see page 5462) property.

See Also

DisplayLabel (see page 5462)

57.1.38.2.57 TBaseAdapterAction.StartIterator Method

Implements IIteratorSupport (see page 996)'s StartIterator method.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

IIteratorIndex (see page 994) and IIteratorSupport (see page 996) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. StartIterator calls ImplImplInIterator.

See Also

IIteratorIndex (see page 994)

IIteratorSupport (see page 996)

startiterator (see page 998)

ImplStartIterator (see page 5458)

57.1.38.2.58 TBaseAdapterAction.WebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

The state of the parent adapter may dictate when a button or hyperlink for a particular adapter action should not be enabled. For example, TDataSetAdapterPrevRowAction returns False when the dataset is positioned at BOF. WebEnabled calls ImplWebEnabled (see page 5458). Server-side script can access the value returned by WebEnabled using the Enabled property of an adapter action object.

See Also

ImplWebEnabled (see page 5458)

57.1.38.3 TBaseAdapterAction Properties

The properties of the TBaseAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.

Public Properties

	Name	Description
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.

57.1.38.3.1 TBaseAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

GetActionName (see page 5444)

57.1.38.3.2 TBaseAdapterAction.Adapter Property

Provides access to the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter component.

57.1.38.3.3 TBaseAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.38.3.4 TBaseAdapterAction.Errors Property

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

This is Errors, a member of class TBaseAdapterAction.

57.1.38.3.5 TBaseAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6759)

hasexecuteaccess (see page 5539)

57.1.38.3.6 TBaseAdapterAction.ItemCount Property

Returns the number of items defined by an adapter action.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

An adapter action may have a list of commands. The ItemCount property is implemented by GetItemCount (see page 5448). The TAdapterGotoPageAction (see page 5280) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 5280) displays a different page of the parent TPagedAdapter (see page 5830).

See Also

GetItemCount (see page 5448)

ItemIndex (see page 5464)

TAdapterGotoPageAction (see page 5280)

GetIsAdapterActionList (see page 5448)

57.1.38.3.7 TBaseAdapterAction.ItemIndex Property

Returns the index of the current item of an adapter action.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

An adapter action may have a list of commands. The ItemIndex indicates the current item in the list. The TAdapterGotoPageAction (see page 5280) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 5280) displays a different page of the parent TPagedAdapter (see page 5830).

See Also

ItemCount (see page 5463)

TAdapterGotoPageAction (see page 5280)

GetIsAdapterActionList (see page 5448)

57.1.38.3.8 TBaseAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

57.1.38.4 TBaseAdapterAction Events

The events of the TBaseAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.

	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

57.1.38.4.1 TBaseAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest (see page 5441).

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.38.4.2 TBaseAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse

OnBeforeGetActionResponse (see page 5563)

57.1.38.4.3 TBaseAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest (see page 5442).

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.38.4.4 TBaseAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.38.4.5 TBaseAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

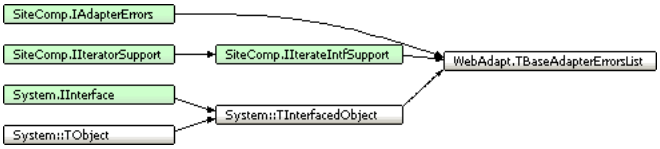
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.39 TBaseAdapterErrorsList Class

TBaseAdapterErrorsList is the base class for adapter error object managers.

Class Hierarchy



Pascal

```
TBaseAdapterErrorsList = class(TInterfacedObject, IAdapterErrors, IIterateIntfSupport);
```

C++

```
class TBaseAdapterErrorsList : public TInterfacedObject, public IAdapterErrors, public
IIterateIntfSupport;
```

File

WebAdapt

Description

TBaseAdapterErrorsList is the base class for objects that manage lists of adapter error objects. Do not instantiate TBaseAdapterErrorsList directly. Use TAdapterErrorsList (see page 5248).

See Also

TAdapterErrorsList (see page 5248)

IAdapterError (see page 934)

57.1.39.1 TBaseAdapterErrorsList Members

The following tables list the members exposed by TBaseAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	EndIterator (see page 5471)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 5471)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5471)	Returns a particular error.
	GetErrorCount (see page 5471)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5472)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5472)	Indicates whether the errors list contains objects associated with a particular adapter field.

	ImplClearErrors (see page 5472)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
	ImplDefineLabel (see page 5472)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
	ImplGetError (see page 5473)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
	ImplGetErrorCount (see page 5473)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
	ImplGetObjectErrors (see page 5473)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
	ImplHasObjectErrors (see page 5473)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5474)	Advances to the objects next item.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5474)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddError (see page 5469)	This is the overview for the AddError method overload.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 5470)	Clear error list.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	DefineLabel (see page 5470)	Define error label for field.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.

	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	ErrorCount (see page 5474)	Number of error objects contained.
	Errors (see page 5475)	Access to error objects.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

57.1.39.2 TBaseAdapterErrorsList Methods

The methods of the TBaseAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 5471)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 5471)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5471)	Returns a particular error.
	GetErrorCount (see page 5471)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5472)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5472)	Indicates whether the errors list contains objects associated with a particular adapter field.
	ImplClearErrors (see page 5472)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
	ImplDefineLabel (see page 5472)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
	ImplGetError (see page 5473)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
	ImplGetErrorCount (see page 5473)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
	ImplGetObjectErrors (see page 5473)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
	ImplHasObjectErrors (see page 5473)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5474)	Advances to the objects next item.
	StartIterator (see page 5474)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	AddError (see page 5469)	This is the overview for the AddError method overload.
	ClearErrors (see page 5470)	Clear error list.
	DefineLabel (see page 5470)	Define error label for field.

57.1.39.2.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TBaseAdapterErrorsList.AddError (Exception, string) (see page 5470)	Add error to list.
	TBaseAdapterErrorsList.AddError (Exception, TObject) (see page 5470)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, string, Integer) (see page 5470)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, TObject, Integer) (see page 5470)	Add error to list.

57.1.39.2.1.1 TBaseAdapterErrorsList.AddError Method (Exception, TObject)

Add error to list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); virtual; overload;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
virtual; overload;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); virtual;
overload;
procedure AddError(AException: Exception; AObject: TObject); virtual; overload;
```

C++

```
virtual __fastcall AddError(Exception AException, const UnicodeString AObjectName = '');
virtual __fastcall AddError(const UnicodeString AMessage, const UnicodeString AObjectName =
'', int AID = 0);
virtual __fastcall AddError(const UnicodeString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the list maintained by the TBaseAdapterErrorsList (see page 5467) instance. As implemented in TBaseAdapterErrorsList (see page 5467), AddError does nothing. This method must be implemented by descendant classes.

See Also

AddError (see page 5252)

57.1.39.2.2 TBaseAdapterErrorsList.ClearErrors Method

Clear error list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clear errors deletes all error information maintained by the TBaseAdapterErrorsList (see page 5467) instance.

See Also

AddError (see page 5252)

57.1.39.2.3 TBaseAdapterErrorsList.DefineLabel Method

Define error label for field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

DefineLabel defines a string label that represents a dataset field in error message. AName is the the field name. ALabel is the label.

See Also

AddError (see page 5252)

57.1.39.2.4 TBaseAdapterErrorsList.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TBaseAdapterErrorsList.

57.1.39.2.5 TBaseAdapterErrorsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.

57.1.39.2.6 TBaseAdapterErrorsList.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 934)

57.1.39.2.7 TBaseAdapterErrorsList.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

57.1.39.2.8 TBaseAdapterErrorsList.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const UnicodeString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 939)

57.1.39.2.9 TBaseAdapterErrorsList.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const UnicodeString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 939)

57.1.39.2.10 TBaseAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; virtual;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TBaseAdapterErrorsList.

57.1.39.2.11 TBaseAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); virtual;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.

57.1.39.2.12 TBaseAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; virtual;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TBaseAdapterErrorsList.

57.1.39.2.13 TBaseAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.

57.1.39.2.14 TBaseAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const UnicodeString AName);
```

Description

This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.

57.1.39.2.15 TBaseAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const UnicodeString AName);
```

Description

This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.

57.1.39.2.16 TBaseAdapterErrorsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 998)

enditerator (see page 997)

57.1.39.2.17 TBaseAdapterErrorsList.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TBaseAdapterErrorsList.

57.1.39.3 TBaseAdapterErrorsList Properties

The properties of the TBaseAdapterErrorsList class are listed here.

Public Properties

	Name	Description
R	ErrorCount (see page 5474)	Number of error objects contained.
R	Errors (see page 5475)	Access to error objects.

57.1.39.3.1 TBaseAdapterErrorsList.ErrorCount Property

Number of error objects contained.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the number of error objects available.

See Also

Errors (see page 5475)

57.1.39.3.2 TBaseAdapterErrorsList.Errors Property

Access to error objects.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

Errors returns a specific error object.

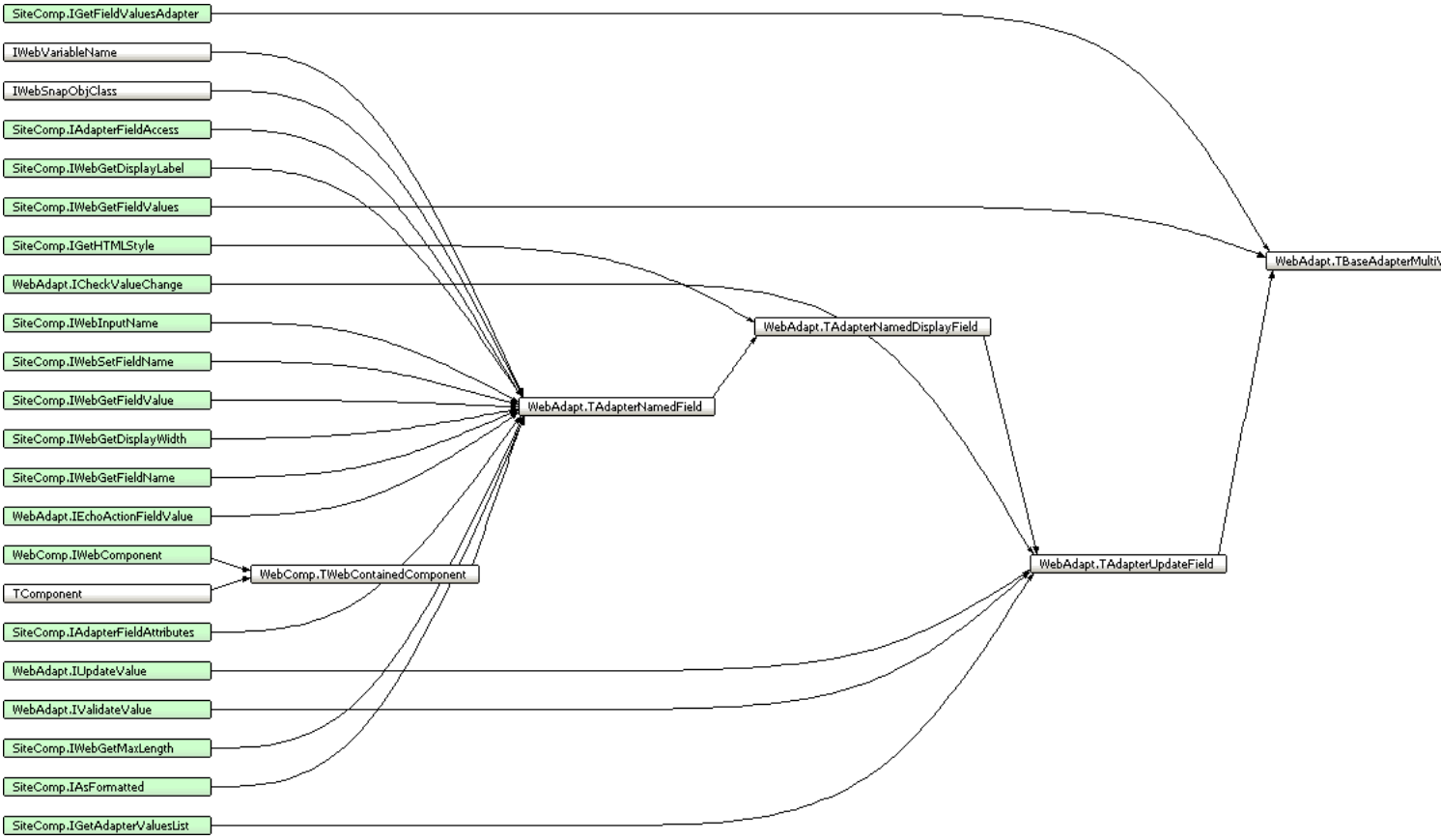
See Also

ErrorCount (see page 5474)

57.1.40 TBaseAdapterMultiValueField Class

TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.

Class Hierarchy



Pascal

```
TBaseAdapterMultiValueField = class(TAdapterUpdateField, IGetFieldValuesAdapter,
IWebGetFieldValues);
```

C++

```
class TBaseAdapterMultiValueField : public TAdapterUpdateField, public IGetFieldValuesAdapter,
public IWebGetFieldValues;
```

File

WebAdapt

Description

TBaseAdapterMultiValueField is used to represent a data value that can have multiple values. Server-side script is used to generate a form displaying this field. Typically, the HTML form will contain group check boxes or a select list supporting multiple selection. When an Adapter (see page 5356) action is executed, TBaseAdapterMultiValueField is used to retrieve the submitted field value(s) from the HTTP request.

See Also

TCustomAdapterMultiValueField (see page 5632)

TCustomAdapterFileField (see page 5590)

57.1.40.1 TBaseAdapterMultiValueField Members

The following tables list the members exposed by TBaseAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
	EndValueIterator (see page 5481)	Completes iteration.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5482)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5482)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.

	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5484)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5418)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.

R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5480)	Constructor for TBaseAdapterMultiValueField (see page 5475).
V	Destroy (see page 5480)	Destructor for TBaseAdapterMultiValueField (see page 5475).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.

	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.40.2 TBaseAdapterMultiValueField.Create Constructor

Constructor for TBaseAdapterMultiValueField (see page 5475).

Pascal

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

C++

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

Description

Creates an instance of TFieldValuesAdapter (see page 5789) and calls the inherited creator method. TFieldValuesAdapter (see page 5789) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 5789)

ImplGetFieldValuesAdapter (see page 5483)

Destroy (see page 5480)

57.1.40.3 TBaseAdapterMultiValueField.Destroy Destructor

Destructor for TBaseAdapterMultiValueField (see page 5475).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterMultiValueField();
```

Description

Destroys the instance of TFieldValuesAdapter (see page 5789) and calls the inherited destructor. TFieldValuesAdapter (see page 5789) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 5789)

ImplGetFieldValuesAdapter (see page 5483)

Create (see page 5480)

57.1.40.4 TBaseAdapterMultiValueField Methods

The methods of the TBaseAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
	EndValueIterator (see page 5481)	Completes iteration.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
V	GetValueCount (see page 5482)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5482)	Returns one of the values associated with the adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.
V	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
V	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
V	ImplHasValue (see page 5484)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.

57.1.40.4.1 TBaseAdapterMultiValueField.EndValueIterator Method

Completes iteration.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

The EndValueIterator procedure is called when iteration has completed.

See Also

NextValueIteration (see page 5484)

StartValueIterator (see page 5484)

57.1.40.4.2 TBaseAdapterMultiValueField.GetCurrentValue Method

Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue returns the result of ImplGetCurrentValue (see page 5483).

See Also

GetCurrentValue (see page 1035)

ImplGetCurrentValue (see page 5483)

57.1.40.4.3 TBaseAdapterMultiValueField.GetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

Description

GetFieldValuesAdapter is called when server-side script references the Values property of this adapter field. GetFieldValuesAdapter calls ImplGetFieldValuesAdapter (see page 5483).

See Also

ImplGetFieldValuesAdapter (see page 5483)

57.1.40.4.4 TBaseAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns 0. Descendants of TBaseAdapterMultiValueField (see page 5475) override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValues (see page 5482)

57.1.40.4.5 TBaseAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns Unassigned. Descendants override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValueCount (see page 5482)

57.1.40.4.6 TBaseAdapterMultiValueField.HasValue Method

Implements IWebGetFieldValues (see page 1034)' HasValue method.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

HasValue returns the result of ImplHasValue (see page 5484).

See Also

HasValue (see page 1035)

ImplHasValue (see page 5484)

57.1.40.4.7 TBaseAdapterMultiValueField.ImplGetCurrentValue Method

Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.

Pascal

```
function ImplGetCurrentValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetCurrentValue();
```

Description

ImplGetCurrentValue is called by GetCurrentValue (see page 5481). ImplGetCurrentValue calls GetValues (see page 5482) with the value of the internal iterator index.

See Also

GetCurrentValue (see page 5481)

GetValues (see page 5482)

57.1.40.4.8 TBaseAdapterMultiValueField.ImplGetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function ImplGetFieldValuesAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetFieldValuesAdapter();
```

Description

ImplGetFieldValuesAdapter returns an instance of TFieldValuesAdapter (see page 5789).

See Also

TFieldValuesAdapter (see page 5789)

GetFieldValuesAdapter (see page 5482)

Create (see page 5480)

Destroy (see page 5480)

57.1.40.4.9 TBaseAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue is called by HasValue (see page 5483). ImplHasValue returns False (indicating that there is no matching value). Descendants override this method.

See Also

HasValue (see page 5483)

57.1.40.4.10 TBaseAdapterMultiValueField.NextValueIteration Method

Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration increments an internal iterator index. false is returned if the iterator index is not less than the GetValueCount (see page 5482) .

See Also

NextValueIteration (see page 1036)

GetValueCount (see page 5482)

57.1.40.4.11 TBaseAdapterMultiValueField.StartValueIterator Method

Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator initializes an internal iterator index to zero. false is returned if GetValueCount (see page 5482) is zero.

See Also

StartValueIterator (see page 1036)

GetValueCount (see page 5482)

57.1.41 TBaseEndUserAdapter Class

TBaseEndUserAdapter contains user information.

Class Hierarchy

Pascal

```
TBaseEndUserAdapter = class(TDefaultFieldsAdapter, IWebEndUser);
```

C++

```
class TBaseEndUserAdapter : public TDefaultFieldsAdapter, public IWebEndUser;
```

File

WebAdapt

Description

TBaseEndUserAdapter contains information about a user such as their name, access rights, and whether they are logged in. TBaseEndUserAdapter has default fields to represent the end user's display name and login state. It has default actions for logging a user in and logging a user out. End user adapters are responsible validating end user access. For example, when the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access the page.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

57.1.41.1 TBaseEndUserAdapter Members

The following tables list the members exposed by TBaseEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5492)	Starts the server-side login process.
V	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
V	ExecuteLogout (see page 5493)	Performs server-side logout.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.

	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5493)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5494)	Gets the display name of the end user.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5495)	Determines if the user is logged in
V	GetLoggedInValue (see page 5495)	Determines if the user is logged in.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
V	GetUserID (see page 5496)	Gets the UserID from this TBaseEndUserAdapter (see page 5485).
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
V	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
V	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
V	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.

	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5501)	Processes the login for the given user.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5502)	Logs in a specified user.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5502)	Generates the login page.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.

	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5503)	Gets the user list.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5503)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 5504)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.

	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
	LoginPage (see page 5503)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.41.2 TBaseEndUserAdapter Methods

The methods of the TBaseEndUserAdapter class are listed here.

Protected Methods

	Name	Description
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
V	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
V	ExecuteLogin (see page 5492)	Starts the server-side login process.
V	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
V	ExecuteLogout (see page 5493)	Performs server-side logout.
	GetDisplayName (see page 5493)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5494)	Gets the display name of the end user.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.

	GetLoggedIn (see page 5495)	Determines if the user is logged in
V	GetLoggedInValue (see page 5495)	Determines if the user is logged in.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
V	GetUserID (see page 5496)	Gets the UserID from this TBaseEndUserAdapter (see page 5485).
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
V	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
V	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
V	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
V	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
V	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).
V	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).
V	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
V	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
V	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
V	ImplLogIn (see page 5501)	Processes the login for the given user.
V	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	LogIn (see page 5502)	Logs in a specified user.
	ProduceLoginPage (see page 5502)	Generates the login page.
	UserList (see page 5503)	Gets the user list.

57.1.41.2.1 TBaseEndUserAdapter.CanExecuteLoginForm Method

Determines if there is a static login page.

Pascal

```
function CanExecuteLoginForm: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLoginForm();
```

Description

The CanExecuteLoginForm calls CanProduceLoginPage (see page 5492) to determine if there is a static login page, or if one can be generated. Then if the user is not already logged in, this function returns true. If the login page cannot be produced or the user is already logged in, then this function returns false.

See Also

CanProduceLoginPage (see page 5492)

57.1.41.2.2 TBaseEndUserAdapter.CanExecuteLogout Method

Determines if the user is logged in.

Pascal

```
function CanExecuteLogout: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLogout();
```

Description

The CanExecuteLogout function calls ImplGetLoggedIn (see page 5499) to determine if the user is logged in. If the user is logged in, this function returns true.

See Also

ImplGetLoggedIn (see page 5499)

57.1.41.2.3 TBaseEndUserAdapter.CanProduceLoginPage Method

Determines whether there is a static login page or an automatically produced login page.

Pascal

```
function CanProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanProduceLoginPage();
```

Description

The CanProduceLoginPage returns true if there is a static login page, or if there is an event handler defined to automatically produce the login page.

See Also

OnProduceLoginPage (see page 5504)

57.1.41.2.4 TBaseEndUserAdapter.DoOnHasRights Method

Determines whether the user has view rights.

Pascal

```
function DoOnHasRights(AEvent: TEndUserRightsEvent; const ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoOnHasRights(TEndUserRightsEvent AEvent, const UnicodeString
ARights);
```

Description

The DoOnHasRights function is called from ImplHasViewRights (see page 5501). It matches the given access rights to the rights in the user list and returns true if the user has view rights.

AEvent is the event handler that DoOnHasRights should call first to check access rights.

ARights is a string that represents the rights the user needs.

See Also

ImplHasViewRights (see page 5501)

HasViewRights (see page 5497)

57.1.41.2.5 TBaseEndUserAdapter.ExecuteLogin Method

Starts the server-side login process.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

The ExecuteLogin function is called from sever-side script to start the login process. This base class version of the function is empty. A customized overridden version of this function can be written to do specific login processing for the application.

See Also

ImplProduceLoginPage (see page 5501)

57.1.41.2.6 TBaseEndUserAdapter.ExecuteLoginForm Method

Creates the HTML login page.

Pascal

```
procedure ExecuteLoginForm; virtual;
```

C++

```
virtual __fastcall ExecuteLoginForm();
```

Description

The ExecuteLoginForm function is called from server-side script to create the HTML login page. This function calls ImplProduceLoginPage (see page 5501) to make the page producer generate the login page.

See Also

ImplProduceLoginPage (see page 5501)

57.1.41.2.7 TBaseEndUserAdapter.ExecuteLogout Method

Performs server-side logout.

Pascal

```
procedure ExecuteLogout; virtual;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

The ExecuteLogout function is called from server-side script to perform the logout process. This base class version of the function is empty. A customized overridden version of this function can be written to do specific logout processing for the application.

See Also

CanExecuteLogout (see page 5491)

57.1.41.2.8 TBaseEndUserAdapter.GetDisplayName Method

Retrieves the display name string.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall UnicodeString GetDisplayName();
```

Description

The GetDisplayName function is called from server-side script to get the display name string. It calls ImplGetDisplayName (see page 5498) to get the display name for this TBaseEndUserAdapter (see page 5485).

See Also

ImplGetDisplayName (see page 5498)

GetDisplayNameValue (see page 5494)

57.1.41.2.9 TBaseEndUserAdapter.GetDisplayNameValue Method

Gets the display name of the end user.

Pascal

```
function GetDisplayNameValue: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

The GetDisplayNameValue is called from server-side script to get the display name of the end user. This base class version of the function returns an empty string. A customized overridden version of this function could be written to return the display name for the end user in your application.

See Also

ImplGetDisplayName (see page 5498)

GetDisplayName (see page 5493)

57.1.41.2.10 TBaseEndUserAdapter.GetEndUserAdapter Method

Gets the end-user adapter.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

The GetEndUserAdapter function is called from server-side script to get the end-user adapter. This function returns itself, that is, this TBaseEndUserAdapter (see page 5485).

See Also

ImplGetEndUserAdapter (see page 5499)

57.1.41.2.11 TBaseEndUserAdapter.GetLoggedIn Method

Determines if the user is logged in

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

The GetLoggedIn function is called from server-side script to determine if the user is logged in. This function calls ImplGetLoggedIn (see page 5499) to get the result.

See Also

ImplGetLoggedIn (see page 5499)

GetLoggedInValue (see page 5495)

57.1.41.2.12 TBaseEndUserAdapter.GetLoggedInValue Method

Determines if the user is logged in.

Pascal

```
function GetLoggedInValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

The GetLoggedInValue function is called from server-side script to determine if the user is logged in. This base class version of the function always returns false. A customized overridden version could be written specifically for your application.

See Also

ImplGetLoggedIn (see page 5499)

GetLoggedIn (see page 5495)

57.1.41.2.13 TBaseEndUserAdapter.GetLoginFormAction Method

Gets the action component for the login form.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

The GetLoginFormAction is called from server-side script to get the action component for the login form. This function calls ImplGetLoginFormAction (see page 5499) to get the action component.

See Also

ImplGetLoginFormAction (see page 5499)

57.1.41.2.14 TBaseEndUserAdapter.GetLogoutAction Method

Gets the action component for the logout form.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

The GetLogoutAction is called from server-side script to get the action component for the logout form. This function calls ImplGetLogoutFormAction to get the action component.

See Also

ImplGetLogoutAction (see page 5500)

57.1.41.2.15 TBaseEndUserAdapter.GetUserID Method

Gets the UserID from this TBaseEndUserAdapter (see page 5485).

Pascal

```
function GetUserID: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

The GetUserID function is called from server-side script to get the UserID from this TBaseEndUserAdapter (see page 5485). This base class version of the function always returns Unassigned. A customized overridden version of this function could be written specifically for your application.

57.1.41.2.16 TBaseEndUserAdapter.HasExecuteRights Method

Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(UnicodeString ARights);
```

Description

The HasExecuteRights function is called from server-side script to determine if the user has execute rights for the TBaseEndUserAdapter (see page 5485). This function calls ImplHasExecuteRights (see page 5500) to check the user rights.

See Also

ImplHasExecuteRights (see page 5500)

HasModifyRights (see page 5497)

HasViewRights (see page 5497)

57.1.41.2.17 TBaseEndUserAdapter.HasModifyRights Method

Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(UnicodeString ARights);
```

Description

The HasModifyRights function is called from server-side script to determine if the user has modify rights for the TBaseEndUserAdapter (see page 5485). This function calls ImplHasModifyRights (see page 5500) to check the user rights.

See Also

ImplHasModifyRights (see page 5500)

HasModifyRights

HasViewRights (see page 5497)

57.1.41.2.18 TBaseEndUserAdapter.HasViewRights Method

Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(UnicodeString ARights);
```

Description

The HasViewRights function is called from server-side script to determine if the user has view rights for the TBaseEndUserAdapter (see page 5485). This function calls ImplHasViewRights (see page 5501) to check the user rights.

See Also

ImplHasViewRights (see page 5501)

HasModifyRights (see page 5497)

HasViewRights

57.1.41.2.19 TBaseEndUserAdapter.ImplCanAddActionClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddActionClass function calls the base class to determine if the given action class can be added. In addition, it allows actions components to be added if they are derived from TCustomEndUserAdapterAction (see page 5673) at some

level. This function returns true if the action class is allowed to be added, or false otherwise.

See Also

ImplCanAddActionClass (see page 5540)

TCustomEndUserAdapterAction (see page 5673)

57.1.41.2.20 TBaseEndUserAdapter.ImplCanAddFieldClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddFieldClass function calls the base class to determine if the given action class can be added. In addition to that, it allows fields components to be added if they are derived from TAdapterEndUserField (see page 5237).

See Also

ImplCanAddFieldClass (see page 5540)

TAdapterEndUserField (see page 5237)

57.1.41.2.21 TBaseEndUserAdapter.ImplGetActionsList Method

Lists supported end-user adapter actions.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

The ImplGetActionsList procedure obtains the list of all supported actions for the end-user adapter.

57.1.41.2.22 TBaseEndUserAdapter.ImplGetDisplayName Method

Gets the display name for this TBaseEndUserAdapter (see page 5485).

Pascal

```
function ImplGetDisplayName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayName();
```

Description

The ImplGetDisplayName function is called from GetDisplayName (see page 5493) to get the display name for this TBaseEndUserAdapter (see page 5485). It gets the display name from the DisplayName field component.

See Also

GetDisplayName (see page 5493)

57.1.41.2.23 TBaseEndUserAdapter.ImplGetEndUserAdapter Method

Returns this TBaseEndUserAdapter (see page 5485).

Pascal

```
function ImplGetEndUserAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetEndUserAdapter();
```

Description

The ImplGetEndUserAdapter simply returns this TBaseEndUserAdapter (see page 5485).

See Also

GetEndUserAdapter (see page 5494)

57.1.41.2.24 TBaseEndUserAdapter.ImplGetFieldsList Method

Obtains the list of all of the fields for the end-user adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The ImplGetFieldsList procedure obtains the list of all of the fields for the end-user adapter.

57.1.41.2.25 TBaseEndUserAdapter.ImplGetLoggedIn Method

Checks the value of the LoggedIn field.

Pascal

```
function ImplGetLoggedIn: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoggedIn();
```

Description

The ImplGetLoggedIn function is called from GetLoggedIn (see page 5495). It checks the value of the LoggedIn field component and returns the result.

See Also

GetLoggedIn (see page 5495)

57.1.41.2.26 TBaseEndUserAdapter.ImplGetLoginFormAction Method

Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).

Pascal

```
function ImplGetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * ImplGetLoginFormAction();
```

Description

The ImplGetLoginFormAction function is called from GetLoginFormAction (see page 5495) to get the LoginForm action component for this TBaseEndUserAdapter (see page 5485).

See Also

GetLoginFormAction (see page 5495)

57.1.41.2.27 TBaseEndUserAdapter.ImplGetLogoutAction Method

Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).

Pascal

```
function ImplGetLogoutAction: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetLogoutAction();
```

Description

The ImplGetLogoutAction function is called from GetLogoutAction (see page 5496) to get the Logout action component for this TBaseEndUserAdapter (see page 5485).

See Also

GetLogoutAction (see page 5496)

57.1.41.2.28 TBaseEndUserAdapter.ImplHasExecuteRights Method

Checks user rights and determines if the user has execute rights.

Pascal

```
function ImplHasExecuteRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteRights(UnicodeString ARights);
```

Description

The ImplHasExecuteRights function is called from the HasExecuteRights (see page 5496) function. It calls the DoOnHasRights (see page 5492) function to check the user rights and determines if the user has execute rights.

See Also

OnHasRights (see page 5503)

57.1.41.2.29 TBaseEndUserAdapter.ImplHasModifyRights Method

Checks user rights and determines if the user has modify rights.

Pascal

```
function ImplHasModifyRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyRights(UnicodeString ARights);
```

Description

The ImplHasModifyRights function is called from the HasModifyRights (see page 5497) function. It calls the DoOnHasRights (see page 5492) function to check the user rights and determine if the user has modify rights.

See Also

OnHasRights (see page 5503)

57.1.41.2.30 TBaseEndUserAdapter.ImplHasViewRights Method

Checks user rights and determines if the user has view rights.

Pascal

```
function ImplHasViewRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewRights(UnicodeString ARights);
```

Description

The ImplHasViewRights is called from the HasViewRights (see page 5497) function. It calls the DoOnHasRights (see page 5492) function to check the user rights and determine if the user has view rights.

See Also

DoOnHasRights (see page 5492)

57.1.41.2.31 TBaseEndUserAdapter.ImplLogIn Method

Processes the login for the given user.

Pascal

```
procedure ImplLogIn(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ImplLogIn(Variant AUserID);
```

Description

The ImplLogIn procedure is called from LogIn (see page 5502). This function calls ExecuteLogin (see page 5492) to process the login for the given user.

See Also

Login (see page 5502)

ExecuteLogin (see page 5492)

57.1.41.2.32 TBaseEndUserAdapter.ImplProduceLoginPage Method

Generate the HTML login page.

Pascal

```
function ImplProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplProduceLoginPage();
```

Description

The ImplProduceLoginPage is called to generate the HTML login page It first calls the OnProduceLoginPage (see page 5504) event handler to allow for a customized version. If no event handler is defined, it then has default code to generate the page using the page producer.

See Also

OnProduceLoginPage (see page 5504)

57.1.41.2.33 TBaseEndUserAdapter.LogIn Method

Logs in a specified user.

Pascal

```
procedure LogIn(AUserID: Variant);
```

C++

```
__fastcall LogIn(Variant AUserID);
```

Description

Call LogIn to record a user as logged in. Typically, LogIn is called in response to a message from the login page that ProduceLoginPage generates.

UserID is the identifier of the user that logged in. It is the value of the UserID property of the entry for the user in a WebUserList component, or, in applications that implement their own login scheme, any other Variant that identifies an end user.

See Also

LoggedIn (see page 5968)

ProduceLoginPage (see page 5967)

TWebUserList (see page 6759)

UserID (see page 6754)

TLoginFormAdapter (see page 5810)

57.1.41.2.34 TBaseEndUserAdapter.ProduceLoginPage Method

Generates the login page.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

The ProduceLoginPage function is called from server-side script to generate the login page. This function calls ImplProduceLoginPage (see page 5501) to generate the page.

See Also

ImplProduceLoginPage (see page 5501)

57.1.41.2.35 TBaseEndUserAdapter.UserList Method

Gets the user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

The UserList function is called from server-side script to get the user list. If the user list is not stored in this TBaseEndUserAdapter (see page 5485), it then attempts to retrieve the user list through the application module.

57.1.41.3 TBaseEndUserAdapter Properties

The properties of the TBaseEndUserAdapter class are listed here.

Public Properties

	Name	Description
	LoginPage (see page 5503)	Specifies the name of the page that acts as the login page.

57.1.41.3.1 TBaseEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter (see page 5485). The name of the login page is the name of the Web page module that generates the login page.

57.1.41.4 TBaseEndUserAdapter Events

The events of the TBaseEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnHasRights (see page 5503)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5504)	Allows implementation of a customized login page.

57.1.41.4.1 TBaseEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights (see page 5492) to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5492)

ImplHasExecuteRights (see page 5500)

ImplHasModifyRights (see page 5500)

ImplHasViewRights (see page 5501)

57.1.41.4.2 TBaseEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage (see page 5501). If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5501)

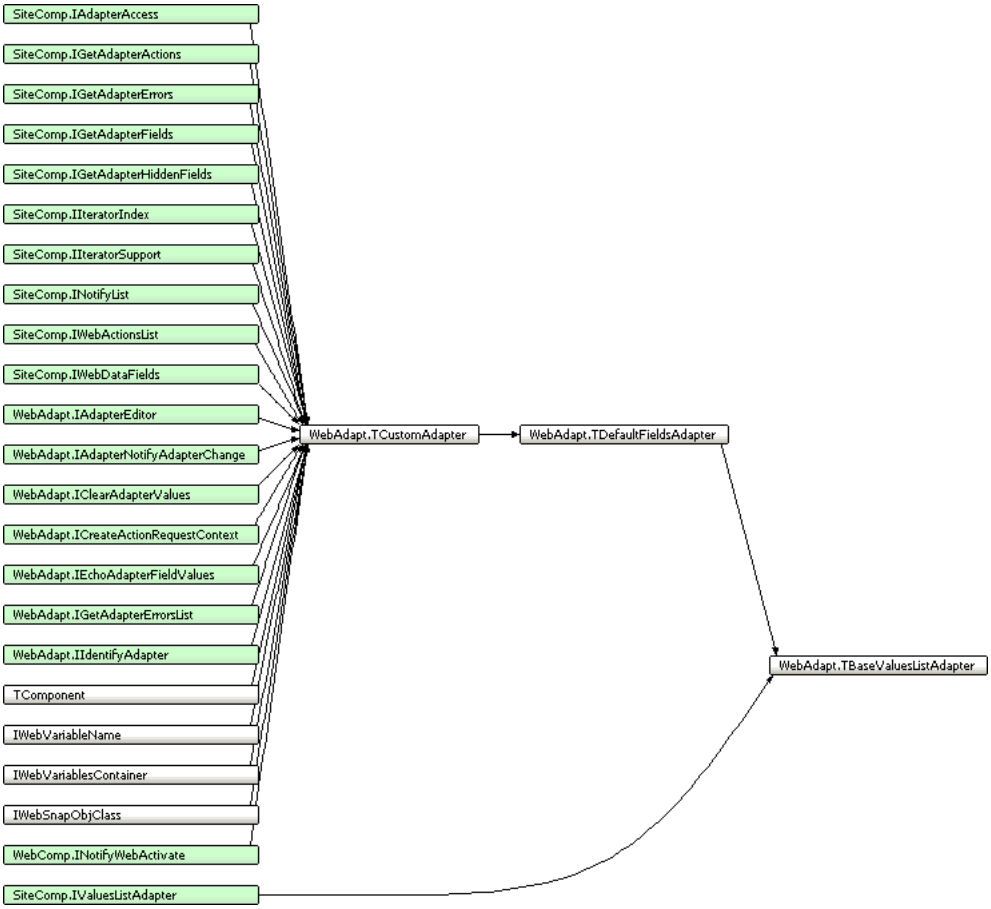
ProduceLoginPage (see page 5502)

CanProduceLoginPage (see page 5492)

57.1.42 TBaseValuesListAdapter Class

TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.

Class Hierarchy



Pascal

```
TBaseValuesListAdapter = class(TDefaultFieldsAdapter, IValuesListAdapter);
```

C++

```
class TBaseValuesListAdapter : public TDefaultFieldsAdapter, public IValuesListAdapter;
```

File

WebAdapt

Description

Descendants of TBaseValuesListAdapter (such as TStringsValuesList (see page 5843) and TDataSetValuesList) are used to provide name/value pairs for an adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList property. Typically, the list of name/value pairs is used by server-side scripts to populate <select>, <input type=checkbox>, or <input type=radio>.

TBaseValuesListAdapter has two default fields. One field is responsible for providing a value and the other for providing a name.

See Also

TStringsValuesList (see page 5843)

TDataSetValuesList

TAdapterUpdateField (see page 5407)

57.1.42.1 TBaseValuesListAdapter Members

The following tables list the members exposed by TBaseValuesListAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5510)	Returns an image associated with the current name/value pair.

	GetListImageOfValue (see page 5510)	Looks up an image for a value.
	GetListName (see page 5511)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5511)	Looks up a name for a value.
	GetListValue (see page 5511)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5512)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5512)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5513)	Adds default field components to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetListImage (see page 5513)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 5513)	Looks up an image for a value.
V	ImplGetListName (see page 5514)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 5514)	Looks up a name for a value.
V	ImplGetListValue (see page 5514)	Responsible for returning the value of the name/value pair.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
V	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
V	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
V	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
V	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
V	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.

	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).

	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.42.2 TBaseValuesListAdapter Methods

The methods of the TBaseValuesListAdapter class are listed here.

Protected Methods

	Name	Description
	GetListImage (see page 5510)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5510)	Looks up an image for a value.
	GetListName (see page 5511)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5511)	Looks up a name for a value.
	GetListValue (see page 5511)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	GetWebSnapObjClass (see page 5512)	Returns the class type for this adapter field.
V	ImplCanAddFieldClass (see page 5512)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplGetFieldsList (see page 5513)	Adds default field components to this adapter.
V	ImplGetListImage (see page 5513)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 5513)	Looks up an image for a value.
V	ImplGetListName (see page 5514)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 5514)	Looks up a name for a value.
V	ImplGetListValue (see page 5514)	Responsible for returning the value of the name/value pair.

57.1.42.2.1 TBaseValuesListAdapter.GetListImage Method

Returns an image associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the result of ImplGetListImage (see page 5513).

See Also

ImplGetListImage (see page 5513)

57.1.42.2.2 TBaseValuesListAdapter.GetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue returns the result of ImplGetListImageOfValue (see page 5513).

See Also

ImplGetListImageOfValue (see page 5513)

57.1.42.2.3 TBaseValuesListAdapter.GetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall UnicodeString GetListName();
```

Description

GetListName returns the result of ImplGetListName (see page 5514).

See Also

GetListName

ImplGetListValue (see page 5514)

57.1.42.2.4 TBaseValuesListAdapter.GetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall UnicodeString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue returns the result of ImplGetListNameOfValue (see page 5514).

See Also

ImplGetListNameOfValue (see page 5514)

57.1.42.2.5 TBaseValuesListAdapter.GetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the result of ImplGetListValue (see page 5514).

See Also

GetListName (see page 5511)

ImplGetListValue (see page 5514)

57.1.42.2.6 TBaseValuesListAdapter.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetValueFieldClass (see page 5512)

57.1.42.2.7 TBaseValuesListAdapter.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetNameFieldClass (see page 5512)

57.1.42.2.8 TBaseValuesListAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

57.1.42.2.9 TBaseValuesListAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass check the field class by calling GetNameFieldClass (see page 5512), GetValueFieldClass (see page 5512), and TCustomAdapter (see page 5519)'s ImplCanAddFieldClass method.

See Also

GetNameFieldClass (see page 5512)

GetValueFieldClass (see page 5512)

ImplCanAddFieldClass (see page 5540)

57.1.42.2.10 TBaseValuesListAdapter.ImplGetFieldsList Method

Adds default field components to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList adds a value field and a name field using the classes returned by GetNameFieldClass (see page 5512) and GetValueFieldClass (see page 5512).

See Also

GetNameFieldClass (see page 5512)

GetValueFieldClass (see page 5512)

57.1.42.2.11 TBaseValuesListAdapter.ImplGetListImage Method

Responsible returning an image associated with the current name/value pair.

Pascal

```
function ImplGetListImage: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImage();
```

Description

ImplGetListImage is a placeholder. Descendant adapters that support images must override this method.

See Also

GetListImage (see page 5510)

57.1.42.2.12 TBaseValuesListAdapter.ImplGetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function ImplGetListImageOfValue(Value: Variant): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImageOfValue(Variant Value);
```

Description

ImplGetListImageOfValue is a placeholder. Descendant classes that support image lookups must override this function.

See Also

GetListImageOfValue (see page 5510)

57.1.42.2.13 TBaseValuesListAdapter.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetListName();
```

Description

ImplGetListName returns an empty string. ImplGetListName must be overridden by a descendant class.

See Also

GetListName (see page 5511)

57.1.42.2.14 TBaseValuesListAdapter.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue is a placeholder. Descendant classes that support name lookups must override this function.

See Also

GetListNameOfValue (see page 5511)

57.1.42.2.15 TBaseValuesListAdapter.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue is a placeholder. ImplGetListValue must be overriden by a descendant class.

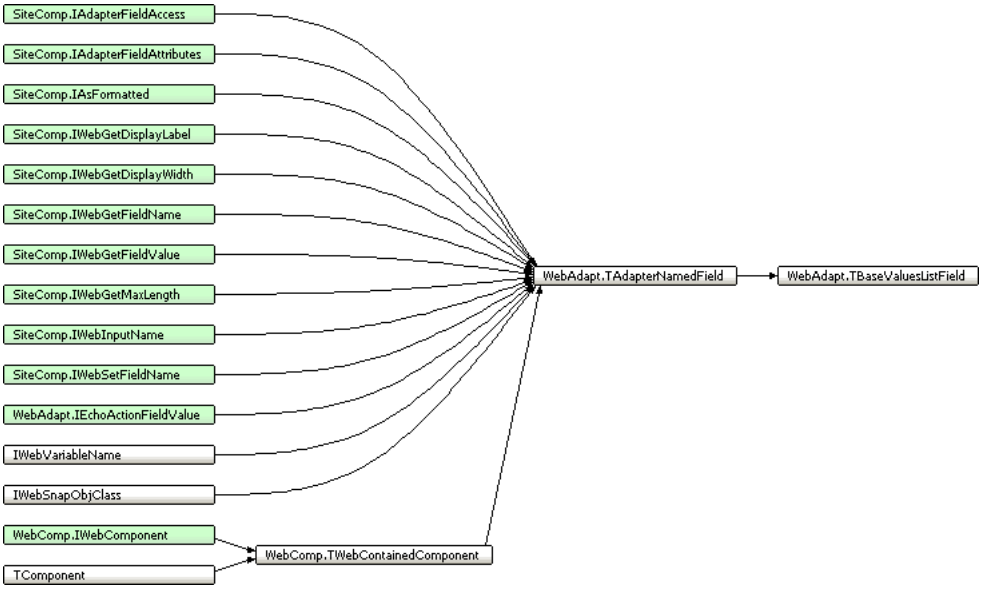
See Also

GetListValue (see page 5511)

57.1.43 TBaseValuesListField Class

TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.

Class Hierarchy



Pascal

```
TBaseValuesListField = class(TAdapterNamedField);
```

C++

```
class TBaseValuesListField : public TAdapterNamedField;
```

File

WebAdapt

Description

Derive from TBaseValuesListField to declare an adapter field that can be a child of TBaseValuesListAdapter (see page 5504).

See Also

TStringsValuesList (see page 5843)

TBaseValuesListAdapter (see page 5504)

TAdapterNamedField (see page 5339)

TValuesListNameField (see page 5852)

TValuesListValueField (see page 5855)

57.1.43.1 TBaseValuesListField Members

The following tables list the members exposed by TBaseValuesListField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5518)	Retrieves the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.

	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5518)	Retrieves the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.

	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.43.2 TBaseValuesListField Methods

The methods of the TBaseValuesListField class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 5518)	Retrieves the parent adapter component.

57.1.43.2.1 TBaseValuesListField.GetAdapter Method

Retrieves the parent adapter component.

Pascal

```
function GetAdapter: TBaseValuesListAdapter;
```

C++

```
__fastcall TBaseValuesListAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 5518) property.

See Also

Adapter (see page 5518)

TBaseValuesListAdapter (see page 5504)

57.1.43.3 TBaseValuesListField Properties

The properties of the TBaseValuesListField class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5518)	Retrieves the parent adapter component.

57.1.43.3.1 TBaseValuesListField.Adapter Property

Retrieves the parent adapter component.

Pascal

```
property Adapter: TBaseValuesListAdapter;
```

C++

```
__property TBaseValuesListAdapter Adapter;
```

Description

The Adapter property is used by descendants to access the parent adapter component.

See Also

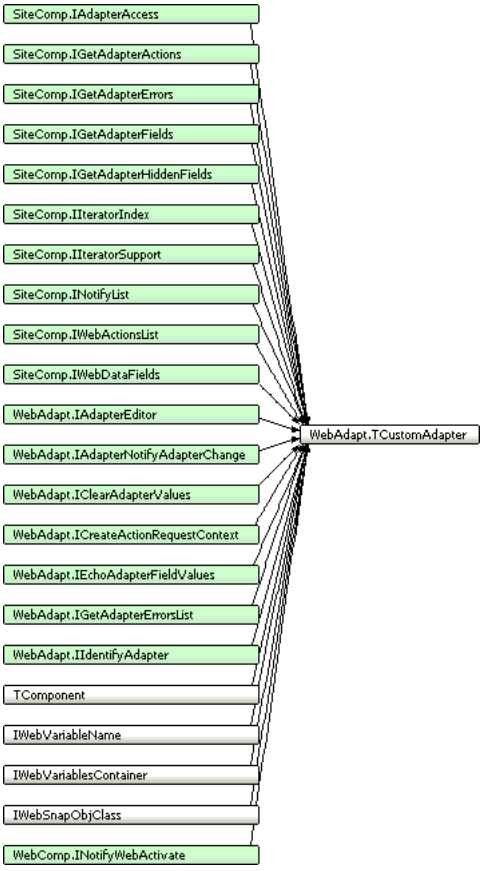
Adapter

TBaseValuesListAdapter (see page 5504)

57.1.44 TCustomAdapter Class

TCustomAdapter is a base class for adapter components.

Class Hierarchy



Pascal

```
TCustomAdapter = class(TComponent, IIdentifyAdapter, IWebVariableName, IWebVariablesContainer,
INotifyList, IWebSnapObjClass, IGetAdapterErrors, IGetAdapterErrorsList, INotifyWebActivate,
IAdapterEditor, IGetAdapterFields, IGetAdapterActions, IIteratorSupport, IClearAdapterValues,
IEchoAdapterFieldValues, IAdapterAccess, IGetAdapterHiddenFields, ICreateActionRequestContext,
IWebDataFields, IWebActionsList, IAdapterNotifyAdapterChange, IIteratorIndex);
```

C++

```
class TCustomAdapter : public TComponent, public IIdentifyAdapter, public IWebVariableName,
public IWebVariablesContainer, public INotifyList, public IWebSnapObjClass, public
IGetAdapterErrors, public IGetAdapterErrorsList, public INotifyWebActivate, public
IAdapterEditor, public IGetAdapterFields, public IGetAdapterActions, public IIteratorSupport,
public IClearAdapterValues, public IEchoAdapterFieldValues, public IAdapterAccess, public
IGetAdapterHiddenFields, public ICreateActionRequestContext, public IWebDataFields, public
```

```
IWebActionsList, public IAdapterNotifyAdapterChange, public IIteratorIndex;
```

File

WebAdapt

Description

TCustomAdapter implements interfaces used to access an adapter properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter class.

See Also

TAdapter (see page 5179)

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

TFieldValuesAdapter (see page 5789)

TDataSetValuesList

TStringsValuesList (see page 5843)

TApplicationAdapter (see page 5427)

TLoginFormAdapter (see page 5810)

TDataSetAdapter

TPagedAdapter (see page 5830)

57.1.44.1 TCustomAdapter Members

The following tables list the members exposed by TCustomAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.

	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.

	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).

	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	Actions (see page 5558)	Actions is the container for adapter actions components.
	Data (see page 5558)	Data is the container for adapter field components.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.

57.1.44.2 TCustomAdapter.Create Constructor

Constructor for TCustomAdapter (see page 5519).

Pascal

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

C++

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

Description

Creates instance variables that hold interfaces, object references used by TCustomAdapter (see page 5519).

See Also

Create_d

57.1.44.3 TCustomAdapter.Destroy Destructor

Destructor for TCustomAdapter (see page 5519).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapter();
```

Description

Frees objects used by TCustomAdapter (see page 5519).

See Also

System::TObject::Free (see page 3382)

57.1.44.4 TCustomAdapter Methods

The methods of the TCustomAdapter class are listed here.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.

	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Public Methods

	Name	Description
	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

57.1.44.4.1 TCustomAdapter.AddAdapterModeHiddenField Method

Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddAdapterModeHiddenField(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddAdapterModeHiddenField(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may support modes. For example, TDataSetAdapter supports Edit, Insert (see page 3698), and Browse modes. The hidden field defined by AddAdapterModeHiddenField is added to an HTML form using the server-side script method HiddenFields.WriteFields.

See Also

listhiddenfields (see page 5550)

57.1.44.4.2 TCustomAdapter.AddOrigValueHiddenFields Method

Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddOrigValueHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddOrigValueHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

In order to ensure that the data associated with the adapter hasn't been changed by another user, descendant adapters may check that the current values of adapter fields are the same as the original values. For example, TDataSetAdapter raises an exception if the original values are not the same as the dataset's values. The hidden fields defined by AddOrigValueHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 5550)

57.1.44.4.3 TCustomAdapter.AddRecordKeysHiddenFields Method

Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddRecordKeysHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddRecordKeysHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may use key values to define the relationship between values in the HTTP request and adapter data

values. For example, TDataSetAdapter supports updates to multiple detail records by using hidden fields to identify the relationship between values in the HTTP request with dataset rows. The hidden fields defined by AddRecordKeysHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 5550)

57.1.44.4.4 TCustomAdapter.CanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

CanAddActionClass is called at design-time when editing an adapter. CanAddActionClass returns the result of ImplCanAddActionClass (see page 5540).

See Also

ImplCanAddActionClass (see page 5540)

57.1.44.4.5 TCustomAdapter.CanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

CanAddFieldClass is called at design-time when editing an adapter. CanAddFieldClass returns the result of ImplCanAddFieldClass (see page 5540).

See Also

ImplCanAddFieldClass (see page 5540)

57.1.44.4.6 TCustomAdapter.CheckModifyAccess Method

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This is CheckModifyAccess, a member of class TCustomAdapter.

57.1.44.4.7 TCustomAdapter.ClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

ClearAdapterValues calls ImplClearAdapterValues (see page 5540).

See Also

ImplNotifyDeactivate (see page 5547)

ImplClearAdapterValues (see page 5540)

57.1.44.4.8 TCustomAdapter.ClearErrors Method

Clears the content of the adapter's Errors (see page 5559) property.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

The errors are cleared when the web module is deactivated. This ensures that the errors encountered during the processing of an HTTP request will not affect another HTTP request.

See Also

Errors (see page 5559)

ClearErrors (see page 5470)

ImplNotifyDeactivate (see page 5547)

57.1.44.4.9 TCustomAdapter.CreateActionRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

Child adapter actions call this method.

See Also

IAdapterDispatchParams (see page 929)

CreateRequestContext (see page 5440)

WebContext (see page 5977)

57.1.44.4.10 TCustomAdapter.CreateErrorsList Method

Creates the object used to manage errors associated with this adapter.

Pascal

```
function CreateErrorsList: TBaseAdapterErrorsList; virtual;
```

C++

```
virtual __fastcall TBaseAdapterErrorsList CreateErrorsList();
```

Description

CreateErrorsList creates TAdapterErrorsList (see page 5248). Descendants may override this function to create a custom errors list.

See Also

Errors (see page 5559)

TAdapterErrorsList (see page 5248)

57.1.44.4.11 TCustomAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

EncodeActionParams is called by a child adapter actions when generating the name/value pairs that must be sent in the HTTP request when executing an action. For example, TDefaultFieldsPagedAdapter (see page 5733) writes the current page number as a name/value pair. ExtractRequestParams (see page 5530) and EncodeActionParams work together. ExtractRequestParams (see page 5530) reads name/value pairs that are written by EncodeActionParams.

See Also

EncodeActionParams (see page 5740)

ExtractRequestParams (see page 5530)

57.1.44.4.12 TCustomAdapter.EndIterator Method

Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. EndIterator calls ImplEndIterator (see page 5541).

See Also

enditerator (see page 997)

ImplEndIterator (see page 5541)

57.1.44.4.13 TCustomAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); virtual;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. For example, TDefaultFieldsPagedAdapter (see page 5733) extracts the current page number from the request. ExtractRequestParams and EncodeActionParams (see page 5529) work together. ExtractRequestParams reads name/value pairs that are written by EncodeActionParams (see page 5529).

See Also

ExtractRequestParams (see page 5740)

EncodeActionParams (see page 5529)

57.1.44.4.14 TCustomAdapter.FindAction Method

Returns an action component with a matching action name.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

The action name is used when a component property references an adapter action by name. Actions (see page 5558) also have a variable name that is used to reference the action with server-side script. The variable name and action name are usually the same. The only time that they need to be different is when the action name is not a valid server-side script identifier.

See Also

FindField (see page 5530)

57.1.44.4.15 TCustomAdapter.FindField Method

Returns an adapter field component with a matching field name.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

The field name is used when a component property references an adapter field by name. Adapter fields also have a variable name that is used to reference the field with server-side script. The variable name and field name are usually the same. The only time that they need to be different is when the field name is not a valid server-side script identifier.

See Also

FindField

57.1.44.4.16 TCustomAdapter.FindNamedAction Method

Finds an action with a matching variable name.

Pascal

```
function FindNamedAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedAction(const UnicodeString AName);
```

Description

The AName parameter contains the name used when referencing an action with server-side script. This name may not be the same as the component name. For example, server-side script accesses actions in a TDataSetAdapter with using the command name (e.g.; DeleteRow) rather than the adapter action component name.

See Also

FindNamedVariable (see page 5531)

57.1.44.4.17 TCustomAdapter.FindNamedField Method

Finds an adapter field with a matching variable name.

Pascal

```
function FindNamedField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedField(const UnicodeString AName);
```

Description

The AName parameter contains the name used when referencing a field with server-side script. This name may not be the same as the component name. For example, server-side script accesses fields in a TDataSetAdapter with using the dataset field name rather than the adapter field component name.

See Also

FindNamedVariable (see page 5531)

57.1.44.4.18 TCustomAdapter.FindNamedVariable Method

Finds a field or action with a matching variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

FindNamedVariable calls FindNamedField (see page 5531) and FindNamedAction (see page 5531). The AName parameter contains the name used when referencing an action or field with server-side script. This name may not be the same as the component name. For example, server-side script references fields in a TDataSetAdapter using the dataset field name rather than the adapter field component name.

See Also

FindNamedAction (see page 5531)

FindNamedField (see page 5531)

57.1.44.4.19 TCustomAdapter.FindVariable Method

Finds an adapter action or adapter field with the given variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

FindVariable is called when server side script references a property of an adapter object. FindVariable calls ImplFindVariable (see page 5541).

See Also

ImplFindVariable (see page 5541)

57.1.44.4.20 TCustomAdapter.FindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer is called when server side script references a property of a property. FindVariableContainer returns the result of ImplFindVariableContainer (see page 5541).

See Also

FindVariable (see page 5532)

ImplFindVariableContainer (see page 5541)

57.1.44.4.21 TCustomAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; virtual;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

Descendants override this method to specify a different adapter action container.

See Also

Actions (see page 5558)

57.1.44.4.22 TCustomAdapter.GetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

GetActionsList is called at design time to get a list containing names and objects. Each name identifies an action name to insert and each object identifies a class of adapter action component. When editing the adapter, these action names are displayed in the dialog box that is used to add new actions. For example, when editing a TDataSetAdapter a list of all of the dataset commands are displayed such as NextRow, PrevRow, DeleteRow, etc.

See Also

ImplGetActionsList (see page 5542)

57.1.44.4.23 TCustomAdapter.GetAdapterActions Method

Returns the adapter actions container object.

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

Returns the adapter actions container object.

See Also

Actions (see page 5558)

57.1.44.4.24 TCustomAdapter.GetAdapterErrors Method

Pascal

```
function GetAdapterErrors: TObject;
```

C++

```
__fastcall TObject * GetAdapterErrors();
```

Description

This is GetAdapterErrors, a member of class TCustomAdapter.

57.1.44.4.25 TCustomAdapter.GetAdapterErrorsList Method

<insert text here>

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

Returns an object used to track errors.

See Also

TBaseAdapterErrorsList (see page 5467)

Errors (see page 5559)

57.1.44.4.26 TCustomAdapter.GetAdapterFields Method

Returns the adapter fields container object.

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

Returns the adapter fields container object.

See Also

Data (see page 5558)

57.1.44.4.27 TCustomAdapter.GetAdapterHiddenFields Method

Returns a list of hidden fields that hold adapter state information.

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

GetAdapterHiddenFields populates a list by calling ListHiddenFields (see page 5550).

See Also

listhiddenfields (see page 5550)

57.1.44.4.28 TCustomAdapter.GetAdapterHiddenRecordFields Method

Returns a list of hidden fields that associate HTML form fields and adapter field.

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

GetAdapterHiddenRecordsFields populates a list by calling ListHiddenRecordFields (see page 5550). If the list is empty, nil (Delphi) or NULL (C++) is returned. Otherwise, the list is returned.

See Also

ListHiddenRecordFields (see page 5550)

57.1.44.4.29 TCustomAdapter.GetChildren Method

Provides the interface for a method that adapters use to return their child components.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TCustomAdapter (see page 5519) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It calls the Proc procedure passing the Actions (see page 5558) and Data (see page 5558) properties. This allows the adapter actions and adapter fields to be loaded and saved with the TCustomAdapter (see page 5519) object.

See Also

Actions (see page 5558)

Data (see page 5558)

57.1.44.4.30 TCustomAdapter.GetDataClass Method

Specifies the class to create as a container for adapter fields components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; virtual;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

Descendants override this method to specify a different adapter fields container.

See Also

TAdapterField (see page 5256)

Data (see page 5558)

57.1.44.4.31 TCustomAdapter.GetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

EchoActionFieldValues (see page 5559) calls ImplEchoActionFieldValues.

See Also

EchoActionFieldValues (see page 5559)

ImplGetEchoActionFieldValues (see page 5542)

57.1.44.4.32 TCustomAdapter.GetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

GetFieldsList is called at design time to get a list containing names and objects. Each name identifies a field name to insert and each object identifies a class of adapter field. When editing the adapter, these field names are displayed in the dialog box that is used to add new adapter fields. For example, when editing a TDataSetAdapter a list of all of the dataset's fields are displayed. GetFieldsList calls ImplGetFieldsList (see page 5542).

See Also

ImplGetFieldsList (see page 5542)

57.1.44.4.33 TCustomAdapter.GetHiddenFieldOptions Method

Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

Pascal

```
function GetHiddenFieldOptions: TAdapterHiddenFieldOptions; virtual;
```

C++

```
virtual __fastcall TAdapterHiddenFieldOptions GetHiddenFieldOptions();
```

Description

An adapter may use hidden fields to pass state information down to the client with the HTTP response so that the state information is available when processing a resulting HTTP response. Descendant classes may override this method.

See Also

listhiddenfields (see page 5550)

ListHiddenRecordFields (see page 5550)

57.1.44.4.34 TCustomAdapter.GetIteratorIndex Method

Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

Adapters may support multiple data records. For example, TDataSetAdapter provides access to the records in a dataset. GetIteratorIndex calls ImplGetIteratorIndex (see page 5543).

See Also

GetIteratorIndex (see page 995)

ImplGetIteratorIndex (see page 5543)

57.1.44.4.35 TCustomAdapter.GetVariable Method

Returns an action or field.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

GetVariableCount (see page 5538) and GetVariable is used to iterate over actions and fields in an adapter. The I parameter is a zero-based index that must be less than the result of GetVariableCount (see page 5538). GetVariable calls ImplGetVariable (see page 5543).

See Also

GetVariable

ImplGetVariable (see page 5543)

GetVariableCount (see page 5538)

57.1.44.4.36 TCustomAdapter.GetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount and GetVariable (see page 5537) is used to iterate over the actions and fields in an adapter. GetVariableCount calls ImplGetVariableCount (see page 5544).

See Also

GetVariable (see page 5537)

ImplGetVariableCount (see page 5544)

57.1.44.4.37 TCustomAdapter.GetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter with server-side script. GetVariableName calls ImplGetVariableName (see page 5544).

See Also

ImplGetVariableName (see page 5544)

57.1.44.4.38 TCustomAdapter.GetVisibleActions Method

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

This is GetVisibleActions, a member of class TCustomAdapter.

57.1.44.4.39 TCustomAdapter.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class TCustomAdapter.

57.1.44.4.40 TCustomAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

57.1.44.4.41 TCustomAdapter.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class TCustomAdapter.

57.1.44.4.42 TCustomAdapter.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class TCustomAdapter.

57.1.44.4.43 TCustomAdapter.HasViewAccess Method

Checks the end-user's right to view this adapter's fields.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 5546).

See Also

ViewAccess (see page 5560)

57.1.44.4.44 TCustomAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass returns false. This means that only TAdapterAction (see page 5190) can be added. Descendants override this method to allow additional field types.

See Also

CanAddActionClass (see page 5527)

ImplCanAddClass (see page 5201)

57.1.44.4.45 TCustomAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass returns false. This means that only TAdapterField (see page 5256) can be added. Descendants override this method to allow additional field types.

See Also

CanAddFieldClass (see page 5527)

ImplCanAddClass (see page 5268)

57.1.44.4.46 TCustomAdapter.ImplClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ImplClearAdapterValues; virtual;
```

C++

```
virtual __fastcall ImplClearAdapterValues();
```

Description

ClearAdapterValues (see page 5528) clears the echo flag.

See Also

ImplSetEchoActionFieldValues (see page 5548)

ClearAdapterValues (see page 5528)

57.1.44.4.47 TCustomAdapter.ImplEndIterator Method

Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); virtual;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplEndIterator sets the ItemIndex to 0 and the iterator flag to false.

See Also

enditerator (see page 997)

EndIterator (see page 5529)

57.1.44.4.48 TCustomAdapter.ImplFindVariable Method

Finds an adapter action or adapter field child with a matching variable name.

Pascal

```
function ImplFindVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariable(const UnicodeString AName);
```

Description

ImpFindVariable returns the result of FindNamedVariable (see page 5531).

See Also

FindVariable (see page 5532)

57.1.44.4.49 TCustomAdapter.ImplFindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function ImplFindVariableContainer(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer (see page 5532) is called when server side script references a property of a property. ImplFindVariableContainer tries to find an adapter field or adapter action that is a container for other objects.

See Also

FindVariableContainer (see page 5532)

57.1.44.4.50 TCustomAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList is called by GetActionsList (see page 5533). ImplGetActionsList provides an empty list. ImplGetActionsList may be overridden by descendant classes.

See Also

GetActionsList (see page 5533)

57.1.44.4.51 TCustomAdapter.ImplGetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValues: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValues();
```

Description

ImplGetEchoActionFieldValues returns the value passed to ImplSetEchoActionFieldValues (see page 5548). If ImplSetEchoActionFieldValues (see page 5548) has not been processed during the current HTTP request then False is returned.

See Also

EchoActionFieldValues (see page 5559)

GetEchoActionFieldValues (see page 5536)

57.1.44.4.52 TCustomAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList is called by GetFieldsList (see page 5536). ImplGetFieldsList provides an empty list. ImplGetFieldsList may be overridden by descendant classes.

See Also

GetFieldsList (see page 5536)

57.1.44.4.53 TCustomAdapter.ImplGetIteratorIndex Method

Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex returns the value ItemIndex. ItemIndex is modified by ImplStartIterator (see page 5548), ImplNextIteration (see page 5546), and ImplEndIterator (see page 5541).

See Also

GetIteratorIndex (see page 995)

GetIteratorIndex (see page 5537)

ImplStartIterator (see page 5548)

ImplNextIteration (see page 5546)

ImplEndIterator (see page 5541)

57.1.44.4.54 TCustomAdapter.ImplGetVariable Method

Returns an action or field.

Pascal

```
function ImplGetVariable(I: Integer): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetVariable(int I);
```

Description

ImplGetVariable is called by GetVariable (see page 5537). ImplGetVariable uses the I parameter to locate an adapter action or adapter field.

See Also

GetVariable (see page 5537)

57.1.44.4.55 TCustomAdapter.ImplGetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function ImplGetVariableCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetVariableCount();
```

Description

ImplGetVariableCount is called by GetVariableCount (see page 5538).

See Also

GetVariableCount (see page 5538)

57.1.44.4.56 TCustomAdapter.ImplGetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

ImplGetVariableName returns the component name.

See Also

GetVariableName (see page 5538)

57.1.44.4.57 TCustomAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions is called by GetVisibleActions (see page 5538). ImplGetVisibleActions may be overriden by descendant classes.

See Also

getvisibleactions (see page 5538)

57.1.44.4.58 TCustomAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields is called by GetVisibleFields (see page 5538). ImplGetVisibleFields may be overridden by descendant classes.

See Also

getvisiblefields (see page 5538)

57.1.44.4.59 TCustomAdapter.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter's actions.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 5539) to check the end-user's right to execute the adapter's actions. If the ExecuteAccess (see page 5560) property is not blank, this function calls the EndUserAdapter's HasExecuteRights function, passing the ExecuteAccess (see page 5560) property value.

See Also

hasexecuteaccess (see page 5539)

executeaccess (see page 5560)

HasExecuteAccess (see page 5451)

57.1.44.4.60 TCustomAdapter.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter's fields.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 5539) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 5560) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 5560) property value.

See Also

hasmodifyaccess (see page 5539)

modifyaccess (see page 5560)

hasmodifyaccess (see page 5539)

57.1.44.4.61 TCustomAdapter.ImplHasViewAccess Method

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This is ImplHasViewAccess, a member of class TCustomAdapter.

57.1.44.4.62 TCustomAdapter.ImplInIterator Method

Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplGetInIterator returns the value of the iteration flag. The iteration flag is set to true by TCustomAdapter (see page 5519)'s ImplStartIterator (see page 5548) method.

See Also

InIterator (see page 995)

InIterator (see page 5549)

ImplStartIterator (see page 5548)

57.1.44.4.63 TCustomAdapter.ImplNextIteration Method

Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplNextIteration increments ItemIndex and returns true if the iteration index is less than ItemCount.

See Also

NextIteration (see page 997)

NextIteration (see page 5551)

ImplStartIterator (see page 5548)

ImplEndIterator (see page 5541)

57.1.44.4.64 TCustomAdapter.ImplNotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure ImplNotifyActivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyActivate();
```

Description

A web module is activated when an HTTP request is received. ImplNotifyActivate is a placeholder. Descendant classes override this method to initialize the adapter before an HTTP request processed.

See Also

NotifyActivate (see page 5551)

57.1.44.4.65 TCustomAdapter.ImplNotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure ImplNotifyDeactivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. ImplNotifyDeactivate discards adapter state information including adapter errors, iteration flag, and iteration index.

See Also

NotifyDeactivate (see page 5552)

57.1.44.4.66 TCustomAdapter.ImplSetActions Method

Initializes the Actions (see page 5558) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5558) property is the container for adapter action components. Descendants may override this method.

See Also

Actions (see page 5558)

SetActions (see page 5552)

57.1.44.4.67 TCustomAdapter.ImplSetData Method

Initializes the Data (see page 5558) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5558) property is the container for adapter field components. Descendants may override this method.

See Also

Data (see page 5558)

SetData (see page 5553)

57.1.44.4.68 TCustomAdapter.ImplSetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure ImplSetEchoActionFieldValues(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValues(Boolean AValue);
```

Description

ImplEchoActionFieldValues modified the EchoActionFieldValue property of the adapter's fields.

See Also

GetEchoActionFieldValues (see page 5536)

SetEchoActionFieldValues (see page 5553)

EchoActionFieldValue (see page 5356)

57.1.44.4.69 TCustomAdapter.ImplStartIterator Method

Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplStartIterator sets ItemIndex to 0 and sets the iteration flag to true. true is returned.

See Also

startiterator (see page 998)

StartIterator (see page 5553)

ImplNextIteration (see page 5546)

ImplEndIterator (see page 5541)

57.1.44.4.70 TCustomAdapter.InIterator Method

Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. InIterator calls ImplInIterator (see page 5546).

See Also

InIterator (see page 995)

ImplInIterator (see page 5546)

57.1.44.4.71 TCustomAdapter.IsActionInUse Method

Indicates whether a particular action has been added to the adapter.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

IsActionInUse is called at design time to determine whether an action with a particular name has been added to the adapter. When editing the adapter, the names of actions in use are not displayed in the dialog box used to add new actions.

See Also

IsFieldInUse (see page 5549)

GetFieldsList (see page 5536)

57.1.44.4.72 TCustomAdapter.IsFieldInUse Method

Indicates whether a particular field has been added to the adapter.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

IsFieldInUse is called at design time to determine whether a field with a particular name has been added to the adapter. When editing the adapter, the names of these fields if use are not displayed in the dialog box used to add new fields.

See Also

IsActionInUse (see page 5549)

GetFieldsList (see page 5536)

57.1.44.4.73 TCustomAdapter.ListHiddenFields Method

Pascal

```
procedure ListHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

This is ListHiddenFields, a member of class TCustomAdapter.

57.1.44.4.74 TCustomAdapter.ListHiddenRecordFields Method

Specifies the HTML form hidden fields needed by the adapter.

Pascal

```
procedure ListHiddenRecordFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenRecordFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Some adapters may use hidden fields to define the relationship between input fields in the HTML form and data of the adapter. For example, an HTML form that contains an HTML table for updating 5 detail records of a TDataSetAdapter will have 5 sets of hidden fields. Each set provides state information needed by the adapter to determine the relationship between the input fields in the HTTP request and the dataset.

ListHiddenRecordFields may call AddOrigValueHiddenFields (see page 5526) and/or AddRecordKeysHiddenFields (see page 5526) depending on the flags returned by GetHiddenFieldOptions (see page 5536).

See Also

GetHiddenFieldOptions (see page 5536)

TAdapterHiddenFieldsList (see page 5290)

ListHiddenRecordFields

AddOrigValueHiddenFields (see page 5526)

AddRecordKeysHiddenFields (see page 5526)

GetAdapterHiddenRecordFields (see page 5535)

57.1.44.4.75 TCustomAdapter.NextIteration Method

Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. NextIteration calls ImplNextIteration (see page 5546).

See Also

NextIteration (see page 997)

ImplNextIteration (see page 5546)

57.1.44.4.76 TCustomAdapter.Notification Method

Clears references to deleted components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted.

57.1.44.4.77 TCustomAdapter.NotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

A web module is activated when an HTTP request is received. NotifyActivate calls ImplNotifyActivate (see page 5547).

See Also

ImplNotifyActivate (see page 5547)

NotifyDeactivate (see page 5552)

57.1.44.4.78 TCustomAdapter.NotifyAdapterChange Method

Notifies components that the adapter has changed in some way.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

NotifyAdapterChange calls components that have registered a notification callback to let them know that the adapter has changed. For example, TAdapterFieldGroup needs to know when an adapter field is added to the adapter component that the TAdapterFieldGroup references.

See Also

NotifyList (see page 5560)

57.1.44.4.79 TCustomAdapter.NotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. NotifyDeactivate calls ImplNotifyDeactivate (see page 5547).

See Also

ImplNotifyDeactivate (see page 5547)

NotifyActivate (see page 5551)

57.1.44.4.80 TCustomAdapter.SetActions Method

Initializes the Actions (see page 5558) property.

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5558) property is the container for adapter action components. SetActions calls ImplSetActions (see page 5547).

See Also

Actions (see page 5558)

ImplSetActions (see page 5547)

57.1.44.4.81 TCustomAdapter.SetData Method

Initializes the Data (see page 5558) property.

Pascal

```
procedure SetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall SetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5558) property is the container for adapter field components. SetData calls ImplSetData (see page 5548).

See Also

Data (see page 5558)

ImplSetData (see page 5548)

57.1.44.4.82 TCustomAdapter.SetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

SetEchoActionFieldValues calls ImplSetActionFieldValues.

See Also

EchoActionFieldValues (see page 5559)

ImplSetEchoActionFieldValues (see page 5548)

57.1.44.4.83 TCustomAdapter.StartIterator Method

Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. StartIterator calls ImplStartIterator (see page 5548).

See Also

startiterator (see page 998)

ImplStartIterator (see page 5548)

57.1.44.4.84 TCustomAdapter.UpdateFieldsExecuteValues Method

Executes the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsExecuteValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsExecuteValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsExecuteValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ExecuteValue function. Adapter fields raise an exception to indicate an error. If an exception occurs, the exception message is added to the Errors (see page 5559) object and execution continues.

When a field is executable, the field value passed in the HTTP request is used to execute a command rather than update a value. For example, an HTML form may contain a table with a delete row checkbox on each row. When the form is submitted, the values in the HTTP request indicate whether to delete one or more rows. Executable adapter fields must implement the IExecuteValue (see page 5869) interface.

See Also

Errors (see page 5559)

IActionRequest

IExecuteValue (see page 5869)

57.1.44.4.85 TCustomAdapter.UpdateFieldsGetAnyChanges Method

Determines whether the values in the HTTP request change any adapter fields.

Pascal

```
function UpdateFieldsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateFieldsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

TAdapterUpdateField (see page 5407)'s CheckValueChange method loops through the adapter fields passed in the AAdapterFields parameter, calling the CheckValueChange function. UpdateFieldsGetAnyChanges will return true if any adapter field returns true when its CheckValueChange method is called.

See Also

Errors (see page 5559)

ValidateValue (see page 5419)

IActionRequest

CheckValueChange (see page 5412)

57.1.44.4.86 TCustomAdapter.UpdateFieldsUpdateValues Method

Updates the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsUpdateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the UpdateValue function. Adapter fields raise an exception to indicate a update error. If an exception occurs, the exception message is added to the Errors (see page 5559) object and updating continues. OnBeforeUpdateFields (see page 5564) and OnAfterUpdateFields (see page 5562) are at the beginning and end of this method.

See Also

OnBeforeUpdateFields (see page 5564)

OnAfterUpdateFields (see page 5562)

Errors (see page 5559)

UpdateValue (see page 5419)

IActionRequest

57.1.44.4.87 TCustomAdapter.UpdateFieldsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, th exception messages are added to the Errors (see page 5559) object and validation continues. OnBeforeValidateFields (see page 5564) and OnAfterValidateFields (see page 5563) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 5564)

OnAfterValidateFields (see page 5563)

Errors (see page 5559)

ValidateValue (see page 5419)

IActionRequest

57.1.44.4.88 TCustomAdapter.UpdateRecords Method

Updates the adapter's fields with the field values passed in an HTTP request.

Pascal

```
function UpdateRecords(AActionRequest: IActionRequest = nil): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecords(IActionRequest AActionRequest = nil);
```

Description

Call UpdateRecords to update the adapter's fields with the field values passed in an HTTP request. Typically, UpdateRecords will be called by an adapter action that submits an HTML form. UpdatesRecords first verifies that the end-user has rights to modify the adapter by calling CheckModifyAccess (see page 5527). CheckModifyAccess (see page 5527) will raise an exception if access is denied. Next, UpdateRecordsGetAnyChanges (see page 5556) is called to determine whether the values in the HTTP request represent changes to the adapter fields. If there are changes then UpdateRecordsValidateValues (see page 5557) is called followed by UpdateRecordsUpdateValues (see page 5557). Exceptions raised during validation or update are added to the adapter's errors object. UpdateRecords returns true if the errors object contains no errors. The AActionRequest parameter is optional.

See Also

IActionRequest

checkmodifyaccess (see page 5527)

UpdateRecordsGetAnyChanges (see page 5556)

UpdateRecordsValidateValues (see page 5557)

UpdateRecordsUpdateValues (see page 5557)

Errors (see page 5559)

57.1.44.4.89 TCustomAdapter.UpdateRecordsGetAnyChanges Method

Determines whether values passed in the HTTP request will change any adapter fields.

Pascal

```
function UpdateRecordsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecordsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsGetAnyChanges iterates all the records in the HTTP request, calling UpdateFieldsGetAnyChanges (see page 5554) for each record.

See Also

UpdateFieldsGetAnyChanges (see page 5554)

57.1.44.4.90 TCustomAdapter.UpdateRecordsUpdateValues Method

Updates adapter fields with values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsUpdateValues iterates all the records in the HTTP request, calling UpdateFieldUpdateValues for each record.

See Also

UpdateFieldsUpdateValues (see page 5555)

57.1.44.4.91 TCustomAdapter.UpdateRecordsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues (see page 5555) loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, the exception message is added to the Errors (see page 5559) object and validation continues. OnBeforeValidateFields (see page 5564) and OnAfterValidateFields (see page 5563) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 5564)

OnAfterValidateFields (see page 5563)

Errors (see page 5559)

ValidateValue (see page 5419)

IActionRequest

57.1.44.4.92 TCustomAdapter.WebComponentsChange Method

WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Pascal

```
procedure WebComponentsChange(Sender: TObject);
```

C++

```
__fastcall WebComponentsChange(TObject * Sender);
```

Description

WebComponentsChange will be called, for example, when an adapter field is added to the adapter or when an adapter field is removed from the adapter. WebComponentsChange calls NotifyAdapterChange (see page 5551).

See Also

NotifyAdapterChange (see page 5551)

57.1.44.5 TCustomAdapter Properties

The properties of the TCustomAdapter class are listed here.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.

57.1.44.5.1 TCustomAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 5198)

57.1.44.5.2 TCustomAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 5265)

57.1.44.5.3 TCustomAdapter.EchoActionFieldValues Property

Modifies the adapter's echo flag.

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

Description

The echo flag is used to alter the value returned by the adapter's fields. When this flag is set, the adapter's fields retrieves their value from the HTTP request. When this flag is clear, the adapter's fields retrieve their current value. Typically, this flag is set after an error occurs while executing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValues (see page 5553)

GetEchoActionFieldValues (see page 5536)

57.1.44.5.4 TCustomAdapter.Errors Property

Returns an object used to track errors on behalf of this adapter.

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

Adapter actions typically catch exceptions raised during execution and add the exception messages to the parent adapter's errors object. Server-side script may enumerate the errors recorded by the Errors property in order to write error messages to an HTML page.

See Also

GetAdapterErrorsList (see page 5534)

TBaseAdapterErrorsList (see page 5467)

TAdapterErrorsList (see page 5248)

57.1.44.5.5 TCustomAdapter.ExecuteAccess Property

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

This is ExecuteAccess, a member of class TCustomAdapter.

57.1.44.5.6 TCustomAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TCustomAdapter.

57.1.44.5.7 TCustomAdapter.NotifyList Property

Keeps track of components that are to be notified when the adapter is changed.

Pascal

```
property NotifyList: TNotifyList;
```

C++

```
__property TNotifyList NotifyList;
```

Description

Keeps track of components that are to be notified when the adapter is changed.

See Also

NotifyAdapterChange (see page 5551)

TNotifyList (see page 910)

57.1.44.5.8 TCustomAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

implhasviewaccess (see page 5546)

57.1.44.5.9 TCustomAdapter.VisibleActions Property

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

Description

This is VisibleActions, a member of class TCustomAdapter.

57.1.44.5.10 TCustomAdapter.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class TCustomAdapter.

57.1.44.6 TCustomAdapter Events

The events of the TCustomAdapter class are listed here.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

57.1.44.6.1 TCustomAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.44.6.2 TCustomAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.44.6.3 TCustomAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnBeforeUpdateFields (see page 5564)

Errors (see page 5559)

57.1.44.6.4 TCustomAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5555)

OnBeforeValidateFields (see page 5564)

Errors (see page 5559)

57.1.44.6.5 TCustomAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.44.6.6 TCustomAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to

indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.44.6.7 TCustomAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnAfterUpdateFields (see page 5562)

Errors (see page 5559)

57.1.44.6.8 TCustomAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5555)

OnAfterValidateFields (see page 5563)

Errors (see page 5559)

57.1.44.6.9 TCustomAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.44.6.10 TCustomAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when called to retrieve their value.

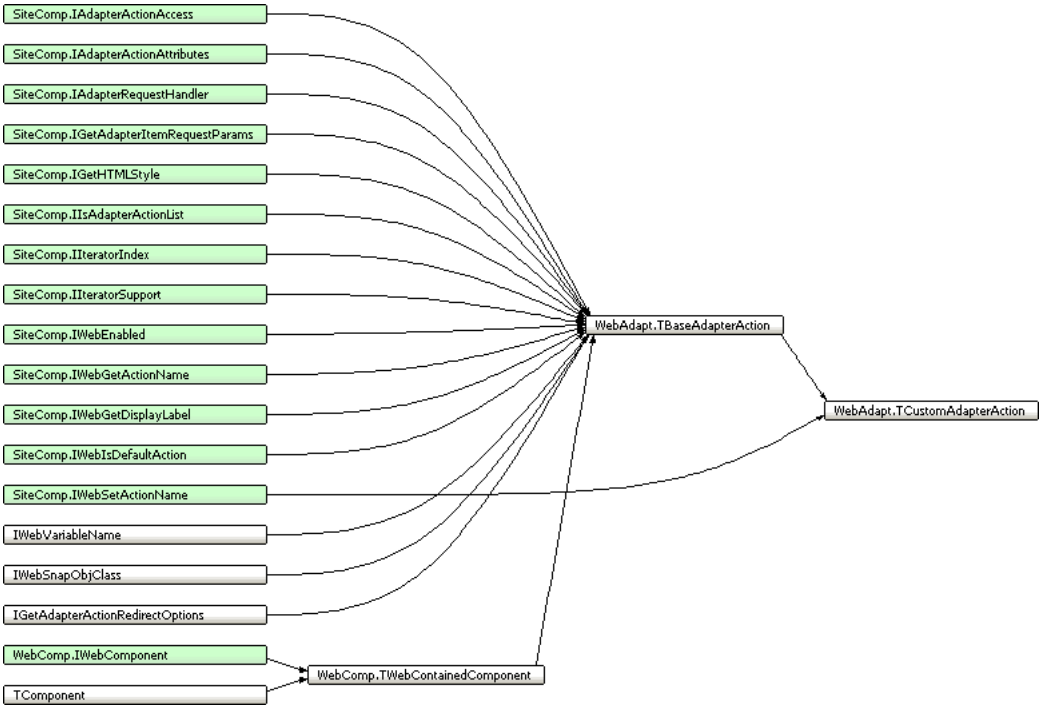
See Also

IIteratorSupport (see page 996)

57.1.45 TCustomAdapterAction Class

TCustomAdapterAction is a base class for adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterAction = class(TBaseAdapterAction, IWebSetActionName);
```

C++

```
class TCustomAdapterAction : public TBaseAdapterAction, public IWebSetActionName;
```

File

WebAdapt

Description

TCustomAdapterAction calls event handlers to retrieve the adapter action parameters and execute the adapter action. Derive from TCustomAdapterAction to define an adapter action component that can be added to any adapter. Derive from TImplementAdapterAction to define an adapter action component that works with specific adapter component types.

See Also

TBaseAdapterAction (see page 5435)

TAdapter (see page 5179)

TAdapterAction (see page 5190)

TImplementedAdapterAction (see page 5803)

57.1.45.1 TCustomAdapterAction Members

The following tables list the members exposed by TCustomAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.

	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).

	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.45.2 TCustomAdapterAction Methods

The methods of the TCustomAdapterAction class are listed here.

Protected Methods

	Name	Description
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.

	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.

57.1.45.2.1 TCustomAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute (see page 5574) event handler.

See Also

ImplExecuteActionRequest (see page 5452)

OnExecute (see page 5574)

57.1.45.2.2 TCustomAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetActionName();
```

Description

Gets the name used to reference this adapter action with server-side script.

See Also

ActionName (see page 5572)

ImplGetActionName (see page 5453)

57.1.45.2.3 TCustomAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

When this adapter action is executed, it provides name/value pairs which will be written when the HTML page is generated and passed back in the HTTP request. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted. When the HTML page is generated a hyperlink or button is written that identifies the name and parameters of the action. When the hyperlink or button is clicked the name of the action and parameters are passed back to the web application in the HTTP request.

See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.45.2.4 TCustomAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; override;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 5464) property.

See Also

RedirectOptions (see page 5464)

57.1.45.2.5 TCustomAdapterAction.ImplSetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure ImplSetActionName(const Value: string);
```

C++

```
__fastcall ImplSetActionName(const UnicodeString Value);
```

Description

ImplSetActionName is called by SetActionName (see page 5572). ImplSetActionName stores the name in an instance variable.

See Also

SetActionName (see page 5572)

57.1.45.2.6 TCustomAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled calls the OnGetEnabled (see page 5575) event handler.

See Also

ImplWebEnabled (see page 5458)

OnGetEnabled (see page 5575)

57.1.45.2.7 TCustomAdapterAction.SetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure SetActionName(const Value: string);
```

C++

```
__fastcall SetActionName(const UnicodeString Value);
```

Description

SetActionName is typically called at design time to set the action name of a new adapter action. SetActionName calls ImplSetActionName (see page 5571).

See Also

ImplSetActionName (see page 5571)

57.1.45.3 TCustomAdapterAction Properties

The properties of the TCustomAdapterAction class are listed here.

Public Properties

	Name	Description
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.

57.1.45.3.1 TCustomAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 5570)

ImplSetActionName (see page 5571)

57.1.45.4 TCustomAdapterAction Events

The events of the TCustomAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.45.4.1 TCustomAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.45.4.2 TCustomAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.45.4.3 TCustomAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.45.4.4 TCustomAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.45.4.5 TCustomAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest (see page 5570).

See Also

ImplExecuteActionRequest (see page 5570)

OnGetParams (see page 5466)

57.1.45.4.6 TCustomAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled (see page 5571) calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that this adapter action should not be enabled.

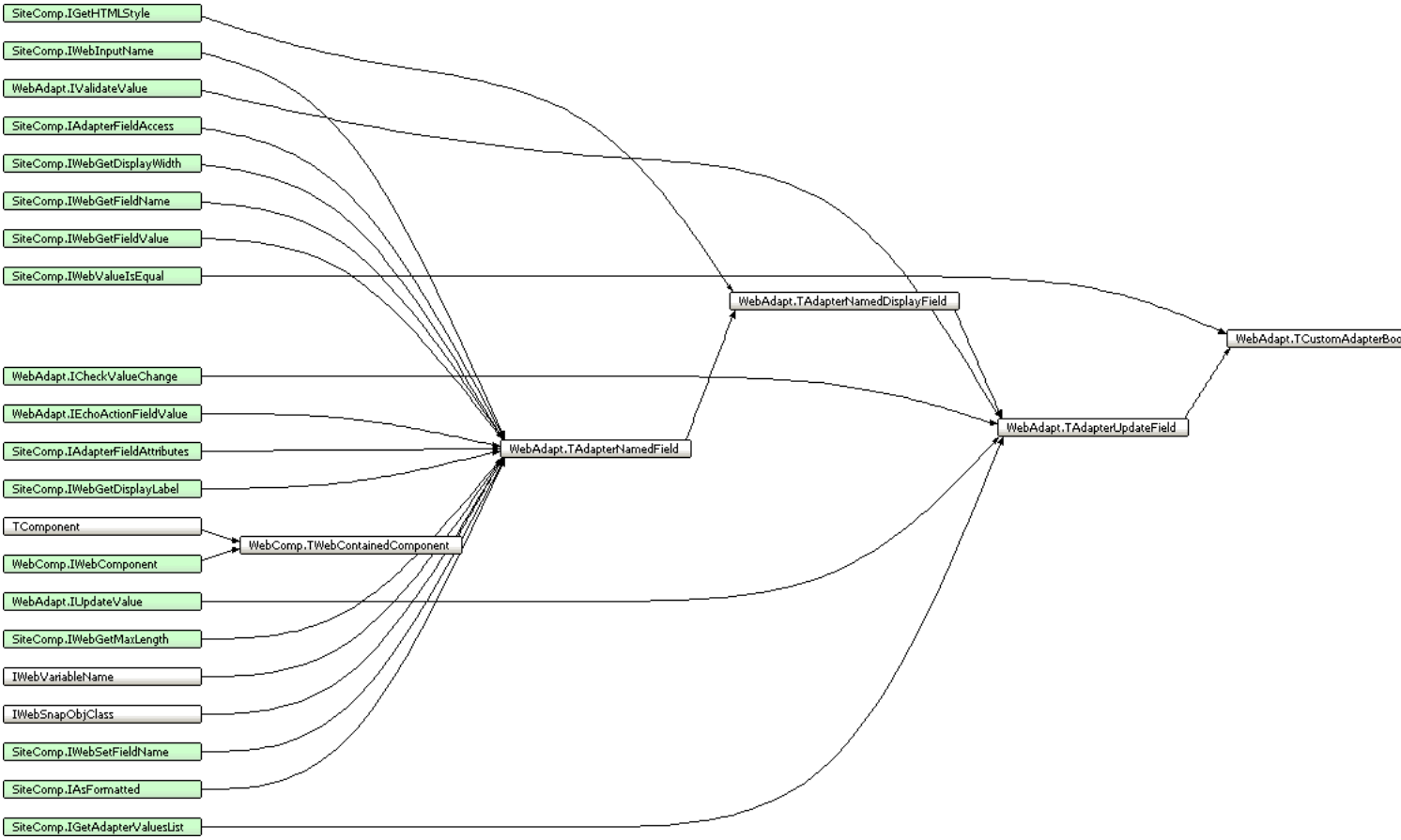
See Also

ImplWebEnabled (see page 5571)

57.1.46 TCustomAdapterBooleanField Class

tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.

Class Hierarchy



Pascal

```
TCustomAdapterBooleanField = class(TAdapterUpdateField, IWebValueIsEqual);
```

C++

```
class TCustomAdapterBooleanField : public TAdapterUpdateField, public IWebValueIsEqual;
```

File

WebAdapt

Description

tcustomadapterbooleanfield calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 5407)

TAdapterBooleanField (see page 5212)

TAdapter (see page 5179)

57.1.46.1 TCustomAdapterBooleanField Members

The following tables list the members exposed by TCustomAdapterBooleanField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5580)	Retrieves a Boolean value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5414)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5580)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5581)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.

	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5581)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5582)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5583)	Called to retrieve the adapter field's value.

	OnUpdateValue (see page 5583)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
V	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.

	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.46.2 TCustomAdapterBooleanField Methods

The methods of the TCustomAdapterBooleanField class are listed here.

Protected Methods

	Name	Description
V	ConvertValue (see page 5580)	Retrieves a Boolean value entered by the end-user.
V	GetDisplayStyleType (see page 5580)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5581)	Suggests an HTML input type for this adapter field.
V	ImplGetValue (see page 5581)	Returns the value of this adapter field as a variant.
V	ImplIsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.
V	ImplUpdateValue (see page 5582)	Called to update the adapter field's value with a value entered by the end-user.
	IsEqual (see page 5582)	Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.

57.1.46.2.1 TCustomAdapterBooleanField.ConvertValue Method

Retrieves a Boolean value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; override;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue overrides TAdapterUpdateField (see page 5407)'s ConvertValue method. Convert value attempts to cast the value entered by the end-user to a Boolean. If the cast fails, false is returned.

See Also

ConvertValue (see page 5413)

57.1.46.2.2 TCustomAdapterBooleanField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
```

```
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5331)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block (paragraph).

See Also

GetDisplayStyleType (see page 5335)

TAdapterDisplayHTMLElementType

57.1.46.2.3 TCustomAdapterBooleanField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliCheckBox indicating that a check box should be used to input this adapter field's value.

See Also

GetInputStyleType (see page 5335)

TAdapterInputHTMLElementType

57.1.46.2.4 TCustomAdapterBooleanField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5356) property is true, then the value entered by end-user is returned. If the EchoActionFieldValue (see page 5356) property is false then the OnGetValue (see page 5583) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5583)

57.1.46.2.5 TCustomAdapterBooleanField.ImplIsEqual Method

Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.

Pascal

```
function ImplIsEqual(const AValue: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsEqual(const Variant AValue);
```

Description

ImplIsEqual is called by IsEqual (see page 5582). ImplIsEqual compares the parameter value with the value associated with this adapter field.

See Also

IsEqual (see page 5582)

57.1.46.2.6 TCustomAdapterBooleanField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 5583) event handler is not nil (Delphi) or NULL (C++), it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 5583)

57.1.46.2.7 TCustomAdapterBooleanField.IsEqual Method

Indicates whether the value of this field is equal to a particular Variant (see page 3430) value.

Pascal

```
function IsEqual(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant AValue);
```

Description

IsEqual is called when server-side script calls the IsEqual method of this adapter field. Server-side script may use the IsEqual function to determine whether to check an HTML checkbox that displays this adapter field's value. IsEqual returns the result of ImplIsEqual (see page 5582).

See Also

ImplIsEqual (see page 5582)

57.1.46.3 TCustomAdapterBooleanField Events

The events of the TCustomAdapterBooleanField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5583)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5583)	Called to update the adapter field's value with a value entered by the end-user.

57.1.46.3.1 TCustomAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5581) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5581)

57.1.46.3.2 TCustomAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

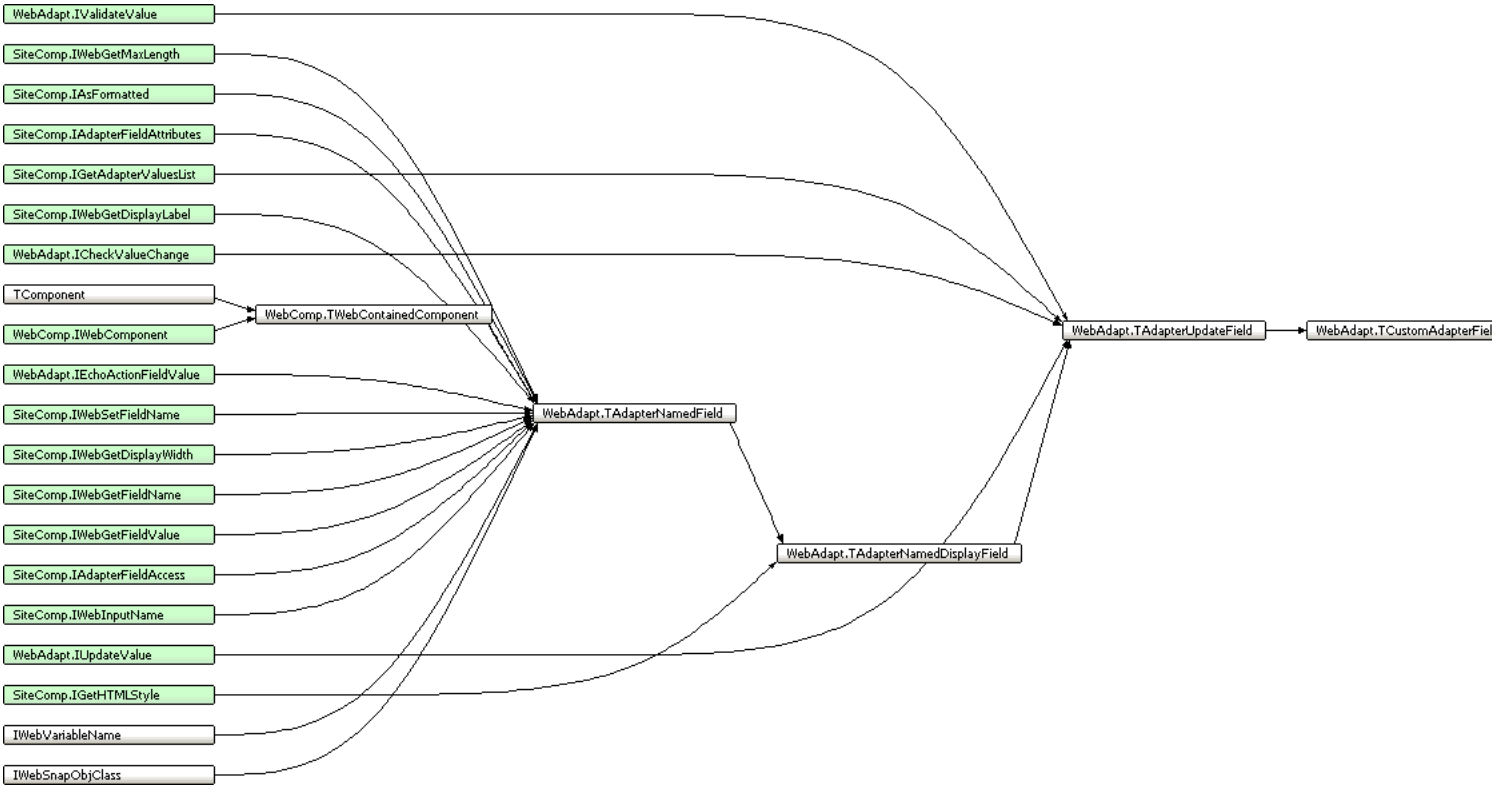
See Also

ImplUpdateValue (see page 5582)

57.1.47 TCustomAdapterField Class

TCustomAdapterField is a base class for adapter fields that provide a single value.

Class Hierarchy



Pascal

```
TCustomAdapterField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterField : public TAdapterUpdateField;
```

File

WebAdapt

Description

TCustomAdapterField calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 5407)

TAdapterField (see page 5256)

TAdapter (see page 5179)

57.1.47.1 TCustomAdapterField Members

The following tables list the members exposed by TCustomAdapterField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5588)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5335)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5335)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5414)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.

	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5588)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5589)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5589)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5590)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.

V	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
V	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.

	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.47.2 TCustomAdapterField Methods

The methods of the TCustomAdapterField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 5588)	Validates a single value entered by the end-user.
V	ImplGetValue (see page 5588)	Returns the value of this adapter field as a variant.
V	ImplUpdateValue (see page 5589)	Called to update the adapter field's value with a value entered by the end-user.

57.1.47.2.1 TCustomAdapterField.DoOnValidateValue Method

Validates a single value entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue calls the inherited DoOnValidateValue. If this succeeds by not raising an exception then additional checks are made. One check tests that the request does not contain more than one value for this adapter field. Another check is performed if the Required (see page 5339) property is True. In this case, the request must contain a value for this adapter field. An exception is raised if any of these checks fail.

See Also

DoOnValidateValue (see page 5414)

Required (see page 5339)

57.1.47.2.2 TCustomAdapterField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5356) property is True, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 5356) property is False then the OnGetValue (see page 5589) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5589)

CheckViewAccess (see page 5344)

EchoActionFieldValue (see page 5356)

57.1.47.2.3 TCustomAdapterField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value using the given action request. The AfieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated agains the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 5590) event handler is not nil, it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 5590)

57.1.47.3 TCustomAdapterField Events

The events of the TCustomAdapterField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5589)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5590)	Called to update the adapter field's value with a value entered by the end-user.

57.1.47.3.1 TCustomAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5588) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5588)

57.1.47.3.2 TCustomAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

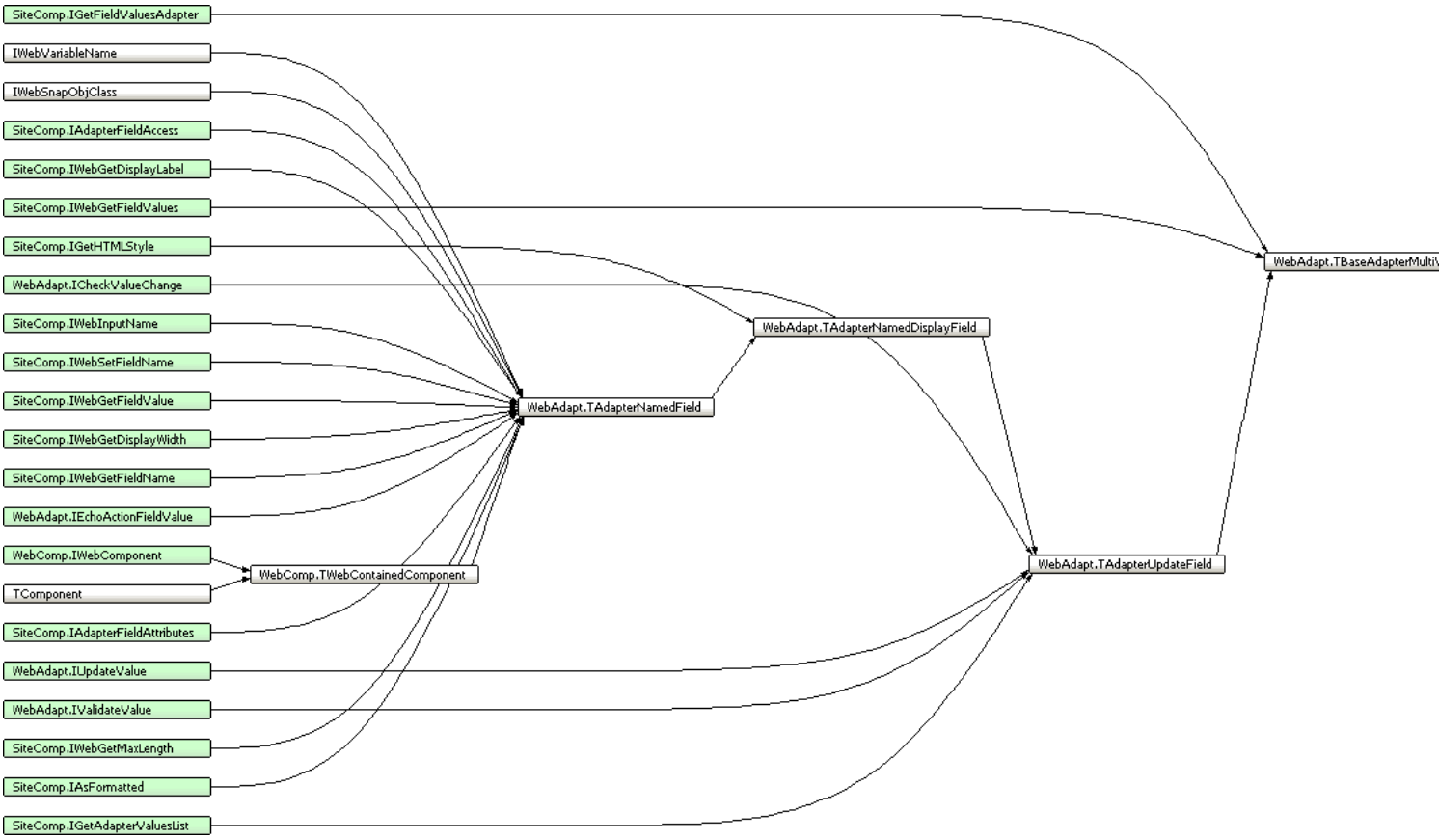
Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

57.1.48 TCustomAdapterFileField Class

TCustomAdapterFileField is a base class for adapter fields that can have associated files.

Class Hierarchy



Pascal

```
TCustomAdapterFileField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterFileField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterFileField when the adapter field can have zero or more associated files. Typically, file input HTML elements are used to upload files. Multiple file input HTML elements having the same name can be used to upload multiple files in a single request.

See Also

TBaseAdapterMultiValueField (see page 5475)

TAdapterFileField (see page 5270)

TAdapterImageField (see page 5297)

57.1.48.1 TCustomAdapterFileField Members

The following tables list the members exposed by TCustomAdapterFileField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5595)	Validates files uploaded by the end-user.
	EndValueIterator (see page 5481)	Completes iteration.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5595)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
	GetFileNames (see page 5596)	Returns the count of files associated with this adapter field.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5596)	Suggests an HTML input type for this adapter field.

	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetValueCount (see page 5596)	Returns the number of values associated with this adapter field.
	GetValues (see page 5597)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5597)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5484)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5597)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.
	Notification (see page 5419)	Clears references to a deleted component.

	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetFileCount (see page 5599)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5599)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5599)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5600)	Validates values entered by the end-user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 5480)	Constructor for TBaseAdapterMultiValueField (see page 5475).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 5480)	Destructor for TBaseAdapterMultiValueField (see page 5475).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.

V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
R	FileCount (see page 5598)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 5598)	Returns a file associated with this adapter field.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.48.2 TCustomAdapterFileField Methods

The methods of the TCustomAdapterFileField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 5595)	Validates files uploaded by the end-user.
	GetDisplayStyleType (see page 5595)	Suggests an HTML display type for this adapter field.
	GetFileNames (see page 5596)	Returns the count of files associated with this adapter field.
	GetInputStyleType (see page 5596)	Suggests an HTML input type for this adapter field.
	GetValueCount (see page 5596)	Returns the number of values associated with this adapter field.
	GetValues (see page 5597)	Returns one of the values associated with the adapter field.
	ImplCheckValueChange (see page 5597)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplUpdateValue (see page 5597)	Updates the adapter field with value(s) entered by the end-user.

57.1.48.2.1 TCustomAdapterFileField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5407)'s DoOnValidateValue method. If the OnValidateFiles (see page 5600) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate uploaded files. If the Required (see page 5339) property is true, DoOnValidateValues also verifies that 1 of more values have been entered by the end-user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5414)

OnValidateValue (see page 5421)

Required (see page 5339)

OnValidateFiles (see page 5600)

57.1.48.2.2 TCustomAdapterFileField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5331)'s GetDisplayStyleType. GetDispayStyleTypereturns htmldList, indicating that this adapter field should be displayed by as a list of read-only values.

See Also

GetDisplayStyle (see page 5334)

TAdapterDisplayHTMLElementType

57.1.48.2.3 TCustomAdapterFileField.GetFileNames Method

Returns the count of files associated with this adapter field.

Pascal

```
function GetFileNames(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetFileNames(int I);
```

Description

GetFileNames implements the FileNames (see page 5598) property. If the EchoActionFieldValue (see page 5356) flag is true then the filename is extracted from the HTTP request. Otherwise, the OnGetFileNames (see page 5599) event is called. GetFileNames returns and empty string if a value can't be found because OnGetFileNames (see page 5599) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a filename for this adapter field.

See Also

FileNames (see page 5598)

OnGetFileNames (see page 5599)

57.1.48.2.4 TCustomAdapterFileField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that file upload HTML elements should be used to input values for this adapter field.

See Also

GetInputStyleType (see page 5335)

TAdapterInputHTMLElementType

57.1.48.2.5 TCustomAdapterFileField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the FileCount (see page 5598) property. If the EchoActionFieldValue (see page 5356) flag is

true then the result is the number of values in the HTTP request for this adapter field. Otherwise, the OnGetFileCount (see page 5599) event is called. GetValueCount returns zero if a value can't be found because OnGetFileCount (see page 5599) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

FileCount (see page 5598)

OnGetFileCount (see page 5599)

57.1.48.2.6 TCustomAdapterFileField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns the result of GetFileNames (see page 5596).

See Also

GetFileNames (see page 5596)

57.1.48.2.7 TCustomAdapterFileField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded new files for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5407)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded any files.

See Also

IActionRequest

ImplCheckValueChange (see page 5416)

57.1.48.2.8 TCustomAdapterFileField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 5407)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUploadFiles (see page 5599) event is called.

See Also

IActionRequest

OnUploadFiles (see page 5599)

ImplUpdateValue (see page 5418)

57.1.48.3 TCustomAdapterFileField Properties

The properties of the TCustomAdapterFileField class are listed here.

Public Properties

	Name	Description
	FileCount (see page 5598)	Returns one of the filenames associated with this adapter field.
	FileNames (see page 5598)	Returns a file associated with this adapter field.

57.1.48.3.1 TCustomAdapterFileField.FileCount Property

Returns one of the filenames associated with this adapter field.

Pascal

```
property FileCount: Integer;
```

C++

```
__property int FileCount;
```

Description

```
__property int FileCount = {read=GetValueCount, nodefault};
```

FileCount can be used with the FileNames (see page 5598) property to access all the filenames associated with this adapter field. FileCount is implemented by GetValueCount (see page 5596).

See Also

FileNames (see page 5598)

GetValueCount (see page 5596)

57.1.48.3.2 TCustomAdapterFileField.FileNames Property

Returns a file associated with this adapter field.

Pascal

```
property FileNames [I: Integer]: string;
```

C++

```
__property UnicodeString FileNames[int I];
```

Description

FileNames can be used with the FileCount (see page 5598) property to access all the filenames associated with this adapter field. FileNames is implemented by GetFileNames (see page 5596).

See Also

FileCount (see page 5598)

GetFileNames (see page 5596)

57.1.48.4 TCustomAdapterFileField Events

The events of the TCustomAdapterFileField class are listed here.

Public Events

	Name	Description
	OnGetFileCount (see page 5599)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5599)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5599)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5600)	Validates values entered by the end-user.

57.1.48.4.1 TCustomAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write (see page 3789) an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 5596)

57.1.48.4.2 TCustomAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write (see page 3789) an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 5596)

57.1.48.4.3 TCustomAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue (see page 5597). Write (see page 3789) an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5597)

TUpdateFileList (see page 5850)

57.1.48.4.4 TCustomAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue (see page 5595). Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

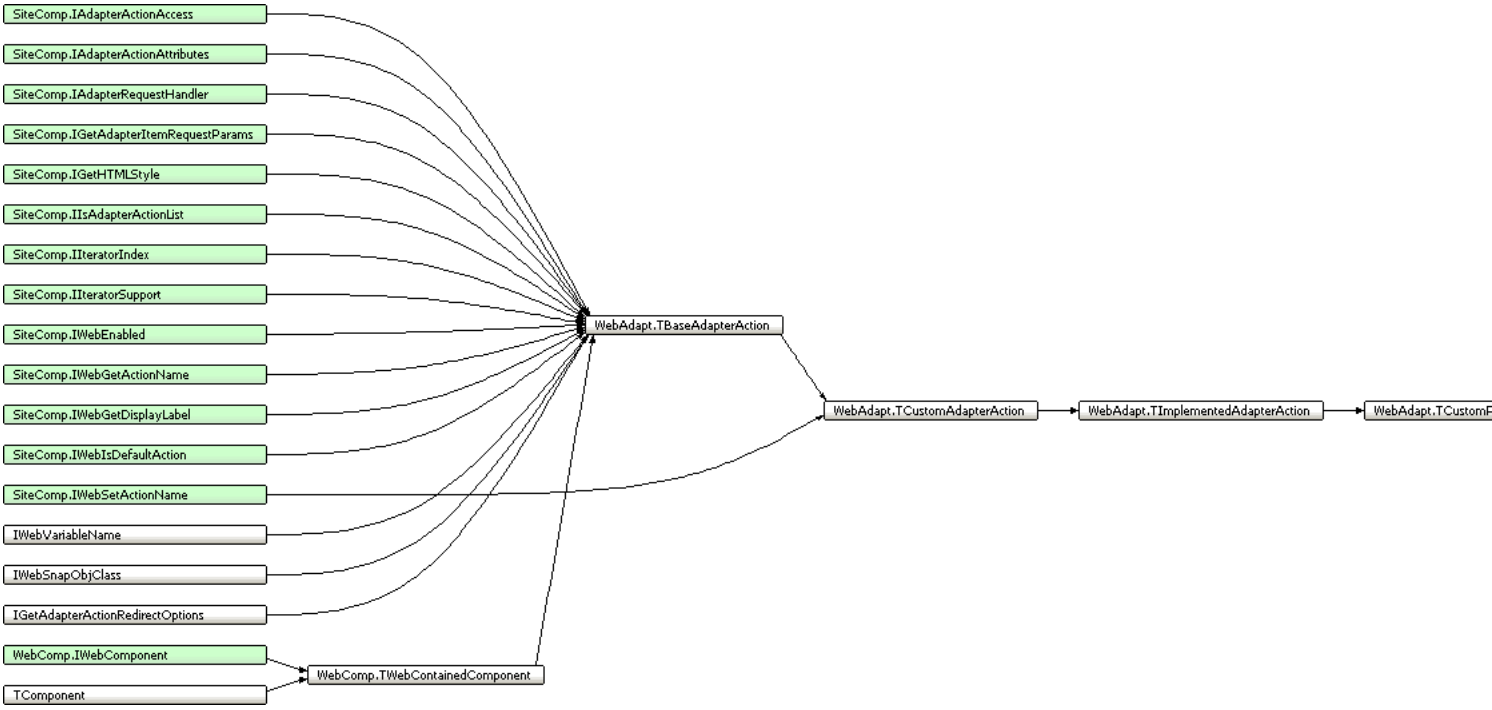
DoOnValidateValue (see page 5595)

TUpdateFileList (see page 5850)

57.1.49 TCustomAdapterGotoPageAction Class

TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5733).

Class Hierarchy



Pascal

```
TCustomAdapterGotoPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterGotoPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TCustomAdapterGotoPageAction examines the parameter and changes the current page in the parent adapter.

See Also

TCustomPagedAdapterAction (see page 5708)

TAdapterGotoPageAction (see page 5280)

57.1.49.1 TCustomAdapterGotoPageAction Members

The following tables list the members exposed by TCustomAdapterGotoPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.

	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5605)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5605)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5606)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5606)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5606)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5607)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5607)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5608)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5608)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetPageName (see page 5608)	Provides the display labels for each goto page command.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.49.2 TCustomAdapterGotoPageAction Methods

The methods of the TCustomAdapterGotoPageAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5605)	Gets the name used to reference this adapter action with server-side script.
	GetDisplayStyleType (see page 5605)	Suggests an HTML display type for this adapter action.
	GetItemCount (see page 5606)	Returns the number of commands defined by this adapter action.
	ImplExecuteActionRequest (see page 5606)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5606)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5607)	Returns the suggested display label for this adapter action.
	ImplGetIsAdapterActionList (see page 5607)	Indicates whether this adapter action may have multiple commands.
	ImplIsDefaultAction (see page 5608)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 5608)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.49.2.1 TCustomAdapterGotoPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns 'GotoPage'.

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.49.2.2 TCustomAdapterGotoPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that goto page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 5445)

57.1.49.2.3 TCustomAdapterGotoPageAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

TCustomAdapterGotoPageAction (see page 5600) returns the value of the parent adapter's PageCount property.

See Also

ItemCount (see page 5463)

ItemIndex (see page 5464)

GetIsAdapterActionList (see page 5448)

PageCount (see page 5747)

57.1.49.2.4 TCustomAdapterGotoPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest sets the parent adapter's current page using the page index passed in the AActionRequest parameter.

See Also

ImplExecuteActionRequest (see page 5452)

57.1.49.2.5 TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5444). ImplGetAdapterItemRequestParams adds the value of the iteration index as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

GetAdapterItemRequestParams (see page 5444)

OnGetActionParams (see page 5564)

OnGetParams (see page 5466)

ItemIndex (see page 5464)

57.1.49.2.6 TCustomAdapterGotoPageAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the display label suggested for the goto page command that is identified by the iteration index. If the OnGetPageName (see page 5608) event handler is not nil (Delphi) or NULL (C++), it is called. Otherwise, the ItemIndex (see page 5464) property is converted to a string and returned.

See Also

DisplayLabel (see page 5462)

ItemIndex (see page 5464)

OnGetPageName (see page 5608)

57.1.49.2.7 TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 5448). ImplGetIsAdapterActionList returns true if the parent adapters PageSize property is greater than zero.

See Also

GetIsAdapterActionList (see page 5448)

PageSize (see page 5747)

57.1.49.2.8 TCustomAdapterGotoPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5459). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate that the goto page commands are appropriate.

See Also

IsDefaultAction (see page 5459)

57.1.49.2.9 TCustomAdapterGotoPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of false if the iteration index is equal to the parent adapter's current page. In other words, the command that goes to the parent adapter's current page will be disabled. Commands to goto all other pages will be enabled.

See Also

WebEnabled (see page 5461)

CurrentPage (see page 5746)

ItemIndex (see page 5464)

57.1.49.3 TCustomAdapterGotoPageAction Events

The events of the TCustomAdapterGotoPageAction class are listed here.

Public Events

	Name	Description
	OnGetPageName (see page 5608)	Provides the display labels for each goto page command.

57.1.49.3.1 TCustomAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

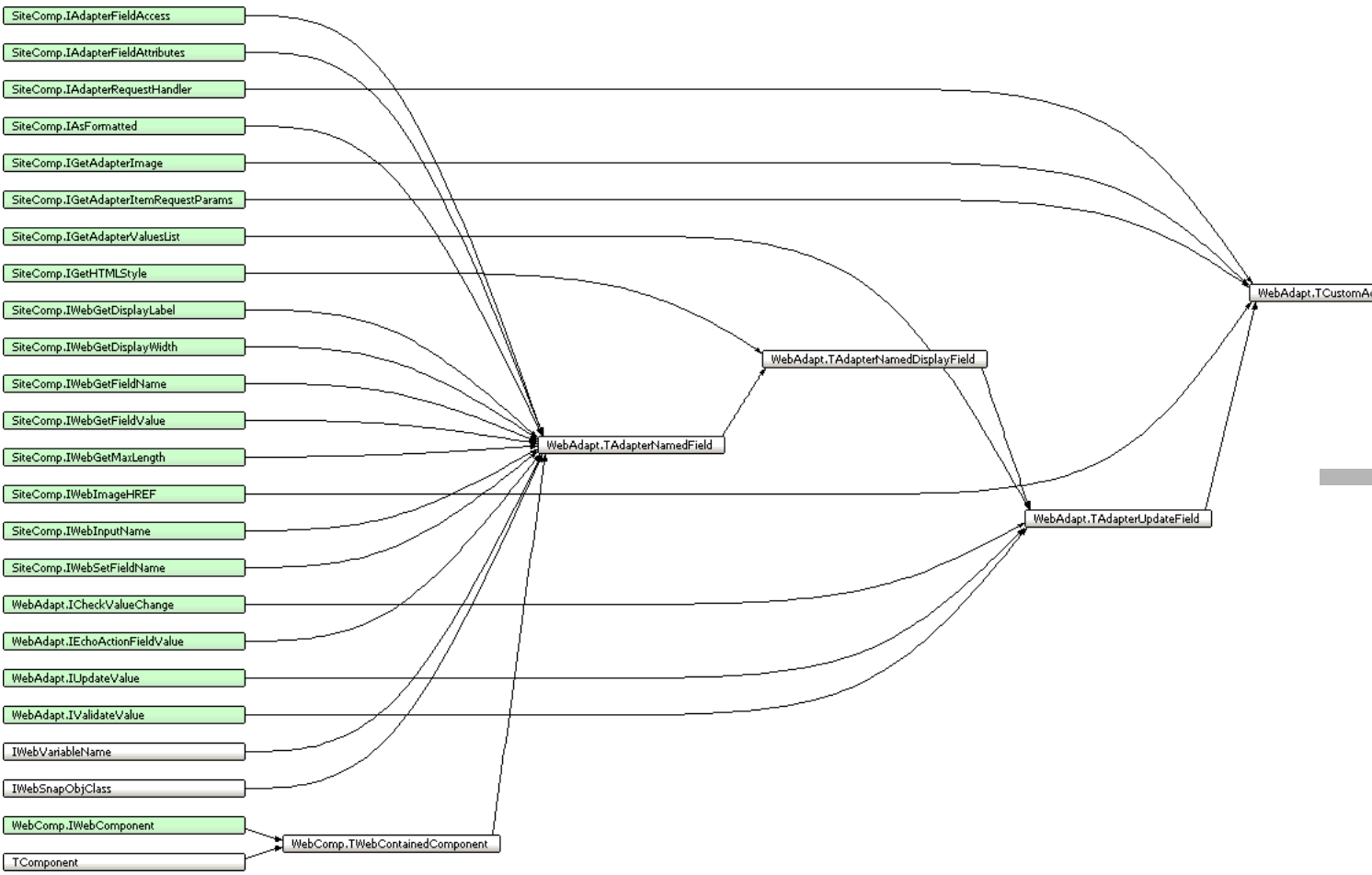
See Also

ImplGetDisplayLabel (see page 5607)

57.1.50 TCustomAdapterImageField Class

TCustomAdapterImageField is a base class for adapter fields that have an associated image file.

Class Hierarchy



Pascal

```
TCustomAdapterImageField = class(TAdapterUpdateField, IGetAdapterImage,
IGetAdapterItemRequestParams, IAdapterRequestHandler, IWebImageHREF);
```

C++

```
class TCustomAdapterImageField : public TAdapterUpdateField, public IGetAdapterImage, public
IGetAdapterItemRequestParams, public IAdapterRequestHandler, public IWebImageHREF;
```

File

WebAdapt

Description

Derived from TCustomAdapterImageField when the adapter field can an associated image file. Typically, a file input HTML element is used to upload an image file. HTML IMG elements are used to display the image.

See Also

TAdapterImageField (see page 5297)

TAdapterFileField (see page 5270)

57.1.50.1 TCustomAdapterImageField Members

The following tables list the members exposed by TCustomAdapterImageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 5614)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 5614)	Validates files uploaded by the end-user.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 5615)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 5615)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5616)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5616)	Suggests an HTML input type for this adapter field.

	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HandleRequest (see page 5616)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5617)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5617)	Creates AdapterRequest and AdapterResponse objects.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5618)	Returns the name of the image file associated with this adapter field.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 5618)	Handles a request for the image file associated with this adapter field.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5619)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 5619)	Returns a URL that can be used to access the image file associated with this adapter field.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.

	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 5619)	Responds to a request for an image file by providing a stream containing the image.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 5620)	Returns a URL that can be used to access the image file associated with this adapter field.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetHREF (see page 5621)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5622)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5622)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5622)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5623)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5623)	Used to validate files uploaded by the end user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.

	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	HREF (see page 5620)	Defines a URL that references the image associated with the adapter field.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5621)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.50.2 TCustomAdapterImageField Methods

The methods of the TCustomAdapterImageField class are listed here.

Protected Methods

	Name	Description
	CreateRequestContext (see page 5614)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 5614)	Validates files uploaded by the end-user.
	GetAdapterImage (see page 5615)	Provides an object that can be used to access the image properties of this adapter field.

	GetAdapterItemRequestParams (see page 5615)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
V	GetDisplayStyleType (see page 5616)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5616)	Suggests an HTML input type for this adapter field.
	HandleRequest (see page 5616)	Handles a request for the image file associated with this adapter field.
V	ImplCheckValueChange (see page 5617)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5617)	Creates AdapterRequest and AdapterResponse objects.
V	ImplGetValue (see page 5618)	Returns the name of the image file associated with this adapter field.
V	ImplHandleRequest (see page 5618)	Handles a request for the image file associated with this adapter field.
V	ImplUpdateValue (see page 5619)	Updates the adapter field with value entered by the end-user.
V	ImplWebImageHREF (see page 5619)	Returns a URL that can be used to access the image file associated with this adapter field.
V	RenderAdapterImage (see page 5619)	Responds to a request for an image file by providing a stream containing the image.
	WebImageHREF (see page 5620)	Returns a URL that can be used to access the image file associated with this adapter field.

57.1.50.2.1 TCustomAdapterImageField.CreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext implements IAdapterRequestHandler (see page 954)'s CreateRequestContext method. CreateRequestContext calls ImplCreateRequestContext (see page 5617).

See Also

IAdapterDispatchParams (see page 929)

IAdapterRequestHandler (see page 954)

AdapterRequest (see page 5946)

AdapterResponse (see page 5946)

ImplCreateRequestContext (see page 5617)

57.1.50.2.2 TCustomAdapterImageField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5407)'s DoOnValidateValue method. If the OnValidateFile (see page 5623) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate an uploaded file. DoOnValidateValues also validates that not more than one image file was uploaded and, if the required property is true, that a

single file was uploaded. An exception is raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5414)

OnValidateValue (see page 5421)

Required (see page 5339)

OnValidateFile (see page 5623)

57.1.50.2.3 TCustomAdapterImageField.GetAdapterImage Method

Provides an object that can be used to access the image properties of this adapter field.

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

GetAdapterImage returns Self. GetAdapterImage will be called when server-side script references the Image property of this adapter field.

57.1.50.2.4 TCustomAdapterImageField.GetAdapterItemRequestParams Method

Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

WebImageHREF (see page 5620) and GetAdapterItemRequestParams are used to identify the URL that is used to retrieve the image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. If WebImageHREF (see page 5620) returns False then GetAdapterItemRequestParams is called. WebImageHREF (see page 5620) is responsible for returning a reference to an external object. GetAdapterItemRequestParams is used to generate a URL that will result in a request to the adapter field to dynamically generate the image file. GetAdapterItemRequestParams calls OnGetParams (see page 5622) to get the name/value pairs.

See Also

IAdapterItemRequestParams (see page 947)

RenderAdapterImage (see page 5619)

OnGetParams (see page 5622)

WebImageHREF (see page 5620)

TAdapterDispatcher (see page 6122)

57.1.50.2.5 TCustomAdapterImageField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5331)'s GetDisplayStyleType method. GetDispayStyleType returns htmldImage, indicating that this adapter field should be displayed by an HTML image element.

See Also

GetDisplayStyleType (see page 5335)

57.1.50.2.6 TCustomAdapterImageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that a file upload HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5335)

57.1.50.2.7 TCustomAdapterImageField.HandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

HandleRequest implements IAdapterRequestHandler (see page 954)'s HandleRequest method. HandleRequest calls ImplHandleRequest (see page 5618).

See Also

IAdapterDispatchParams (see page 929)

ImplHandleRequest (see page 5618)

IAdapterRequestHandler (see page 954)

57.1.50.2.8 TCustomAdapterImageField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded a new image file for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5407)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded an image file.

See Also

IActionRequest

ImplCheckValueChange (see page 5416)

57.1.50.2.9 TCustomAdapterImageField.ImplCreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext creates a new request and response objects based on the specified parameters and stores it with the global Web context. The adapter dispatcher calls this function before calling HandleRequest (see page 5616). The WebSnap can access the resulting request and response context by reading the AdapterRequest and AdapterResponse properties of the Web context, which is available through the global WebContext (see page 5977) function.

DispatchParams is an interface that contains the parameters that were included in the incoming request. These parameters can then be used to identify an image for this field's value.

See Also

CreateRequestContext (see page 5614)

AdapterRequest (see page 5946)

AdapterResponse (see page 5946)

IAdapterDispatchParams (see page 929)

WebContext (see page 5977)

57.1.50.2.10 TCustomAdapterImageField.ImplGetValue Method

Returns the name of the image file associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the name of the image file associated with this adapter field

First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5356) property is true, then the file name entered by end-user is returned. If the EchoActionFieldValue (see page 5356) property is false then the OnGetImageName (see page 5622) event handler is called to retrieve the name of the image file. The value returned by ImplGetValue is used when server-side script references the Value (see page 5420) or DisplayText properties.

See Also

OnGetImageName (see page 5622)

EchoActionFieldValue (see page 5356)

57.1.50.2.11 TCustomAdapterImageField.ImplHandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 5616). ImplHandleRequest extracts IImageRequest and IImageResponse from the AdapterRequest and AdapterResponse. ImplHandleRequest returns the result of RenderAdapterImage (see page 5619).

See Also

IAdapterDispatchParams (see page 929)

ImplHandleRequest

AdapterRequest (see page 5946)

AdapterResponse (see page 5946)

RenderAdapterImage (see page 5619)

IImageRequest

IImageRequest

57.1.50.2.12 TCustomAdapterImageField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request single file for this adapter field, the OnUploadFile (see page 5623) event handler is called.

See Also

OnUploadFile (see page 5623)

ImplUpdateValue (see page 5418)

57.1.50.2.13 TCustomAdapterImageField.ImplWebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function ImplWebImageHREF(var AHREF: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebImageHREF(UnicodeString AHREF);
```

Description

ImplWebImageHREF is called by WebImageHREF (see page 5620). If the OnGetHREF (see page 5621) event handler is not nil (Delphi) or NULL (C++), ImplWebImageHREF will call it passing the value of the HREF (see page 5620) property.

See Also

WebImageHREF (see page 5620)

HREF (see page 5620)

OnGetHREF (see page 5621)

57.1.50.2.14 TCustomAdapterImageField.RenderAdapterImage Method

Responds to a request for an image file by providing a stream containing the image.

Pascal

```
procedure RenderAdapterImage(ARequest: IImageRequest; AResponse: IImageResponse); virtual;
```

C++

```
virtual __fastcall RenderAdapterImage(IImageRequest ARequest, IImageResponse AResponse);
```

Description

RenderAdapterImage is called when the TAdapterDispatcher detects a request for an image from this adapter field. First, the

access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the OnGetImage (see page 5622) event handler is not nil (Delphi) or NULL (C++), it is called to retrieve the image file as a stream.

See Also

OnGetImage (see page 5622)

TAdapterDispatcher (see page 6122)

ImplHandleRequest (see page 5618)

57.1.50.2.15 TCustomAdapterImageField.WebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(UnicodeString AHREF);
```

Description

WebImageHREF and GetAdapterItemRequestParams (see page 5615) are used to identify the URL that is used to retrieve an image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. WebImageHREF returns the result of ImplWebImageHREF (see page 5619).

See Also

ImplWebImageHREF (see page 5619)

GetAdapterItemRequestParams (see page 5615)

57.1.50.3 TCustomAdapterImageField Properties

The properties of the TCustomAdapterImageField class are listed here.

Public Properties

	Name	Description
	HREF (see page 5620)	Defines a URL that references the image associated with the adapter field.
	ReadOnly (see page 5621)	Indicates whether this adapter field is read-only.

57.1.50.3.1 TCustomAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property UnicodeString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF (see page 5621) event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage (see page 5622) and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF (

see page 5621) event handler.

See Also

ImplWebImageHREF (see page 5619)

OnGetHREF (see page 5621)

OnGetImage (see page 5622)

OnGetImageName (see page 5622)

57.1.50.3.2 TCustomAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5336)

57.1.50.4 TCustomAdapterImageField Events

The events of the TCustomAdapterImageField class are listed here.

Public Events

	Name	Description
	OnGetHREF (see page 5621)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5622)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5622)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5622)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5623)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5623)	Used to validate files uploaded by the end user.

57.1.50.4.1 TCustomAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When

OnGetHREF is called, the HREF (see page 5620) parameter will contain the value of the HREF (see page 5620) property. To dynamically generate an image file for this adapter file, implement the OnGetImage (see page 5622) event handler instead.

See Also

ImplWebImageHREF (see page 5619)

HREF (see page 5620)

OnGetImage (see page 5622)

57.1.50.4.2 TCustomAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF (see page 5621) event handler or use the HREF (see page 5620) property to specify the URL of an external image.

See Also

RenderAdapterImage (see page 5619)

OnGetParams (see page 5622)

OnGetHREF (see page 5621)

HREF (see page 5620)

57.1.50.4.3 TCustomAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 5618)

57.1.50.4.4 TCustomAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage (see page 5622) when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 5615)

OnGetImage (see page 5622)

57.1.50.4.5 TCustomAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue (see page 5619). This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5619)

TAbstractWebRequestFile

57.1.50.4.6 TCustomAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

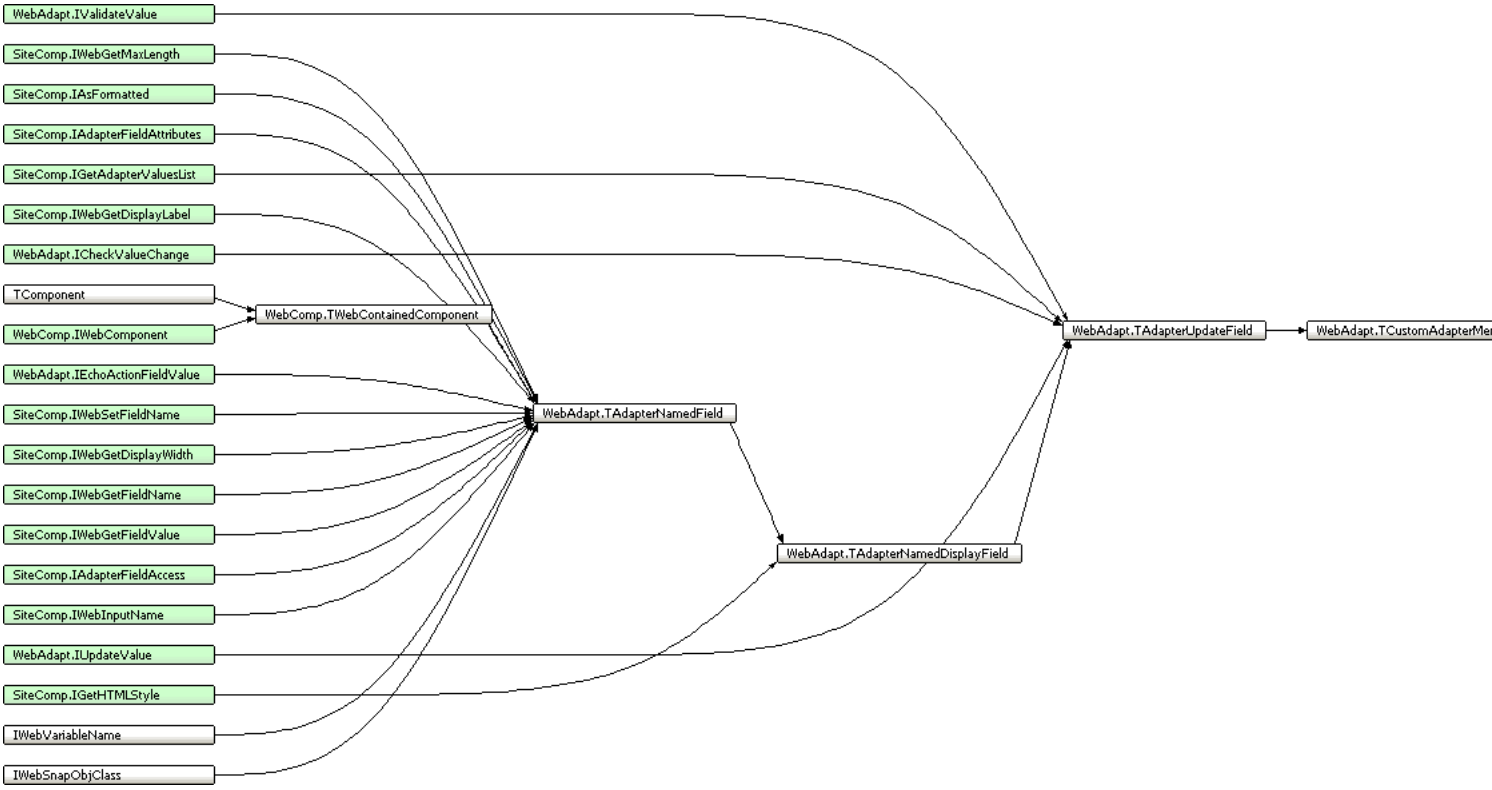
DoOnValidateValue (see page 5614)

TAbstractWebRequestFile

57.1.51 TCustomAdapterMemoField Class

TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

Class Hierarchy



Pascal

```
TCustomAdapterMemoField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterMemoField : public TAdapterUpdateField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMemoField when the adapter field has a multi-line text value. Typically, a text area HTML element is used to edit the value of this adapter field.

See Also

TAdapterUpdateField (see page 5407)

TAdapterMemoField (see page 5312)

57.1.51.1 TCustomAdapterMemoField Members

The following tables list the members exposed by TCustomAdapterMemoField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5628)	Validates a value entered by the end user.
	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5628)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5629)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetValueCount (see page 5415)	Provides the number of values associated with this adapter field.
	GetValues (see page 5415)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5629)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.

	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5630)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5630)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5419)	Clears references to a deleted component.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5631)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5631)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5631)	Called to update the adapter field's value with a file uploaded by the end-user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
V	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.

	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5357)	Provides the value of an adapter field as a variant.
	Value (see page 5420)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.51.2 TCustomAdapterMemoField Methods

The methods of the TCustomAdapterMemoField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 5628)	Validates a value entered by the end user.
	GetDisplayStyleType (see page 5628)	Suggests an HTML display type for this adapter field.
	GetInputStyleType (see page 5629)	Suggests an HTML input type for this adapter field.
	ImplCheckValueChange (see page 5629)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetValue (see page 5630)	Returns the value of this adapter field as a variant.
	ImplUpdateValue (see page 5630)	Updates the adapter field with value entered by the end-user.

57.1.51.2.1 TCustomAdapterMemoField.DoOnValidateValue Method

Validates a value entered by the end user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5407)'s DoOnValidateValue method. After calling TAdapterUpdateField (see page 5407)'s DoOnValidateValue, DoOnValidate checks that only a single text file or file has been uploaded for this field. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5414)

Required (see page 5339)

IActionFieldValue

OnValidateFiles (see page 5600)

57.1.51.2.2 TCustomAdapterMemoField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5331)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block.

See Also

GetDisplayStyleType (see page 5335)

TAdapterDisplayHTMLElementType

57.1.51.2.3 TCustomAdapterMemoField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetDispayStyleTypereturns htmliTextArea, indicating that this adapter field's value should be input with an HTML multiline input field.

See Also

GetInputStyleType (see page 5335)

TAdapterInputHTMLElementType

57.1.51.2.4 TCustomAdapterMemoField.ImplCheckValueChange Method

Indicates whether the end-user may have entered a new value for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5407)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the HTTP request has a new value for this adapter field.

See Also

ImplCheckValueChange (see page 5416)

57.1.51.2.5 TCustomAdapterMemoField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5356) property is true, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 5356) property is false then the OnGetValue (see page 5631) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5631)

EchoActionFieldValue (see page 5356)

57.1.51.2.6 TCustomAdapterMemoField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request contains a string value for this adapter field, the OnUpdateValue (see page 5631) event handler is called. If the action request contains an uploaded file for this adapter field, the OnUploadFile (see page 5631) event handler is called.

See Also

OnUpdateValue (see page 5631)

OnUploadFile (see page 5631)

ImplUpdateValue (see page 5418)

57.1.51.3 TCustomAdapterMemoField Events

The events of the TCustomAdapterMemoField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5631)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5631)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5631)	Called to update the adapter field's value with a file uploaded by the end-user.

57.1.51.3.1 TCustomAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5630) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5630)

57.1.51.3.2 TCustomAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5630)

57.1.51.3.3 TCustomAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

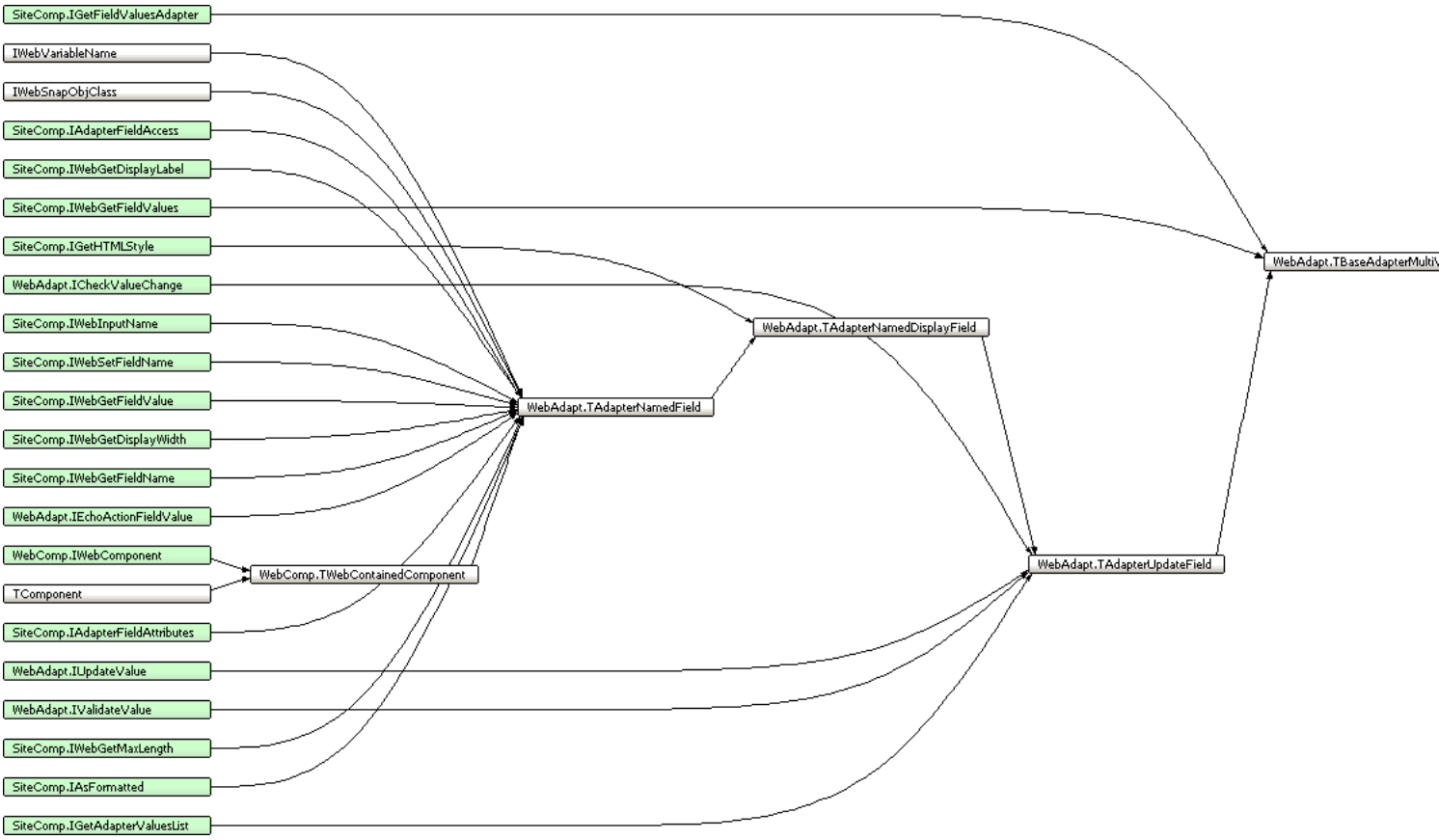
See Also

ImplUpdateValue (see page 5630)

57.1.52 TCustomAdapterMultiValueField Class

TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.

Class Hierarchy



Pascal

```
TCustomAdapterMultiValueField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterMultiValueField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMultiValueField when the adapter field can have multiple values. Typically, multi-value fields are represented in HTML with multiple selection select lists or check boxes.

See Also

TBaseAdapterMultiValueField (see page 5475)

TAdapterMultiValueField (see page 5320)

57.1.52.1 TCustomAdapterMultiValueField Members

The following tables list the members exposed by TCustomAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5412)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5413)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5413)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5413)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5637)	Validates values entered by the end-user.
	EndValueIterator (see page 5481)	Completes iteration.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5414)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5481)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5345)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5334)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5637)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5482)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5335)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5637)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
V	GetValueCount (see page 5638)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5638)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5336)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5336)	Suggests a view mode for this adapter field.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' HasValue method.

	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5415)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5416)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetCurrentValue (see page 5483)	Implements IWebGetFieldValues (see page 1034)' GetCurrentValue (see page 5481) method.
V	ImplGetDisplayLabel (see page 5337)	Implements interface access to the DisplayLabel (see page 5338) property.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5416)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5337)	Implements interface access to the DisplayWidth (see page 5338) property.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5416)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5417)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetFieldValuesAdapter (see page 5483)	Returns the adapter object that implements the Values property.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5337)	Implements interface access to the Required (see page 5339) property.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5351)	Gets the value of this adapter field as a Variant (see page 3430).
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 5417)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasValue (see page 5639)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetEchoActionFieldValue (see page 5417)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 5417)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 5639)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 5418)	Validates values(s) entered by the end-user.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	IsMultiValueField (see page 5418)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5484)	Implements IWebGetFieldValues (see page 1034)' NextValueIteration method.
V	Notification (see page 5419)	Clears references to a deleted component.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5337)	SetDisplayLabel sets the value of the DisplayLabel (see page 5338) property.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartValueIterator (see page 5484)	Implements IWebGetFieldValues (see page 1034)' StartValueIterator method.
	UpdateValue (see page 5419)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5419)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5421)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValueCount (see page 5640)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5641)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5641)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5642)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValue (see page 5421)	Event handler called to validate value(s) entered by the end-user.
	OnValidateValues (see page 5642)	Validates field values entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5334)	Creates an instance of TAdapterNamedDisplayField (see page 5331).
	Create (see page 5411)	Creates an instance of TAdapterUpdateField (see page 5407).
	Create (see page 5480)	Constructor for TBaseAdapterMultiValueField (see page 5475).
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 5480)	Destructor for TBaseAdapterMultiValueField (see page 5475).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndValueIterator (see page 1035)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 969)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetCurrentValue (see page 1035)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 972)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1035)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1036)	Position the iteration pointer on the next value.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.

	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartValueIterator (see page 1036)	Reset (see page 3734) the iteration pointer to the first item.
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5339)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
R	Value (see page 5420)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 5639)	Returns the number of values associated with this adapter field.
R	Values (see page 5640)	Returns one of the values associated with the adapter field.
	ValuesList (see page 5420)	Associates a list of values with the adapter field.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.52.2 TCustomAdapterMultiValueField Methods

The methods of the TCustomAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 5637)	Validates values entered by the end-user.
V	GetDisplayStyleType (see page 5637)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5637)	Suggests an HTML input type for this adapter field.
V	GetValueCount (see page 5638)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5638)	Returns one of the values associated with the adapter field.
V	ImplHasValue (see page 5639)	Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.
V	ImplUpdateValue (see page 5639)	Updates the adapter field with value(s) entered by the end-user.

57.1.52.2.1 TCustomAdapterMultiValueField.DoOnValidateValue Method

Validates values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5407)'s DoOnValidateValue method. If the OnValidateValues (see page 5642) event handler is not nil (Delphi) or NULL (C++), it is called it to validate values. If the Required (see page 5339) property is true, DoOnValidateValues also validates that 1 of more values have been entered by the end user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5414)

Required (see page 5339)

OnValidateValues (see page 5642)

57.1.52.2.2 TCustomAdapterMultiValueField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns the display type for an adapter field. Here, it returns htmldList to indicate a list.

See Also

GetDisplayStyleType (see page 5335)

57.1.52.2.3 TCustomAdapterMultiValueField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. If the ValuesList (see page 5420) property is not nil (Delphi) or NULL (C++), GetInputStyleType returns htmliCheckBox indicating that this adapter field should be

edited using a group of check boxes. If the ValuesList (see page 5420) property is nil (Delphi) or NULL (C++), then htmliTextInput is returned. htmliTextInput indicates that the adapter field value should be edited with a single line input.

See Also

GetInputStyleType (see page 5335)

ValuesList (see page 5420)

57.1.52.2.4 TCustomAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the ValueCount (see page 5639) property. If the EchoActionFieldValue (see page 5356) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValueCount (see page 5640) event is called. GetValueCount returns zero if a value can't be found because OnGetValueCount (see page 5640) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

ValueCount (see page 5639)

Values (see page 5640)

EchoActionFieldValue (see page 5356)

57.1.52.2.5 TCustomAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues implements the Values (see page 5640) property. If the EchoActionFieldValue (see page 5356) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValues (see page 5641) event is called. GetValues returns Unassigned if a value can't be found because OnGetValues (see page 5641) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

Values (see page 5640)

OnGetValues (see page 5641)

EchoActionFieldValue (see page 5356)

EchoActionFieldValue (see page 5356)

57.1.52.2.6 TCustomAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues (see page 1034)' HasValue (see page 5483) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue overrides TBaseAdapterMultiValueField (see page 5475)'s ImplHasValue method. If the OnHasValue (see page 5641) event is not nil (Delphi) or NULL (C++), it will be called for the result of ImplHasValue. Otherwise, ImplHasValue will loop through the values of the adapter field, looking for a matching value.

See Also

ImplHasValue (see page 5484)

HasValue (see page 1035)

57.1.52.2.7 TCustomAdapterMultiValueField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 5407)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUpdateValues (see page 5642) event is called.

See Also

OnUpdateValues (see page 5642)

ImplUpdateValue (see page 5418)

57.1.52.3 TCustomAdapterMultiValueField Properties

The properties of the TCustomAdapterMultiValueField class are listed here.

Public Properties

	Name	Description
	ValueCount (see page 5639)	Returns the number of values associated with this adapter field.
	Values (see page 5640)	Returns one of the values associated with the adapter field.

57.1.52.3.1 TCustomAdapterMultiValueField.ValueCount Property

Returns the number of values associated with this adapter field.

Pascal

```
property ValueCount: Integer;
```

C++

```
__property int ValueCount;
```

Description

ValueCount can be used with the Values (see page 5640) property to access all the values associated with this adapter field. ValueCount is implemented by GetValueCount (see page 5638).

See Also

GetValueCount (see page 5638)

Values (see page 5640)

57.1.52.3.2 TCustomAdapterMultiValueField.Values Property

Returns one of the values associated with the adapter field.

Pascal

```
property Values [I: Integer]: Variant;
```

C++

```
__property Variant Values[int I];
```

Description

Values can be used with the ValueCount (see page 5639) property to access all the values associated with this adapter field. Values is implemented by GetValues (see page 5638).

See Also

GetValues (see page 5638)

ValueCount (see page 5639)

57.1.52.4 TCustomAdapterMultiValueField Events

The events of the TCustomAdapterMultiValueField class are listed here.

Public Events

	Name	Description
	OnGetValueCount (see page 5640)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5641)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5641)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5642)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5642)	Validates field values entered by the end-user.

57.1.52.4.1 TCustomAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues (see page 5641) to provide the individual values.

See Also

GetValueCount (see page 5638)

OnGetValues (see page 5641)

57.1.52.4.2 TCustomAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount (see page 5640) to provide the count of values associated with this adapter field.

See Also

GetValues (see page 5638)

OnGetValueCount (see page 5640)

57.1.52.4.3 TCustomAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue (see page 5639) calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue (see page 5639) will search for a matching value the Values (see page 5640) and ValueCount (see page 5639) properties.

See Also

ImplHasValue (see page 5639)

ValueCount (see page 5639)

Values (see page 5640)

57.1.52.4.4 TCustomAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 5642)

ImplUpdateValue (see page 5639)

57.1.52.4.5 TCustomAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

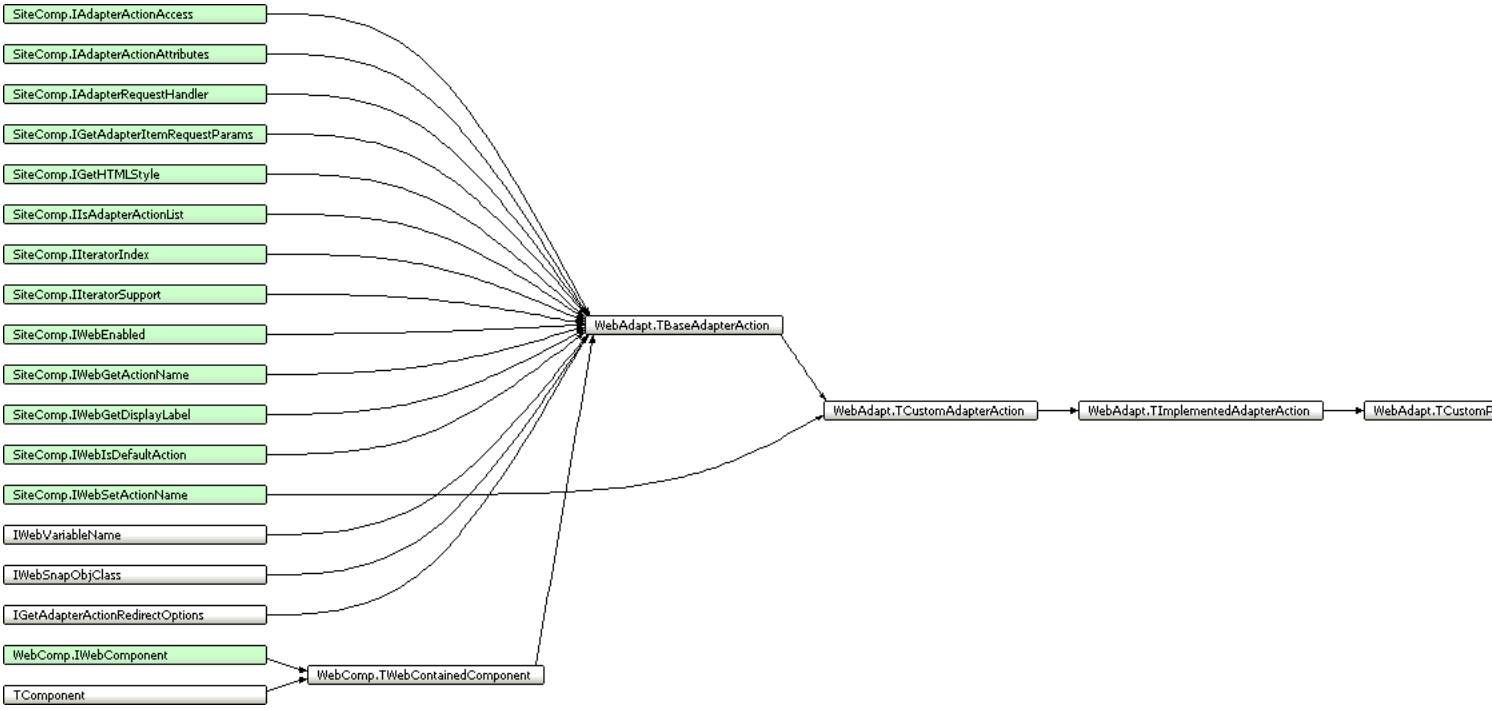
See Also

OnValidateValues

57.1.53 TCustomAdapterPageAction Class

TCustomPagedAdapterAction (see page 5708) is a base class for adapter actions that execute the next or previous page commands.

Class Hierarchy



Pascal

```
TCustomAdapterPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction (see page 5708) provide methods that are used by TAdapterNextPageAction (see page 5358) and TAdapterPrevPageAction (see page 5384).

See Also

TDefaultFieldsPagedAdapter (see page 5733)

TCustomPagedAdapterAction (see page 5708)

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

57.1.53.1 TCustomAdapterPageAction Members

The following tables list the members exposed by TCustomAdapterPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5647)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
V	GetNewPage (see page 5647)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5648)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5648)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.

	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5649)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5649)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5462)	Provides access to the parent adapter component.
	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.

	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.

	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.53.2 TCustomAdapterPageAction Methods

The methods of the TCustomAdapterPageAction class are listed here.

Protected Methods

	Name	Description
	GetDisplayStyleType (see page 5647)	Suggests an HTML display type for this adapter action.
	GetNewPage (see page 5647)	Indicates the page to display when this action is executed.
	ImplExecuteActionRequest (see page 5648)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5648)	Provides the name/value pairs needed to execute this adapter action.
	ImplIsDefaultAction (see page 5649)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 5649)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.53.2.1 TCustomAdapterPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that next page and previous page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 5445)

57.1.53.2.2 TCustomAdapterPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

This is a placeholder. Descendant classes must override this method. true is returned if the action can move to a new page. The AValue parameter identifies the new page. GetNewPage is called by ImplExecuteActionRequest (see page 5648).

See Also

GetNewPage (see page 5363)

GetNewPage (see page 5390)

ImplExecuteActionRequest (see page 5648)

57.1.53.2.3 TCustomAdapterPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 5443). ImplExecuteActionRequest calls ExtractRequestParams followed by GetNewPage (see page 5647). If GetNewPage (see page 5647) returns true then the parent adapter's current page is changed.

See Also

ExecuteActionRequest (see page 5443)

ExtractRequestParams (see page 5740)

GetNewPage (see page 5647)

CurrentPage (see page 5746)

57.1.53.2.4 TCustomAdapterPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5444). ImplGetAdapterItemRequestParams calls the parent adapter's EncodeActionParams method to add the current page as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

ImplGetAdapterItemRequestParams (see page 5453)

GetAdapterItemRequestParams (see page 5444)

EncodeActionParams (see page 5740)

OnGetActionParams (see page 5564)

OnGetParams (see page 5466)

57.1.53.2.5 TCustomAdapterPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5459). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate the a previous or next page command is appropriate.

See Also

IsDefaultAction (see page 5459)

ImplIsDefaultAction (see page 5608)

57.1.53.2.6 TCustomAdapterPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 5461). ImplWebEnabled returns the result of GetNewPage (see page 5647).

See Also

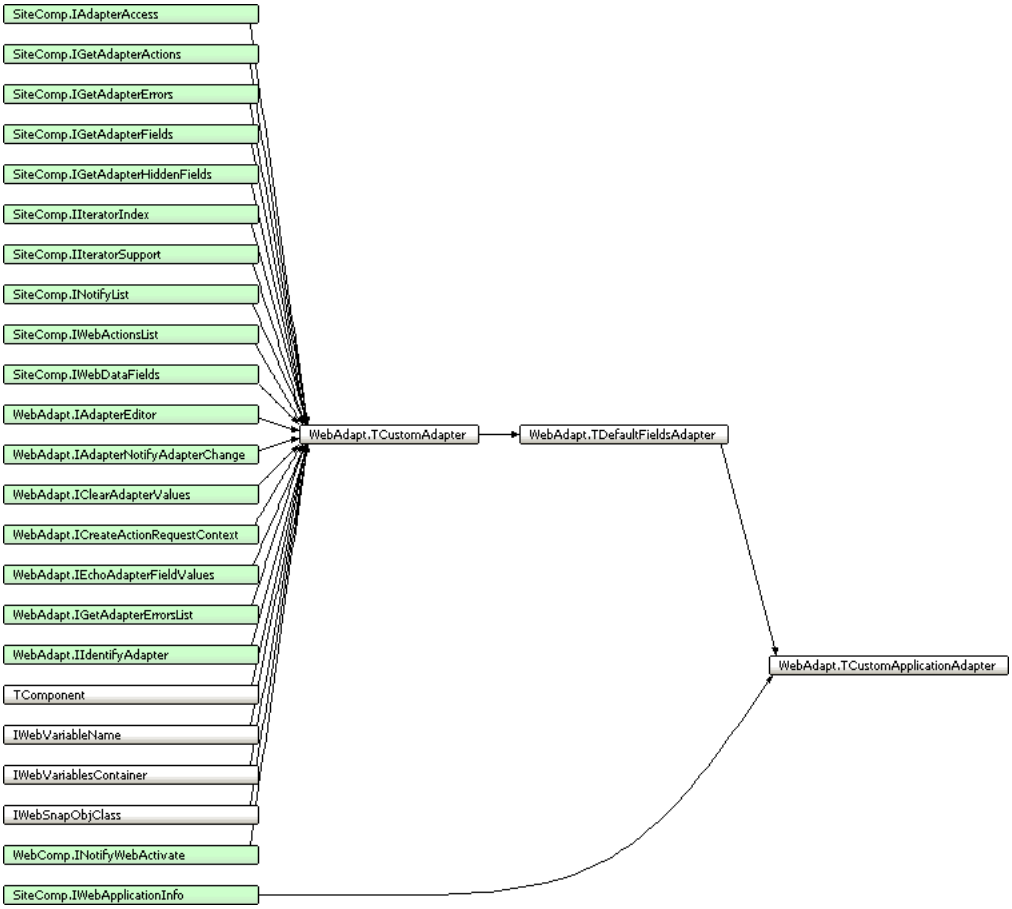
WebEnabled (see page 5461)

GetNewPage (see page 5647)

57.1.54 TCustomApplicationAdapter Class

TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.

Class Hierarchy



Pascal

```
TCustomApplicationAdapter = class(TDefaultFieldsAdapter, IWebApplicationInfo);
```

C++

```
class TCustomApplicationAdapter : public TDefaultFieldsAdapter, public IWebApplicationInfo;
```

File

WebAdapt

Description

TCustomApplicationAdapter implements IWebApplicationInfo (see page 1019). This interface is used to support server-side scripting. TCustomApplicationAdapter automatically adds an adapter field that provides the application title. Additional fields and actions may be added to this adapter. They are accessed as named properties of the server-side script Application object.

57.1.54.1 TCustomApplicationAdapter Members

The following tables list the members exposed by TCustomApplicationAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationTitle method.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.

	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5655)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5655)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5656)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5656)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5656)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
V	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
V	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
V	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
V	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
V	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.

	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.

	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	ApplicationAdapter (see page 1021)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
	ApplicationTitle (see page 5657)	Defines the title of the application.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.54.2 TCustomApplicationAdapter Methods

The methods of the TCustomApplicationAdapter class are listed here.

Protected Methods

	Name	Description
	GetApplicationAdapter (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5655)	Implements IWebApplicationInfo (see page 1019)'s GetApplicationTitle method.
V	GetWebSnapObjClass (see page 5655)	Returns the class type for this adapter field.
V	ImplCanAddActionClass (see page 5655)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5656)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplGetActionsList (see page 5656)	Provides a list of action types that can be added to this adapter.
V	ImplGetFieldsList (see page 5656)	Provides a list of field types that can be added to this adapter.

57.1.54.2.1 TCustomApplicationAdapter.GetApplicationAdapter Method

Implements IWebApplicationInfo (see page 1019)'s GetApplicationAdapter method.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter returns Self.

See Also

GetApplicationAdapter (see page 1021)

57.1.54.2.2 TCustomApplicationAdapter.GetApplicationTitle Method

Implements IWebApplicationInfo (see page 1019)'s GetApplicationTitle method.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall UnicodeString GetApplicationTitle();
```

Description

GetApplicationTitle returns the application title that can be accessed with server-side script as the Application.Title property.

See Also

GetApplicationTitle (see page 1021)

57.1.54.2.3 TCustomApplicationAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

57.1.54.2.4 TCustomApplicationAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows actions classes that derive from TCustomApplicationAdapterAction (see page 5657) in addition to those allows by TCustomAdapter (see page 5519)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 5540)

TCustomApplicationAdapterAction (see page 5657)

57.1.54.2.5 TCustomApplicationAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass allows field classes that derive from TApplicationAdapterField in addition to those allowed by TCustomAdapter (see page 5519)'s ImplCanAddFieldClass method.

See Also

ImplCanAddFieldClass (see page 5540)

57.1.54.2.6 TCustomApplicationAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList clears the list. There are no default actions.

57.1.54.2.7 TCustomApplicationAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetActionsList (see page 5656) lists the title field.

See Also

TAdapterApplicationTitleField (see page 5207)

57.1.54.3 TCustomApplicationAdapter Properties

The properties of the TCustomApplicationAdapter class are listed here.

Public Properties

	Name	Description
	ApplicationTitle (see page 5657)	Defines the title of the application.

57.1.54.3.1 TCustomApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;
```

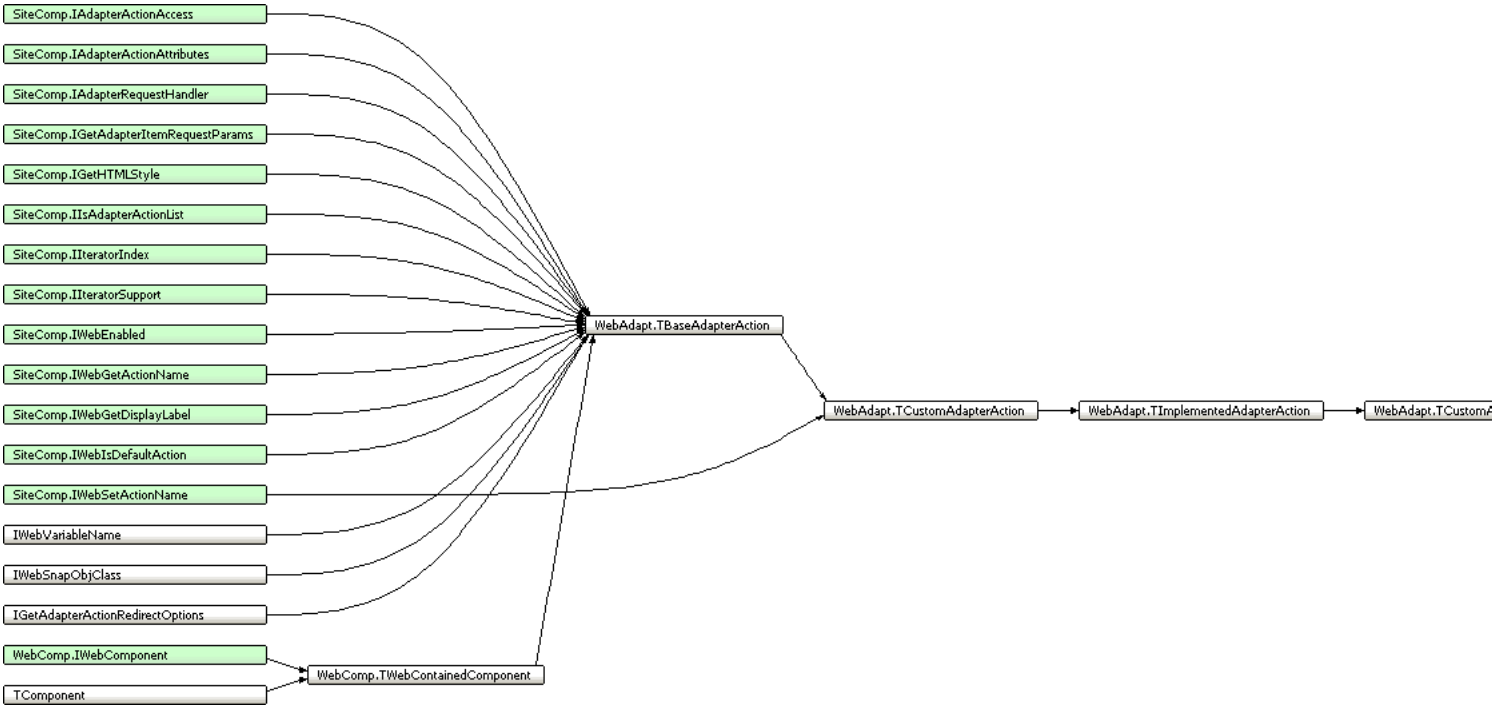
Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

57.1.55 TCustomApplicationAdapterAction Class

TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5649).

Class Hierarchy



Pascal

```
TCustomApplicationAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomApplicationAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomApplicationAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent adapter. Second, it defines a base class that TCustomApplicationAdapter (see page 5649) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5803)

TCustomAdapterAction (see page 5565)

TAdapter (see page 5179)

TApplicationAdapter (see page 5427)

57.1.55.1 TCustomApplicationAdapterAction Members

The following tables list the members exposed by TCustomApplicationAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.

	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5662)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.

	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.

	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.55.2 TCustomApplicationAdapterAction Properties

The properties of the TCustomApplicationAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5662)	Returns the parent adapter of this adapter action.

57.1.55.2.1 TCustomApplicationAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

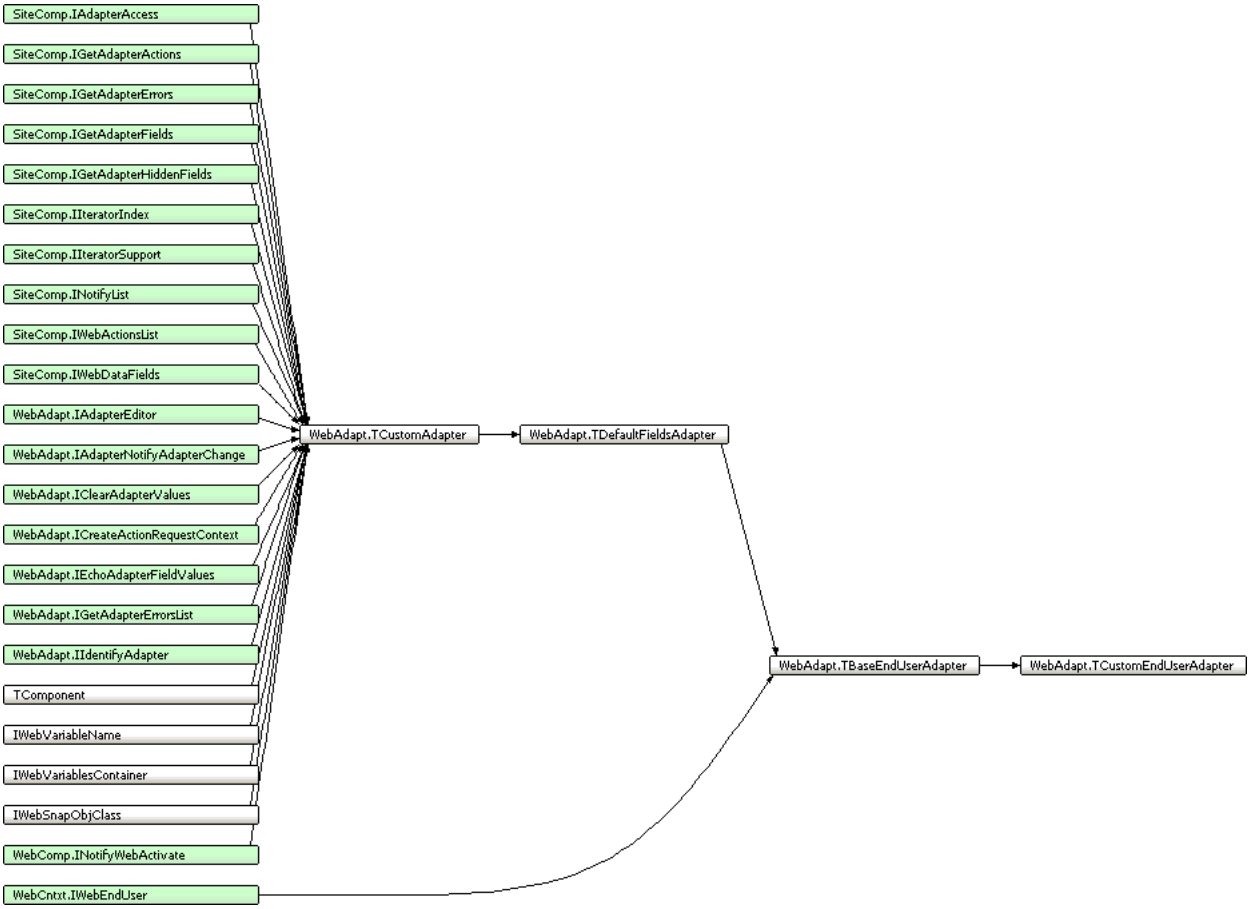
Description

Descendant classes use this property to access the parent adapter.

57.1.56 TCustomEndUserAdapter Class

TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TCustomEndUserAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserAdapter is the base class for end user adapters that use event handlers to implement the default adapter fields and actions. It introduces events to provide values for the default end user fields (DisplayName and LoggedIn) and to implement the default end user actions (logging in and logging out).

As with any end user adapter, TCustomEndUserAdapter has a LoginPage (see page 5503) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

Do not create instances of TCustomEndUserAdapter. Instead, use a descendant such as the TEndUserAdapter (see page 5750) component, which publishes the events whose handlers you must supply to make this adapter work.

See Also

TEndUserAdapter (see page 5750)

57.1.56.1 TCustomEndUserAdapter Members

The following tables list the members exposed by TCustomEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5669)	Generates an OnLogin (see page 5672) event.
V	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
V	ExecuteLogout (see page 5669)	Generates an OnLogout (see page 5673) event.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.

	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5493)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5669)	Generates an OnGetDisplayName (see page 5671) event.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5495)	Determines if the user is logged in
V	GetLoggedInValue (see page 5670)	Generates an OnIsLoggedIn (see page 5672) event.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
V	GetUserID (see page 5670)	Generates an OnGetUserID (see page 5671) event.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
V	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
V	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).
V	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).

	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5501)	Processes the login for the given user.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5502)	Logs in a specified user.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5502)	Generates the login page.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5503)	Gets the user list.

	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5671)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 5671)	Occurs when the adapter needs to provide the ID of the current end user.
	OnHasRights (see page 5503)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5672)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 5672)	Logs in the current end user.
	OnLogout (see page 5673)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5504)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.

	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5967)	Logs in a specified user.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
	LoginPage (see page 5503)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.56.2 TCustomEndUserAdapter Methods

The methods of the TCustomEndUserAdapter class are listed here.

Protected Methods

	Name	Description
	ExecuteLogin (see page 5669)	Generates an OnLogin (see page 5672) event.
	ExecuteLogout (see page 5669)	Generates an OnLogout (see page 5673) event.
	GetDisplayNameValue (see page 5669)	Generates an OnGetDisplayName (see page 5671) event.
	GetLoggedInValue (see page 5670)	Generates an OnIsLoggedIn (see page 5672) event.
	GetUserID (see page 5670)	Generates an OnGetUserID (see page 5671) event.

57.1.56.2.1 TCustomEndUserAdapter.ExecuteLogin Method

Generates an OnLogin (see page 5672) event.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the Login action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process. TCustomEndUserAdapter (see page 5662) responds by generating an OnLogin (see page 5672) event, where you can take whatever action is necessary to log in the specified user.

AUserID is the user ID of the user to log in.

See Also

OnLogin (see page 5672)

57.1.56.2.2 TCustomEndUserAdapter.ExecuteLogout Method

Generates an OnLogout (see page 5673) event.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout. TCustomEndUserAdapter (see page 5662) responds by generating an OnLogout (see page 5673) event, where you can take whatever action is necessary to log the specified user out.

See Also

OnLogout (see page 5673)

57.1.56.2.3 TCustomEndUserAdapter.GetDisplayNameValue Method

Generates an OnGetDisplayName (see page 5671) event.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue. TCustomEndUserAdapter (see page 5662) responds by generating an OnGetDisplayName (see page 5671) event, where you can take supply the name of the current user.

GetDisplayNameValue returns the display name that the OnGetDisplayName (see page 5671) event handler provides. If you do not supply an OnGetDisplayName (see page 5671) event handler, GetDisplayNameValue returns an empty string.

See Also

OnGetDisplayName (see page 5671)

57.1.56.2.4 TCustomEndUserAdapter.GetLoggedInValue Method

Generates an OnIsLoggedIn (see page 5672) event.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue. TCustomEndUserAdapter (see page 5662) responds by generating an OnIsLoggedIn (see page 5672) event, where you can indicate whether the current user has logged in.

GetLoggedInValue returns the value that the OnGetLoggedIn event handler provides. If you do not supply an OnGetLoggedIn event handler, GetLoggedInValue returns false.

See Also

OnIsLoggedIn (see page 5672)

57.1.56.2.5 TCustomEndUserAdapter.GetUserID Method

Generates an OnGetUserID (see page 5671) event.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue. TCustomEndUserAdapter (see page 5662) responds by generating an OnGetUserID (see page 5671) event, where you can take supply the ID of the current user.

GetUserIDValue returns the user ID that the OnGetUserID (see page 5671) event handler provides. If you do not supply an OnGetUserID (see page 5671) event handler, GetUserIDValue returns an unassigned Variant (see page 3430).

See Also

OnGetUserID (see page 5671)

57.1.56.3 TCustomEndUserAdapter Events

The events of the TCustomEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnGetDisplayName (see page 5671)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 5671)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIsLoggedIn (see page 5672)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5672)	Logs in the current end user.
	OnLogout (see page 5673)	Occurs when the current user logs out.

57.1.56.3.1 TCustomEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write (see page 3789) an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser (see page 5962) interface) requests the display name for the current end user, TCustomEndUserAdapter (see page 5662) generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter (see page 5662) always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 5968)

EndUser (see page 5947)

WebContext (see page 5977)

57.1.56.3.2 TCustomEndUserAdapter.OnGetUserID Event

Occurs when the adapter needs to provide the ID of the current end user.

Pascal

```
property OnGetUserID: TEndUserGetUserIDEvent;
```

C++

```
__property TEndUserGetUserIDEvent OnGetUserID;
```

Description

Write (see page 3789) an OnGetDisplayName (see page 5671) event handler to provide the user ID for the current user.

When a server-side script requests ID for the current end user, TCustomEndUserAdapter (see page 5662) generates an OnGetUserID event to retrieve that value.

Note: If you do not supply an OnGetUserID event handler, TCustomEndUserAdapter (see page 5662) always returns an unassigned Variant (see page 3430) as the user ID.

See Also

GetDisplayNameValue (see page 5669)

TWebUserList (see page 6759)

OnLogin (see page 5672)

57.1.56.3.3 TCustomEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write (see page 3789) an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser (see page 5962) interface) inquires whether the current user has logged in, TCustomEndUserAdapter (see page 5662) generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter (see page 5662) always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 5968)

EndUser (see page 5947)

WebContext (see page 5977)

OnLogin (see page 5672)

OnLogout (see page 5673)

57.1.56.3.4 TCustomEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write (see page 3789) an OnLogin event handler to allow the TCustomEndUserAdapter (see page 5662) object to log users in. When a server-side script (or a caller of the IWebEndUser (see page 5962) interface) attempts to log in a user, TCustomEndUserAdapter (see page 5662) generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter (see page 5662) does nothing to log users in.

See Also

LogIn (see page 5967)

EndUser (see page 5947)

WebContext (see page 5977)

OnIsLoggedIn (see page 5672)

OnLogout (see page 5673)

TWebUserList (see page 6759)

OnGetUserID (see page 5671)

57.1.56.3.5 TCustomEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write (see page 3789) an OnLogout event handler to allow the TCustomEndUserAdapter (see page 5662) object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter (see page 5662) generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin (see page 5672) event handler, TCustomEndUserAdapter (see page 5662) does nothing to log users out.

See Also

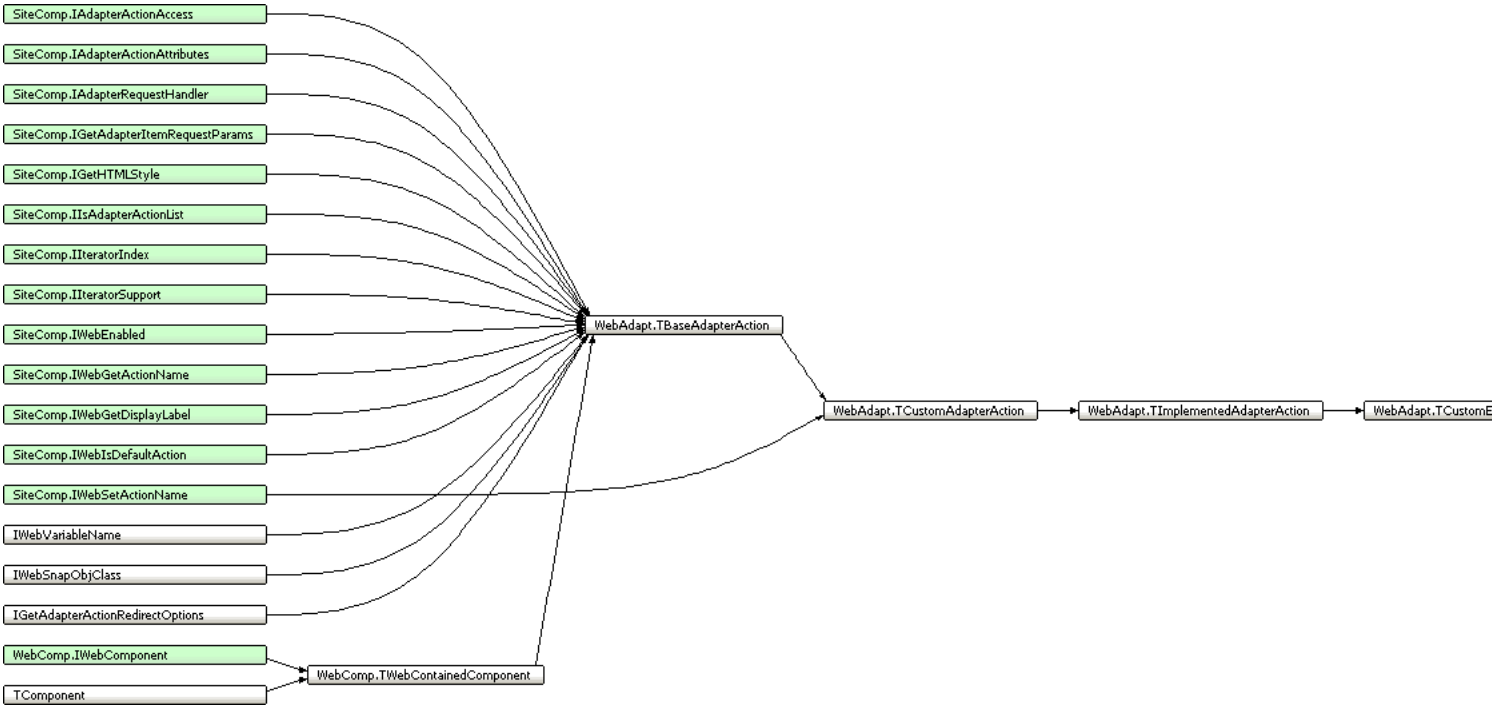
OnIsLoggedIn (see page 5672)

OnLogin (see page 5672)

57.1.57 TCustomEndUserAdapterAction Class

TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5662).

Class Hierarchy



Pascal

```
TCustomEndUserAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomEndUserAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomEndUserAdapterAction has two purposes. First, it provides a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomEndUserAdapter (see page 5662) checks for when determining whether an adapter action can be added as a child.

57.1.57.1 TCustomEndUserAdapterAction Members

The following tables list the members exposed by TCustomEndUserAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.

	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.

	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5678)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).

	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.57.2 TCustomEndUserAdapterAction Properties

The properties of the TCustomEndUserAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5678)	Returns the parent adapter of this adapter action.

57.1.57.2.1 TCustomEndUserAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

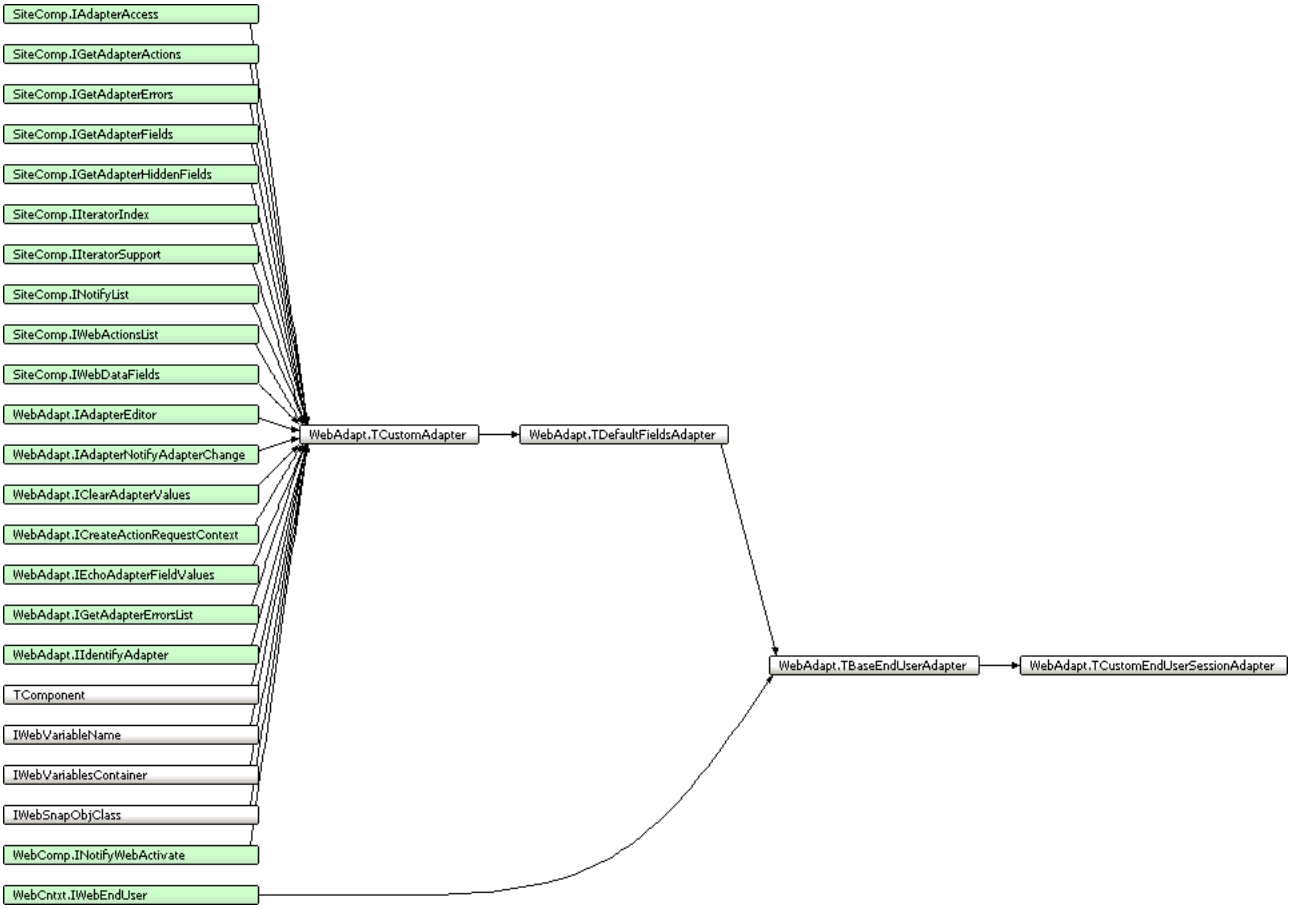
See Also

TCustomEndUserAdapter (see page 5662)

57.1.58 TCustomEndUserSessionAdapter Class

TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TCustomEndUserSessionAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserSessionAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserSessionAdapter is the base class for end user adapters that use a TSessionsService object to implement the default adapter fields and actions. It handles login and logout actions on behalf of the end user, and uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TCustomEndUserSessionAdapter to work properly.

As with any end user adapter, TCustomEndUserSessionAdapter has a LoginPage (see page 5503) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

See Also

TEndUserSessionAdapter (see page 5779)

TSessionsService (see page 6722)

57.1.58.1 TCustomEndUserSessionAdapter Members

The following tables list the members exposed by TCustomEndUserSessionAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5685)	Logs in an end user.
V	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
V	ExecuteLogout (see page 5685)	Logs out the current user.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5493)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5685)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5495)	Determines if the user is logged in
V	GetLoggedInValue (see page 5686)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
V	GetUserID (see page 5686)	Returns the user ID of the current user.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
V	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
V	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).

	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5501)	Processes the login for the given user.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5502)	Logs in a specified user.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5502)	Generates the login page.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.

	UserList (see page 5503)	Gets the user list.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5503)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 5504)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.

	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5967)	Logs in a specified user.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
	LoginPage (see page 5503)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	UserID (see page 5687)	Specifies the user ID of the current end user.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.58.2 TCustomEndUserSessionAdapter Methods

The methods of the TCustomEndUserSessionAdapter class are listed here.

Protected Methods

	Name	Description
V	ExecuteLogin (see page 5685)	Logs in an end user.

	ExecuteLogout (see page 5685)	Logs out the current user.
	GetDisplayNameValue (see page 5685)	Returns the name of the current end user.
	GetLoggedInValue (see page 5686)	Indicates whether the current end user has logged in.
	GetUserID (see page 5686)	Returns the user ID of the current user.

57.1.58.2.1 TCustomEndUserSessionAdapter.ExecuteLogin Method

Logs in an end user.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the Login action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process.

In TCustomEndUserAdapter (see page 5662), ExecuteLogin assigns the supplied user ID to the UserID (see page 5687) property, which saves the current user ID with the current session.

AUserID is the user ID of the user to log in.

57.1.58.2.2 TCustomEndUserSessionAdapter.ExecuteLogout Method

Logs out the current user.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout.

In TCustomEndUserAdapter (see page 5662), ExecuteLogout terminates the current Web session.

57.1.58.2.3 TCustomEndUserSessionAdapter.GetDisplayNameValue Method

Returns the name of the current end user.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue.

In TCustomEndUserSessionAdapter (see page 5678), GetDisplayNameValue returns the value of the UserID (see page

5687) property, formatted as a string.

57.1.58.2.4 TCustomEndUserSessionAdapter.GetLoggedInValue Method

Indicates whether the current end user has logged in.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue.

In TCustomEndUserAdapter (see page 5662), GetLoggedInValue checks whether the UserID (see page 5687) property has been assigned. If UserID (see page 5687) is an assigned Variant (see page 3430) other than an empty string, GetLoggedInValue returns true, indicating that the user has logged in. Otherwise, GetLoggedInValue returns false.

57.1.58.2.5 TCustomEndUserSessionAdapter.GetUserID Method

Returns the user ID of the current user.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID (see page 5687) adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue.

In TCustomEndUserSessionAdapter (see page 5678), GetUserID fetches the value of the UserID (see page 5687) property from the current Web session. This value is set when the user first logs in.

See Also

GetUserID (see page 5670)

ExecuteLogin (see page 5669)

TWebSession (see page 6225)

57.1.58.3 TCustomEndUserSessionAdapter Properties

The properties of the TCustomEndUserSessionAdapter class are listed here.

Public Properties

	Name	Description
	UserID (see page 5687)	Specifies the user ID of the current end user.

57.1.58.3.1 TCustomEndUserSessionAdapter.UserID Property

Specifies the user ID of the current end user.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

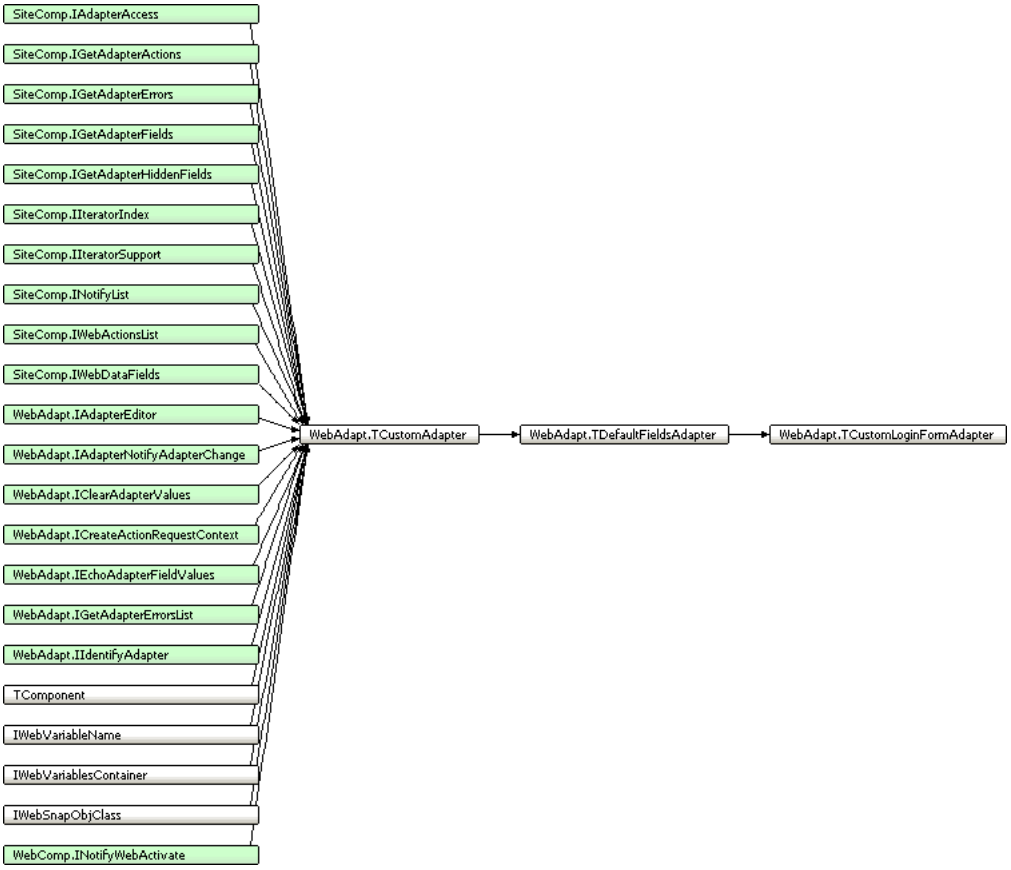
Description

Use UserID to get or set the user ID of the current user. When a user logs in (that is, when TCustomEndUserSessionAdapter (see page 5678) executes the Login action), TCustomEndUserSessionAdapter (see page 5678) assigns the user's ID as the value of UserID. TCustomEndUserSessionAdapter (see page 5678) stores this value with the current session.

57.1.59 TCustomLoginFormAdapter Class

TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.

Class Hierarchy



Pascal

```
TCustomLoginFormAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TCustomLoginFormAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

TCustomLoginFormAdapter introduces default fields and actions that support building a login form. The adapter fields include user name (UserName (see page 5700)), password (Password (see page 5700)), and next page (NextPage (see page 5699)). The next page field lists the Web pages in the application. The properties of TCustomLoginFormAdapter reflect the values that are assigned to those fields. TCustomLoginFormAdapter implements a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 5750) or TEndUserSessionAdapter (see page 5779), TLoginFormAdapter (see page 5810) calls the end user adapter's Login method to log the user in. As an alternate approach, you can provide an OnLogin (see page 5701) event handler.

Do not create instances of TCustomLoginFormAdapter. It is intended only as a base class. Instead, use a TCustomLoginFormAdapter descendant such as TLoginFormAdapter (see page 5810).

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

TAdapterPageProducer (see page 1076)

TLoginFormAdapter (see page 5810)

57.1.59.1 TCustomLoginFormAdapter Members

The following tables list the members exposed by TCustomLoginFormAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.

	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 5693)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5693)	Logs in the user after the user name and password have been validated.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextPage (see page 5694)	Returns the value of the NextPage (see page 5699) property.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5694)	Provides the underlying implementation for the CanAddActionClass (see page 5527) method.
	ImplCanAddFieldClass (see page 5695)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5695)	Provides the underlying implementation for the GetActionsList (see page 5533) method.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5696)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.

	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5696)	Provides the underlying implementation for the NotifyDeactivate (see page 5552) method.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	RaiseBlankPassword (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied user name is blank.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5697)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5698)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5698)	Validates the name that the user supplied to the login form.

	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 5701)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5702)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5702)	Occurs when the login form adapter must validate the name that the user supplied.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.

	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	NextPage (see page 5699)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5700)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 5700)	Indicates whether the user must supply a password to log in.
R	UserName (see page 5700)	Indicates the name that the user supplied to the login form.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.59.2 TCustomLoginFormAdapter Methods

The methods of the TCustomLoginFormAdapter class are listed here.

Protected Methods

	Name	Description
	EndUser (see page 5693)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5693)	Logs in the user after the user name and password have been validated.
V	GetNextPage (see page 5694)	Returns the value of the NextPage (see page 5699) property.
V	ImplCanAddActionClass (see page 5694)	Provides the underlying implementation for the CanAddActionClass (see page 5527) method.
V	ImplCanAddFieldClass (see page 5695)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.
V	ImplGetActionsList (see page 5695)	Provides the underlying implementation for the GetActionsList (see page 5533) method.
V	ImplGetFieldsList (see page 5696)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.
V	ImplNotifyDeactivate (see page 5696)	Provides the underlying implementation for the NotifyDeactivate (see page 5552) method.
	RaiseBlankPassword (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied user name is blank.

	UserList (see page 5697)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5698)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5698)	Validates the name that the user supplied to the login form.

57.1.59.2.1 TCustomLoginFormAdapter.EndUser Method

Returns the interface of the WebSnap application's end user adapter.

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

TCustomLoginFormAdapter (see page 5687) uses EndUser internally to interact with the WebSnap application's end user adapter. For example, the login form adapter calls the Login method on this interface to log in an end user after validating the user name and password.

EndUser returns the interface of an end user adapter such as TEndUserAdapter (see page 5750) or TEndUserSessionAdapter (see page 5779). If the WebSnap application does not include an end user adapter, EndUser returns nil (Delphi) or NULL (C++).

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

LogIn (see page 5967)

57.1.59.2.2 TCustomLoginFormAdapter.ExecuteLogin Method

Logs in the user after the user name and password have been validated.

Pascal

```
procedure ExecuteLogin(AUserID: Variant);
```

C++

```
__fastcall ExecuteLogin(Variant AUserID);
```

Description

The Login action calls ExecuteLogin to log the user in after validating the user name and password. This method first generates an OnLogin (see page 5701) event, so that the OnLogin (see page 5701) event handler can perform any application-specific tasks to log the user in. Then, if the EndUser (see page 5693) property is not nil (Delphi) or NULL (C++), ExecuteLogin calls the end user adapter's Login method, so that the end user adapter can log the user in.

AUserID is the user ID of the end user. If the WebSnap application includes a TWebUserList component, this is the user ID that the Web user list associates with the current user name and password. Otherwise, AUserID is the value of the UserName (see page 5700) property.

See Also

EndUser (see page 5693)

OnLogin (see page 5701)

LogIn (see page 5967)

UserName (see page 5700)

TWebUserList (see page 6759)

57.1.59.2.3 TCustomLoginFormAdapter.GetNextPage Method

Returns the value of the NextPage (see page 5699) property.

Pascal

```
function GetNextPage: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetNextPage();
```

Description

GetNextPage is the getter method for the NextPage (see page 5699) property. See the NextPage (see page 5699) property for more information.

See Also

NextPage (see page 5699)

57.1.59.2.4 TCustomLoginFormAdapter.ImplCanAddActionClass Method

Provides the underlying implementation for the CanAddActionClass (see page 5527) method.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass provides the underlying implementation of the CanAddActionClass (see page 5527) method that is introduced in TCustomAdapter (see page 5519). At design time, the Actions (see page 5732) editor calls this method to determine what actions it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added action.

AClass is the class of the proposed action.

ImplCanAddActionClass returns true if the login form adapter can act as a parent to an action of the specified class.

In TCustomLoginFormAdapter (see page 5687), ImplCanAddActionClass returns true if AClass is (or descends from) TLoginFormAdapterLoginAction (see page 5821) (the Login action).

See Also

ImplCanAddFieldClass (see page 5695)

CanAddActionClass (see page 5527)

TLoginFormAdapterLoginAction (see page 5821)

ImplGetActionsList (see page 5695)

57.1.59.2.5 TCustomLoginFormAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass provides the underlying implementation of the CanAddFieldClass (see page 5527) method that is introduced in TCustomAdapter (see page 5519). At design time, the Fields editor calls this method to determine what fields it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added field.

AClass is the class of the proposed field.

ImplCanAddFieldClass returns true if the login form adapter can act as a parent to a field of the specified class.

In TCustomLoginFormAdapter (see page 5687), ImplCanAddFieldClass returns true if AClass is (or descends from) TAdapterPasswordField (see page 5379), TAdapterUserNameField (see page 5422) or TAdapterNextPageField (see page 5367) (the Password (see page 5700), UserName (see page 5700), or NextPage (see page 5699) field).

See Also

ImplCanAddActionClass (see page 5694)

CanAddFieldClass (see page 5527)

TAdapterNextPageField (see page 5367)

TAdapterPasswordField (see page 5379)

TAdapterUserNameField (see page 5422)

ImplGetFieldsList (see page 5696)

57.1.59.2.6 TCustomLoginFormAdapter.ImplGetActionsList Method

Provides the underlying implementation for the GetActionsList (see page 5533) method.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList provides the underlying implementation of the GetActionsList (see page 5533) method that is introduced in TCustomAdapter (see page 5519). This method fills a string list with the names of all the actions an adapter can support, associating each action name with a class reference to the corresponding action class.

AList is an existing string list instance to fill with the names of the actions that this adapter supports.

In TCustomLoginFormAdapter (see page 5687), ImplGetActionsList clears the list of any strings it may contain on entry, and adds a single string, 'Login', which is the name of the login action. It associates this string with TLoginFormAdapterLoginAction (see page 5821).

See Also

ImplGetFieldsList (see page 5696)

GetActionsList (see page 5533)

TLoginFormAdapterLoginAction (see page 5821)

57.1.59.2.7 TCustomLoginFormAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 5536) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList provides the underlying implementation of the GetFieldsList (see page 5536) method that is introduced in TCustomAdapter (see page 5519). This method fills a string list with the names of all the fields an adapter can support, associating each field name with a class reference to the corresponding field class.

AList is an existing string list instance to fill with the names of the fields that this adapter supports.

In TCustomLoginFormAdapter (see page 5687), ImplGetFieldsList clears the list of any strings it may contain on entry, and adds the strings, 'UserName (see page 5700)', 'Password (see page 5700)', and 'NextPage (see page 5699)', which it associates with TAdapterUserNameField (see page 5422), TAdapterPasswordField (see page 5379), and TAdapterNextPageField (see page 5367), respectively.

See Also

ImplGetActionsList (see page 5695)

GetFieldsList (see page 5536)

TAdapterNextPageField (see page 5367)

TAdapterPasswordField (see page 5379)

TAdapterUserNameField (see page 5422)

57.1.59.2.8 TCustomLoginFormAdapter.ImplNotifyDeactivate Method

Provides the underlying implementation for the NotifyDeactivate (see page 5552) method.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

ImplNotifyDeactivate provides the underlying implementation of the NotifyDeactivate (see page 5552) method that is introduced in TCustomAdapter (see page 5519). This method is called when the WebSnap application deactivates the Web module that contains the adapter after processing an HTTP request message.

In TCustomLoginFormAdapter (see page 5687), ImplNotifyDeactivate clears the NextPage (see page 5699) property so that

the next time a request message arrives from the login form, it can be redirected to an appropriate next page.

See Also

NotifyDeactivate (see page 5552)

NextPage (see page 5699)

57.1.59.2.9 TCustomLoginFormAdapter.RaiseBlankPassword Method

Raises an EAdapterFieldException (see page 5167) with a message that the supplied password is blank.

Pascal

```
procedure RaiseBlankPassword;
```

C++

```
__fastcall RaiseBlankPassword();
```

Description

TCustomLoginFormAdapter (see page 5687) calls RaiseBlankPassword internally when it is trying to validate a password that is an empty string and the PasswordRequired (see page 5700) property is true.

See Also

PasswordRequired (see page 5700)

EAdapterFieldException (see page 5167)

ValidatePassword (see page 5698)

57.1.59.2.10 TCustomLoginFormAdapter.RaiseBlankUserName Method

Raises an EAdapterFieldException (see page 5167) with a message that the supplied user name is blank.

Pascal

```
procedure RaiseBlankUserName;
```

C++

```
__fastcall RaiseBlankUserName();
```

Description

TCustomLoginFormAdapter (see page 5687) calls RaiseBlankUserName internally when it is trying to validate a user name that is an empty string.

See Also

EAdapterFieldException (see page 5167)

ValidateUserName (see page 5698)

57.1.59.2.11 TCustomLoginFormAdapter.UserList Method

Returns the interface of the WebSnap application's Web user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

TCustomLoginFormAdapter (see page 5687) uses UserList internally to interact with the WebSnap application's Web user list. For example, the login form adapter calls the ValidateUser method on this interface when validating the user name and password.

UserList returns the interface of a Web user list such as TWebUserList. If the WebSnap application does not include a Web user list, UserList returns nil (Delphi) or NULL (C++).

See Also

TWebUserList (see page 6759)

ValidateUser (see page 1059)

57.1.59.2.12 TCustomLoginFormAdapter.ValidatePassword Method

Validates the password that the user supplied to the login form.

Pascal

```
procedure ValidatePassword;
```

C++

```
__fastcall ValidatePassword();
```

Description

The Login action associated with this adapter calls ValidatePassword to validate the password that the user supplies. ValidatePassword first generates an OnValidatePassword (see page 5702) event. If the event handler does not exist, or does not handle the validation, ValidatePassword checks whether the password is an empty string. If so, and if the PasswordRequired (see page 5700) property is true, ValidatePassword raises an EAdapterFieldException (see page 5167) exception.

See Also

OnValidatePassword (see page 5702)

PasswordRequired (see page 5700)

EAdapterFieldException (see page 5167)

Password (see page 5700)

57.1.59.2.13 TCustomLoginFormAdapter.ValidateUserName Method

Validates the name that the user supplied to the login form.

Pascal

```
procedure ValidateUserName;
```

C++

```
__fastcall ValidateUserName();
```

Description

The Login action associated with this adapter calls ValidateUserName to validate the name that the user supplies. ValidateUserName first generates an OnValidateUserName (see page 5702) event. If the event handler does not exist, or does not handle the validation, ValidateUserName checks whether the user name is an empty string. If so, it raises an

EAdapterFieldException (see page 5167) exception.

See Also

OnValidateUserName (see page 5702)

EAdapterFieldException (see page 5167)

UserName (see page 5700)

57.1.59.3 TCustomLoginFormAdapter Properties

The properties of the TCustomLoginFormAdapter class are listed here.

Public Properties

	Name	Description
	NextPage (see page 5699)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5700)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 5700)	Indicates whether the user must supply a password to log in.
R	UserName (see page 5700)	Indicates the name that the user supplied to the login form.

57.1.59.3.1 TCustomLoginFormAdapter.NextPage Property

Names the Web page the user should see after successfully logging in.

Pascal

```
property NextPage: string;
```

C++

```
__property UnicodeString NextPage;
```

Description

NextPage specifies the page that the WebSnap application presents to end users after they successfully log in.

The NextPage adapter field supplies a value for the NextPage property. This is a page that the user selects using the login form. If NextPage is unassigned (that is, the if the login form does not contain a NextPage field or the user does not select a next page), TCustomLoginFormAdapter (see page 5687) checks whether the request parameters include a "success page", and if so, uses that. If there is no "success page" parameter, TCustomLoginFormAdapter (see page 5687) checks the Web context to see if there is a currently requested page the user was trying to reach before being diverted to the login page. If there is no "success page" parameter and no currently requested page, TCustomLoginFormAdapter (see page 5687) uses the application's default page.

See Also

DefaultPage (see page 6169)

RequestedPageName (see page 5949)

TAdapterPageProducer (see page 1076)

TLoginFormAdapter (see page 5810)

SuccessPage (see page 951)

GetNextPage (see page 5694)

57.1.59.3.2 TCustomLoginFormAdapter.Password Property

Indicates the password that the user supplied to the login form.

Pascal

```
property Password: Variant;
```

C++

```
__property Variant Password;
```

Description

Read (see page 3727) Password to obtain the password that the user supplied to the login form. The login form adapter reads this value from the current request message.

See Also

PasswordRequired (see page 5700)

OnValidatePassword (see page 5702)

57.1.59.3.3 TCustomLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword (see page 5702) event handler does not handle the validation of the password, then the login form adapter checks that the value of Password (see page 5700) is not an empty string. If Password (see page 5700) is an empty string, the login form adapter raises an EAdapterFieldException (see page 5167) exception when it tries to validate the password. If the OnValidatePassword (see page 5702) event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's Login action.

See Also

Password (see page 5700)

OnValidatePassword (see page 5702)

TLoginFormAdapterLoginAction (see page 5821)

EAdapterFieldException (see page 5167)

57.1.59.3.4 TCustomLoginFormAdapter.UserName Property

Indicates the name that the user supplied to the login form.

Pascal

```
property UserName: Variant;
```

C++

```
__property Variant UserName;
```

Description

Read (see page 3727) UserName to obtain the name that the user supplied when logging in. The login form adapter reads this value from the current request message.

See Also

Password (see page 5700)

OnValidateUserName (see page 5702)

57.1.59.4 TCustomLoginFormAdapter Events

The events of the TCustomLoginFormAdapter class are listed here.

Public Events

	Name	Description
	OnLogin (see page 5701)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5702)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5702)	Occurs when the login form adapter must validate the name that the user supplied.

57.1.59.4.1 TCustomLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write (see page 3789) an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword (see page 5702) and an OnValidateUserName (see page 5702) event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's Login method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 5702)

OnValidateUserName (see page 5702)

LogIn (see page 5967)

TWebUserList (see page 6759)

UserName (see page 5700)

57.1.59.4.2 TCustomLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write (see page 3789) an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password (see page 5700) is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName (see page 5700) property.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName (see page 5702) event, and then to log the user in using an OnLogin (see page 5701) event (and, if the application includes an end user adapter, by calling the end user adapter's Login method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired (see page 5700) property is true.

See Also

OnLogin (see page 5701)

OnValidateUserName (see page 5702)

Password (see page 5700)

UserName (see page 5700)

PasswordRequired (see page 5700)

57.1.59.4.3 TCustomLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write (see page 3789) an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword (see page 5702) event and before the OnLogin (see page 5701) event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName (see page 5700) property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException (see page 5167) exception if the value of UserName (see page 5700) is an empty string. If the OnValidateUserName event handler finds the user name to be invalid, it should raise an exception.

See Also

OnLogin (see page 5701)

OnValidatePassword (see page 5702)

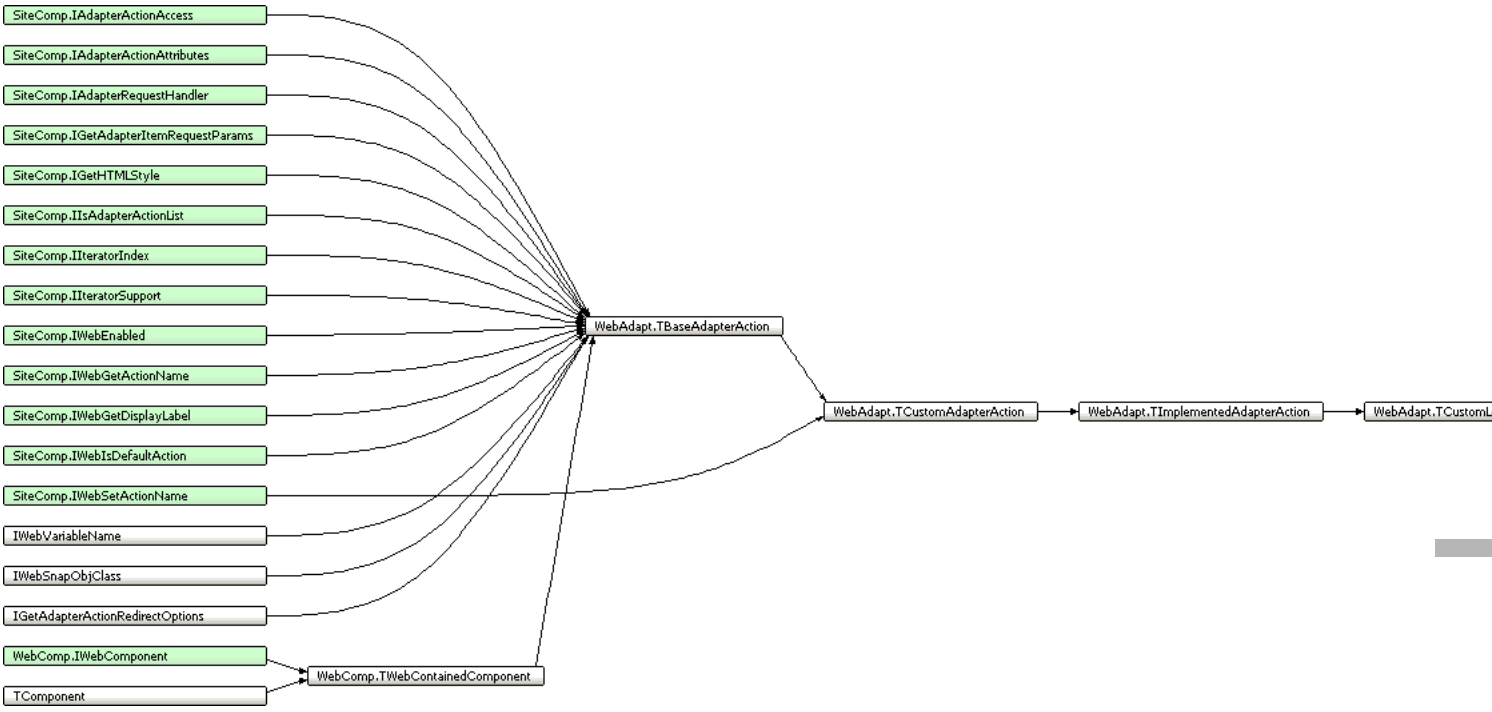
UserName (see page 5700)

EAdapterFieldException (see page 5167)

57.1.60 TCustomLoginFormAdapterAction Class

TCustomEndUserAdapterAction (see page 5673) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5687).

Class Hierarchy



Pascal

```
TCustomLoginFormAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomLoginFormAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomLoginFormAdapterAction has two purposes. First, it implements a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomLoginFormAdapter (see page 5687) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5803)

TCustomLoginFormAdapter (see page 5687)

57.1.60.1 TCustomLoginFormAdapterAction Members

The following tables list the members exposed by TCustomLoginFormAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.

	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5707)	Returns the adapter that acts as a parent to this action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.

W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.60.2 TCustomLoginFormAdapterAction Properties

The properties of the TCustomLoginFormAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5707)	Returns the adapter that acts as a parent to this action.

57.1.60.2.1 TCustomLoginFormAdapterAction.Adapter Property

Returns the adapter that acts as a parent to this action.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

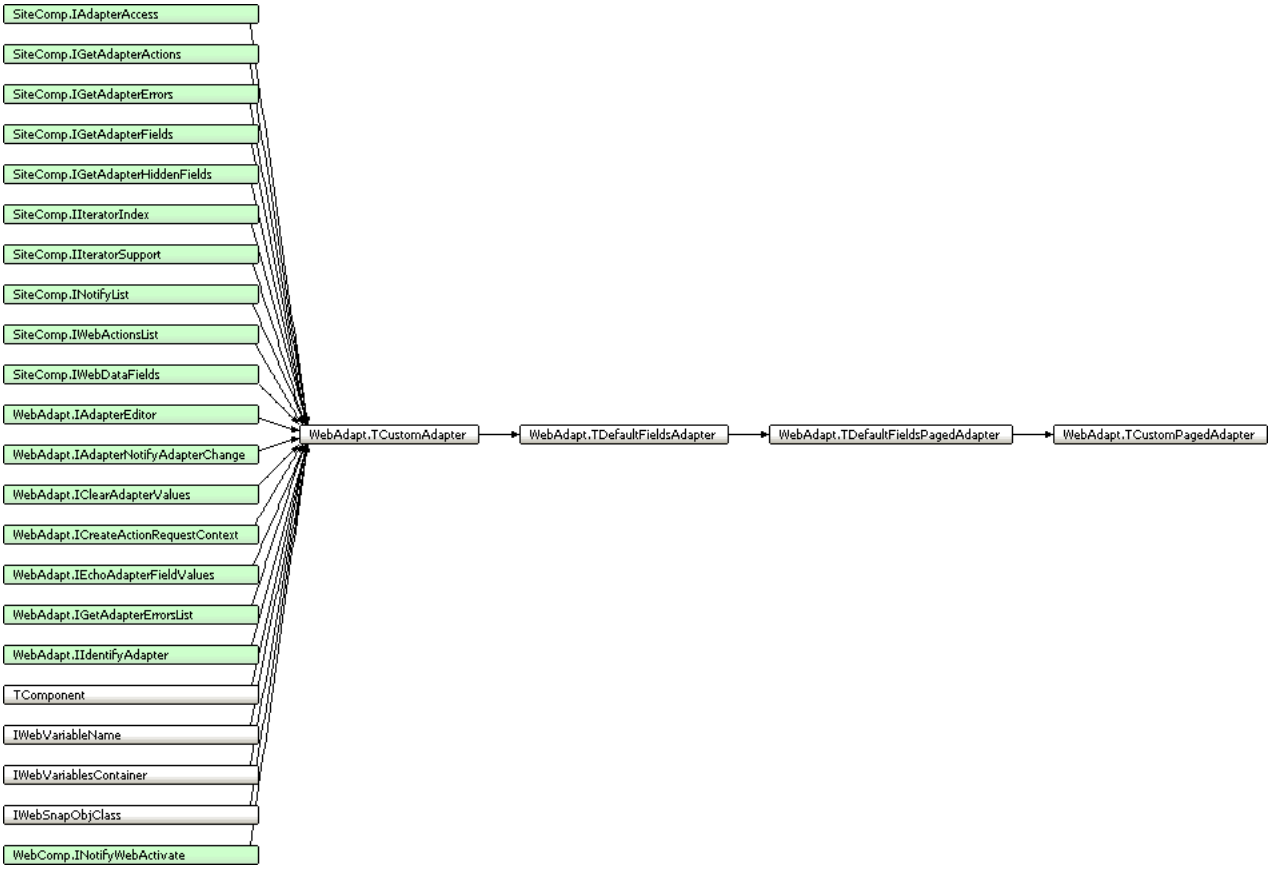
Description

Descendant classes use this property to access the parent adapter.

57.1.61 TCustomPagedAdapter Class

TCustomPagedAdapter is a base class for adapters that support paging.

Class Hierarchy



Pascal

```
TCustomPagedAdapter = class(TDefaultFieldsPagedAdapter);
```

C++

```
class TCustomPagedAdapter : public TDefaultFieldsPagedAdapter;
```

File

WebAdapt

Description

Derive from TCustomPagedAdapter to define a new adapter class that support paging. Paging is enabled when the PageSize (see page 5747) property is set.

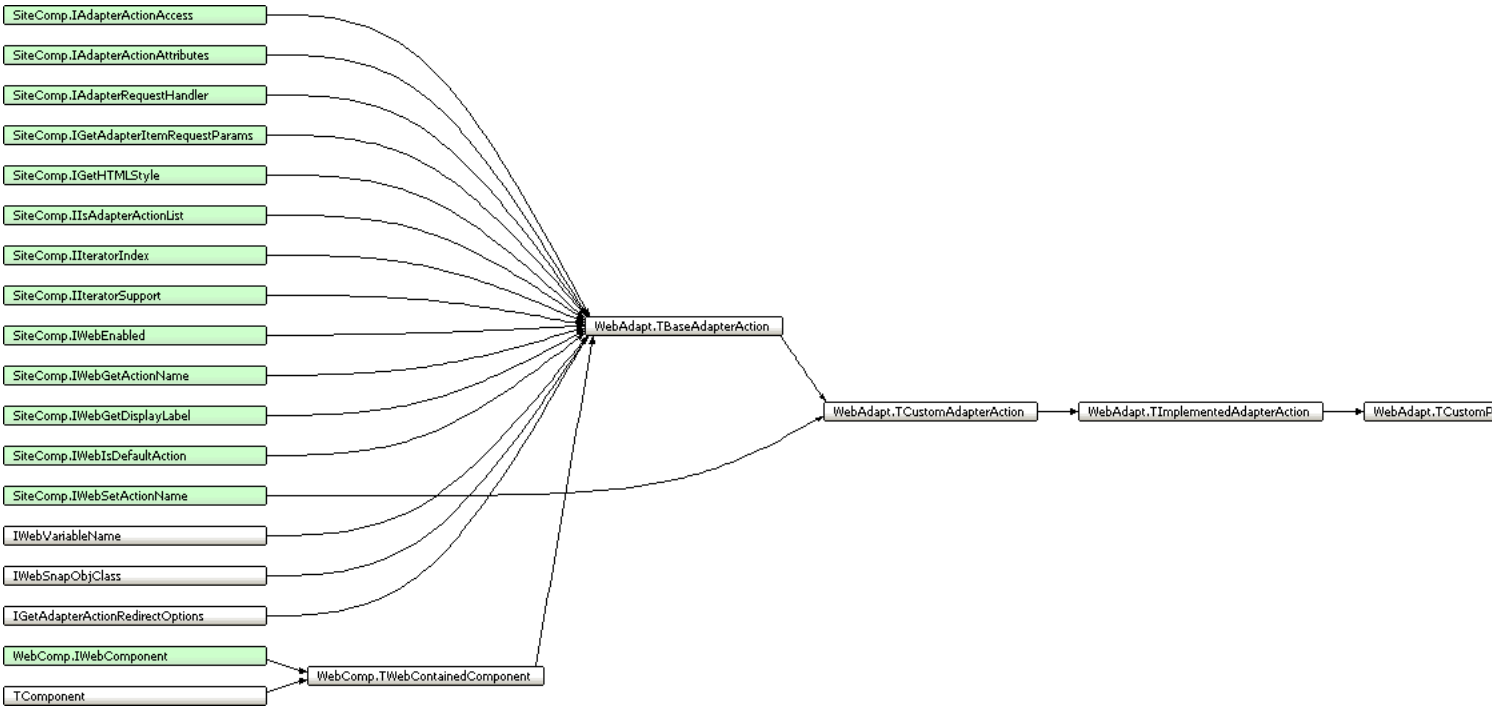
See Also

TPagedAdapter (see page 5830)

57.1.62 TCustomPagedAdapterAction Class

TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5733).

Class Hierarchy



Pascal

```
TCustomPagedAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomPagedAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent adapter. Second, it defines a base class that TDefaultFieldsPagedAdapter (see page 5733) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5803)

TDefaultFieldsPagedAdapter (see page 5733)

TDefaultFieldsPagedAdapter (see page 5733)

57.1.62.1 TCustomPagedAdapterAction Members

The following tables list the members exposed by TCustomPagedAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.

	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5445)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5452)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5570)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.

	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5713)	Returns the parent adapter of this adapter action.
	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5465)	Called after the adapter action has executed.
	OnAfterExecute (see page 5573)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5465)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5573)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5465)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5574)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5466)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5574)	Called before the adapter action response is generated.
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5466)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5462)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

57.1.62.2 TCustomPagedAdapterAction Methods

The methods of the TCustomPagedAdapterAction class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 5713)	Returns the parent adapter of this adapter action.

57.1.62.2.1 TCustomPagedAdapterAction.GetAdapter Method

Returns the parent adapter of this adapter action.

Pascal

```
function GetAdapter: TDefaultFieldsPagedAdapter;
```

C++

```
__fastcall TDefaultFieldsPagedAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 5713) property.

See Also

TDefaultFieldsPagedAdapter (see page 5733)

Adapter (see page 5713)

57.1.62.3 TCustomPagedAdapterAction Properties

The properties of the TCustomPagedAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5713)	Returns the parent adapter of this adapter action.

57.1.62.3.1 TCustomPagedAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TDefaultFieldsPagedAdapter;
```

C++

```
__property TDefaultFieldsPagedAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

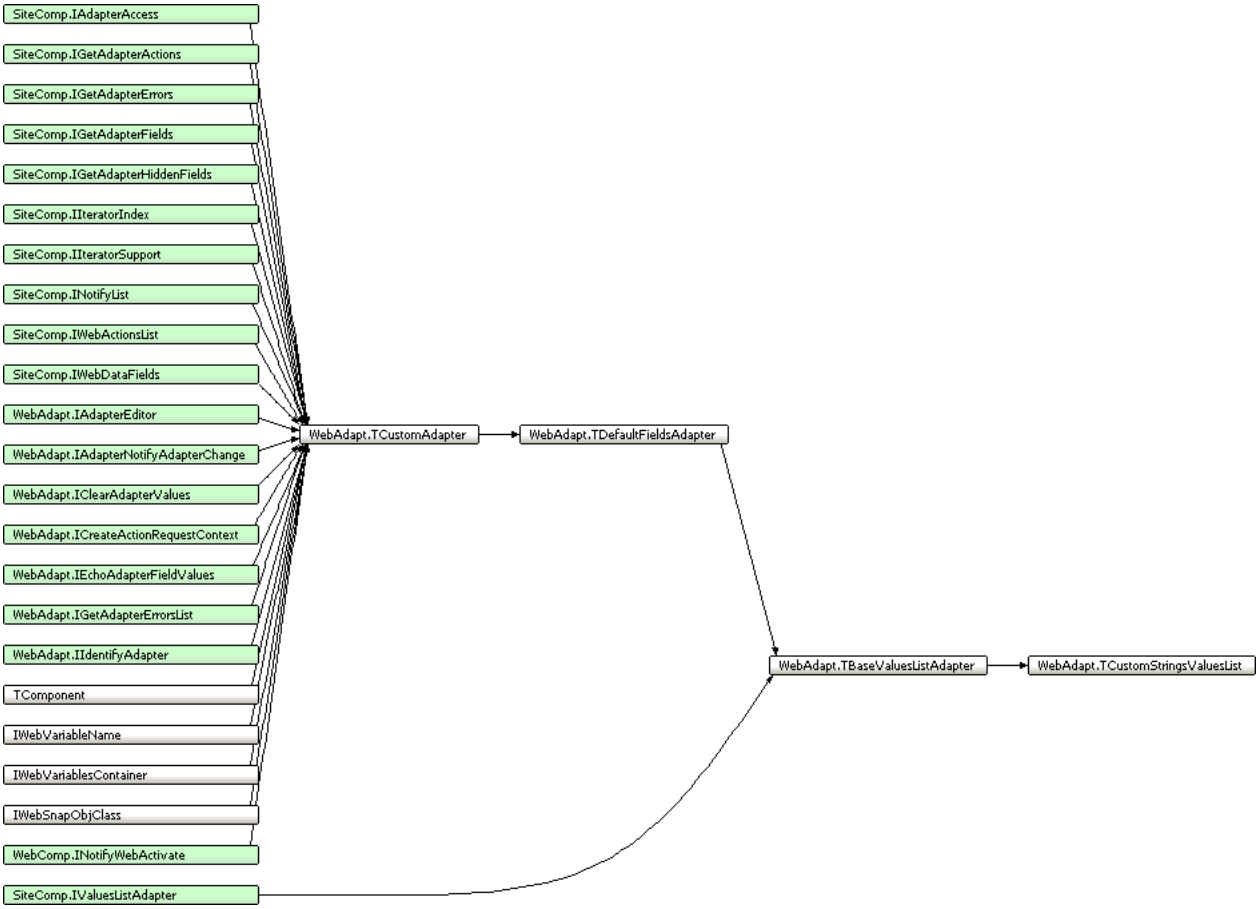
See Also

GetAdapter (see page 5713)

57.1.63 TCustomStringsValuesList Class

TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.

Class Hierarchy



Pascal

```
TCustomStringsValuesList = class(TBaseValuesListAdapter);
```

C++

```
class TCustomStringsValuesList : public TBaseValuesListAdapter;
```

File

WebAdapt

Description

Use descendants of TCustomStringsValuesList with adapter fields to define a list of name/value pairs that are possible values for the adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList property.

See Also

TBaseValuesListAdapter (see page 5504)

TStringsValuesList (see page 5843)

TAdapterUpdateField (see page 5407)

57.1.63.1 TCustomStringsValuesList Members

The following tables list the members exposed by TCustomStringsValuesList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5510)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5510)	Looks up an image for a value.
	GetListName (see page 5511)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5511)	Looks up a name for a value.
	GetListValue (see page 5511)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5512)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5512)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5720)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5513)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 5513)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 5513)	Looks up an image for a value.
	ImplGetListName (see page 5721)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5721)	Looks up a name for a value.
	ImplGetListValue (see page 5721)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5722)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5722)	Clears the iteration index.

V	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
V	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
V	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
V	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
V	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5722)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
V	PrepareStrings (see page 5723)	Customizes the content of the strings list.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5723)	This flag indicates whether the strings list has been prepared or customized.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 5724)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5719)	Create is the constructor for TCustomStringsValuesList (see page 5714).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5719)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetListImage (see page 1015)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1016)	Returns the image object associated with a specified value.
	GetListName (see page 1016)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1016)	Returns the name associated with a specified value.
	GetListValue (see page 1017)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.

R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	Strings (see page 5723)	This string list defines the list of name/value pairs.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.63.2 TCustomStringsValuesList.Create Constructor

Create is the constructor for TCustomStringsValuesList (see page 5714).

Pascal

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

C++

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

Description

Create creates the strings list.

See Also

Strings (see page 5723)

57.1.63.3 TCustomStringsValuesList.Destroy Destructor

Destroy is the destructor for TCustomStringsList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStringsValuesList();
```

Description

Destroy frees the strings list.

See Also

Strings (see page 5723)

57.1.63.4 TCustomStringsValuesList Methods

The methods of the TCustomStringsValuesList class are listed here.

Protected Methods

	Name	Description
V	GetNameFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.

	ImplEndIterator (see page 5720)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.
	ImplGetListName (see page 5721)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5721)	Looks up a name for a value.
	ImplGetListValue (see page 5721)	Responsible for returning the value of the name/value pair.
	ImplNextIteration (see page 5722)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 5722)	Clears the iteration index.
	ImplStartIterator (see page 5722)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	PrepareStrings (see page 5723)	Customizes the content of the strings list.

57.1.63.4.1 TCustomStringsValuesList.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns a TValuesListNameField (see page 5852).

See Also

GetValueFieldClass (see page 5720)

TValuesListNameField (see page 5852)

57.1.63.4.2 TCustomStringsValuesList.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns TValuesListValueField (see page 5855).

See Also

GetNameFieldClass (see page 5720)

TValuesListValueField (see page 5855)

57.1.63.4.3 TCustomStringsValuesList.ImplEndIterator Method

Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.

See Also

enditerator (see page 997)

57.1.63.4.4 TCustomStringsValuesList.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetListName();
```

Description

ImplGetListName uses the iteration index to retrieve a name from the strings list. The name may be blank.

See Also

Strings (see page 5723)

57.1.63.4.5 TCustomStringsValuesList.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue searches the string list for a matching value and returns the name associated with this value. An empty string is returned if the value is not found or if the name associated with the value is blank.

See Also

Strings (see page 5723)

57.1.63.4.6 TCustomStringsValuesList.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue uses the iteration index to retrieve a value from the strings list.

See Also

Strings (see page 5723)

57.1.63.4.7 TCustomStringsValuesList.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration increments the iteration index. false is returned if the iteration index is out of the range defined by the Strings (see page 5723) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

NextIteration (see page 997)

57.1.63.4.8 TCustomStringsValuesList.ImplNotifyDeactivate Method

Clears the iteration index.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The StringsPrepared (see page 5723) property is cleared. This ensures that the strings list will be prepared during the processing of the next HTTP request.

57.1.63.4.9 TCustomStringsValuesList.ImplStartIterator Method

Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator initializes the iteration index to 0. false is returned if the iteration index is out of the range defined by the Strings (see page 5723) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

startiterator (see page 998)

57.1.63.4.10 TCustomStringsValuesList.PrepareStrings Method

Customizes the content of the strings list.

Pascal

```
procedure PrepareStrings; virtual;
```

C++

```
virtual __fastcall PrepareStrings();
```

Description

PrepareStrings calls the OnPrepareStrings (see page 5724) event handler.

See Also

StringsPrepared (see page 5723)

OnPrepareStrings (see page 5724)

57.1.63.5 TCustomStringsValuesList Properties

The properties of the TCustomStringsValuesList class are listed here.

Protected Properties

	Name	Description
	StringsPrepared (see page 5723)	This flag indicates whether the strings list has been prepared or customized.

Public Properties

	Name	Description
	Strings (see page 5723)	This string list defines the list of name/value pairs.

57.1.63.5.1 TCustomStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

57.1.63.5.2 TCustomStringsValuesList.StringsPrepared Property

This flag indicates whether the strings list has been prepared or customized.

Pascal

```
property StringsPrepared: Boolean;
```

C++

```
__property Boolean StringsPrepared;
```

Description

This flag is checked before calling PrepareStrings (see page 5723).

See Also

PrepareStrings (see page 5723)

57.1.63.6 TCustomStringsValuesList Events

The events of the TCustomStringsValuesList class are listed here.

Public Events

	Name	Description
	OnPrepareStrings (see page 5724)	Implement OnPrepareStrings to customize the content of the strings list.

57.1.63.6.1 TCustomStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

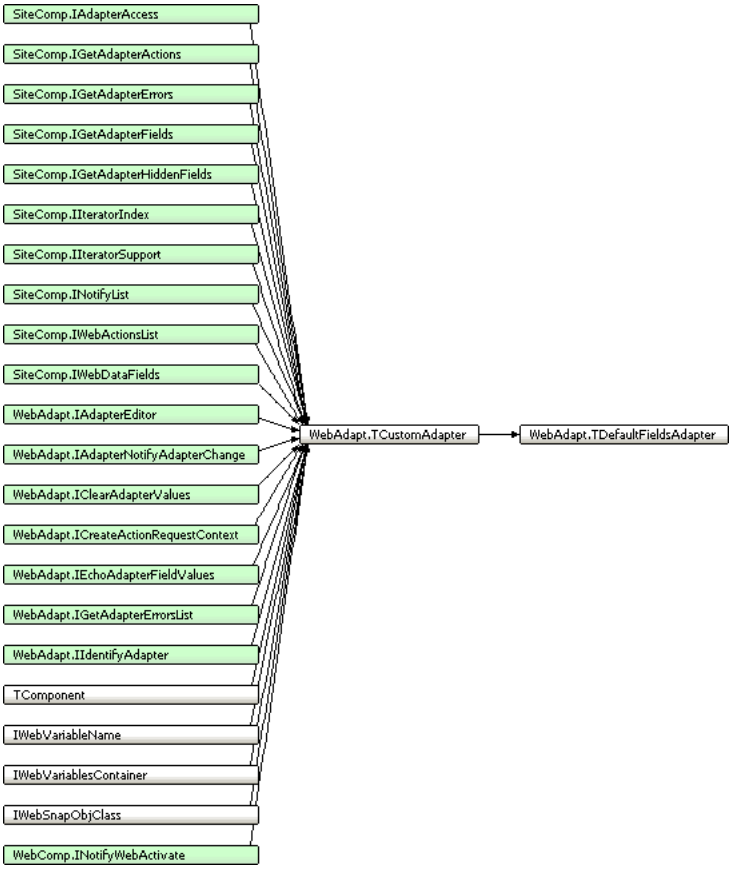
TNotifyEvent

PrepareStrings (see page 5723)

57.1.64 TDefaultFieldsAdapter Class

TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.

Class Hierarchy



Pascal

```
TDefaultFieldsAdapter = class(TCustomAdapter);
```

C++

```
class TDefaultFieldsAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Derive from TDefaultFieldsAdapter to define adapter components that automatically add adapter field and/or adapter action components. Default fields are added to the adapter unless one or more field has been explicitly added.

Default actions are added to the adapter unless one or more action has been explicitly added.

See Also

TCustomAdapter (see page 5519)

57.1.64.1 TDefaultFieldsAdapter Members

The following tables list the members exposed by TDefaultFieldsAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.

	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.

	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.

	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.64.2 TDefaultFieldsAdapter Methods

The methods of the TDefaultFieldsAdapter class are listed here.

Protected Methods

	Name	Description
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
V	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.

57.1.64.2.1 TDefaultFieldsAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions calls ImplGetActionsList (see page 5542). Descendant classes should override TCustomAdapter (see page 5519)'s ImplGetActionsList (see page 5542) method to provide the list of default actions that will be automatically added to this adapter.

See Also

TWebComponentList (see page 5999)

ImplGetActionsList (see page 5542)

57.1.64.2.2 TDefaultFieldsAdapter.AddDefaultFields Method

Adds default field components to this adapter.

Pascal

```
procedure AddDefaultFields(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultFields(TWebComponentList AComponents);
```

Description

AddDefaultFields calls ImplGetFieldList. Descendant classes should override TCustomAdapter (see page 5519)'s ImplGetFieldList method to provide the list of default fields that will be automatically added to this adapter.

See Also

TWebComponentList (see page 5999)

ImplGetFieldsList (see page 5542)

57.1.64.2.3 TDefaultFieldsAdapter.DefaultWebComponentsChange Method

Notifies other components of changes to the list of default actions or fields.

Pascal

```
procedure DefaultWebComponentsChange(Sender: TObject);
```

C++

```
__fastcall DefaultWebComponentsChange(TObject * Sender);
```

Description

The list of default fields or actions may be based on properties of the adapter or properties of other components. For example, the default fields of TDataSetAdapter are based on the fields in the dataset. If the TDataSetAdapter references a different dataset then the default fields list will change. If the dataset's field list changes then TDataSetAdapter's default fields will change. DefaultWebCompoentsChange is called in these circumstances so that components that are using this adapter can be notified of changes to the list of fields or actions. DefaultWebComponentsChange calls NotifyAdapterChange (see page 5551).

See Also

NotifyAdapterChange (see page 5551)

57.1.64.2.4 TDefaultFieldsAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; override;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

GetActionsClass returns TAdapterDefaultActions (see page 5220).

See Also

TAdapterDefaultActions (see page 5220)

57.1.64.2.5 TDefaultFieldsAdapter.GetDataClass Method

Specifies the class to create as a container for adapter field components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; override;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

GetDataClass returns TAdapterDefaultFields (see page 5226).

See Also

TAdapterDefaultFields (see page 5226)

57.1.64.2.6 TDefaultFieldsAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions returns a list of actions that have been added to the adapter. If no adapter fields have been explicitly added, ImplGetVisibleActions returns a list containing all default actions.

See Also

TWebComponentList (see page 5999)

ImplGetVisibleActions (see page 5544)

57.1.64.2.7 TDefaultFieldsAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields returns a list of fields that have been added to the adapter. If no adapter fields have been explicitly added,

ImplGetVisibleFields returns a list containing all default fields.

See Also

TWebComponentList (see page 5999)

ImplGetVisibleFields (see page 5544)

57.1.64.2.8 TDefaultFieldsAdapter.ImplSetActions Method

Initializes the Actions (see page 5732) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5732) property is the container for adapter action components.

See Also

ImplSetActions (see page 5547)

57.1.64.2.9 TDefaultFieldsAdapter.ImplSetData Method

Initializes the Data (see page 5733) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5733) property is the container for adapter field components.

See Also

ImplSetData (see page 5548)

57.1.64.3 TDefaultFieldsAdapter Properties

The properties of the TDefaultFieldsAdapter class are listed here.

Public Properties

	Name	Description
	Actions (see page 5732)	Container for adapter actions components.
	Data (see page 5733)	Container for adapter field components.

57.1.64.3.1 TDefaultFieldsAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterDefaultActions;
```

C++

```
__property TAdapterDefaultActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.64.3.2 TDefaultFieldsAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterDefaultFields;
```

C++

```
__property TAdapterDefaultFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

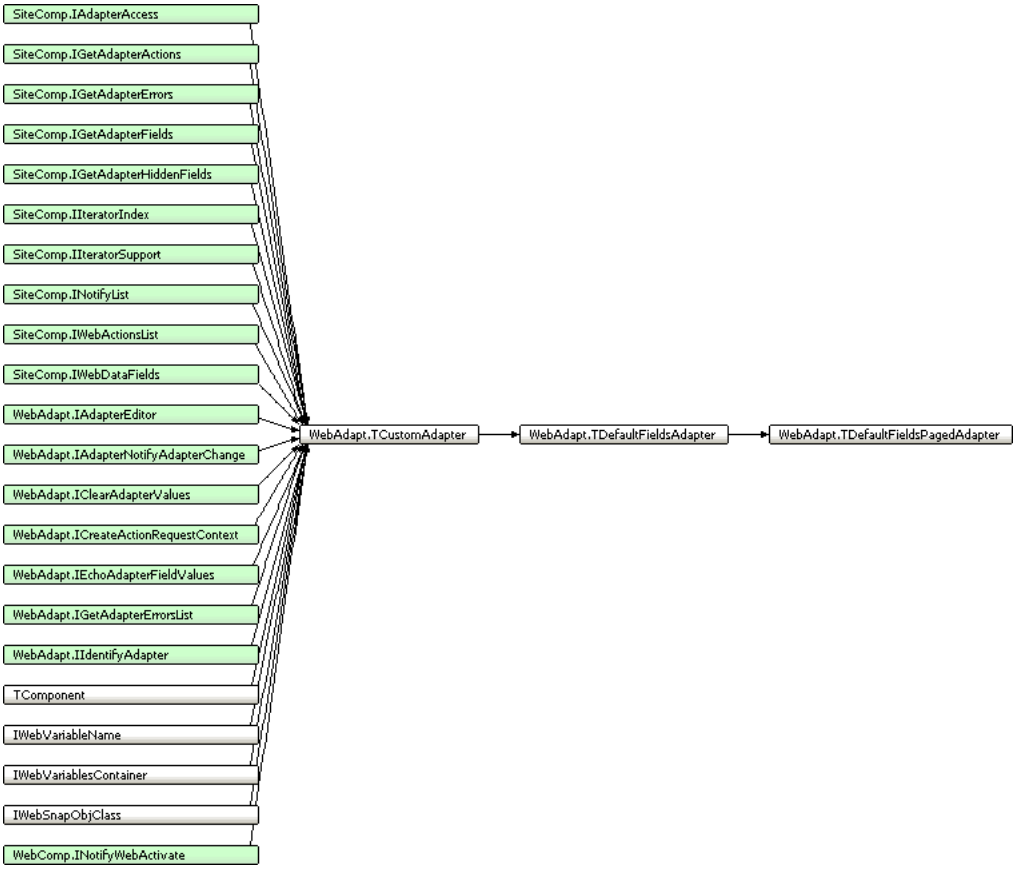
See Also

TAdapterDefaultFields (see page 5226)

57.1.65 TDefaultFieldsPagedAdapter Class

TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

Class Hierarchy



Pascal

```
TDefaultFieldsPagedAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TDefaultFieldsPagedAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

Enable paging by setting the PageSize (see page 5747) property. When paging is enabled, the way that the adapter enumerates records is affected. The adapter calculates the number of pages by dividing the number of records by the page size. It keeps track of a current page number. When the enumerating records, the records on the current page are enumerated. Server-side script simply enumerates the records. Server-side script does not need to be aware of the page size or current page number. When paging is enabled, default actions for moving between pages are automatically added to the adapter.

See Also

TDefaultFieldsAdapter (see page 5724)

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

57.1.65.1 TDefaultFieldsPagedAdapter Members

The following tables list the members exposed by TDefaultFieldsPagedAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddDefaultActions (see page 5739)	Adds default action components to this adapter.
	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
	EncodeActionParams (see page 5740)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 5740)	Indicates whether the record index has gone past the last record in the current page.
	ExtractRequestParams (see page 5740)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEOF (see page 5741)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetFirstRecord (see page 5741)	Set the record position to the first record.

	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextRecord (see page 5741)	Increments the record position.
	GetPageCount (see page 5742)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5742)	Returns the total number of records.
	GetRecordIndex (see page 5742)	Returns the index of the current record.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5743)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5743)	Implements IteratorSupport's EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5743)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5744)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5744)	Clears the current page variable.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5745)	Implements IteratorSupport.StartIterator (see page 5553) to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.

	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	SetPageSize (see page 5745)	Updates the pagesize variable.
	SkipToPage (see page 5746)	Skips records until the current record is within the current page.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5748)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5749)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5749)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5749)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5749)	Implement this event to identify the zero-based current record.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.

	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
	CurrentPage (see page 5746)	Sets or gets the current page index.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
R	PageCount (see page 5747)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 5747)	Sets or gets the page size variable.
R	RecordCount (see page 5747)	Returns the total number of records.
R	RecordIndex (see page 5748)	Returns the index of the current record.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.

	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.65.2 TDefaultFieldsPagedAdapter Methods

The methods of the TDefaultFieldsPagedAdapter class are listed here.

Protected Methods

	Name	Description
	AddDefaultActions (see page 5739)	Adds default action components to this adapter.
	EncodeActionParams (see page 5740)	Writes name/value pairs that represent adapter state.
	EndOfPage (see page 5740)	Indicates whether the record index has gone past the last record in the current page.
	ExtractRequestParams (see page 5740)	Reads name/value pairs from the HTTP request.
	GetEOF (see page 5741)	Indicates whether the record position has passed the last record.
	GetFirstRecord (see page 5741)	Set the record position to the first record.
	GetNextRecord (see page 5741)	Increments the record position.
	GetPageCount (see page 5742)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5742)	Returns the total number of records.
	GetRecordIndex (see page 5742)	Returns the index of the current record.
	ImplCanAddActionClass (see page 5743)	Indicates whether a type of adapter action can be added to this adapter.
	ImplEndIterator (see page 5743)	Implements IteratorSupport's EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplGetActionsList (see page 5743)	Provides a list of action types that can be added to this adapter.
	ImplNextIteration (see page 5744)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 5744)	Clears the current page variable.
	ImplStartIterator (see page 5745)	Implements IteratorSupport.StartIterator (see page 5553) to support iterating over the records of the adapter.
	SetPageSize (see page 5745)	Updates the pagesize variable.
	SkipToPage (see page 5746)	Skips records until the current record is within the current page.

57.1.65.2.1 TDefaultFieldsPagedAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); override;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions adds TAdapterNextPageAction (see page 5358), TAdapterGotoPageAction (see page 5280), and TAdapterPrevPageAction (see page 5384).

See Also

TWebComponentList (see page 5999)

TAdapterNextPageAction (see page 5358)

TAdapterGotoPageAction (see page 5280)

TAdapterPrevPageAction (see page 5384)

57.1.65.2.2 TDefaultFieldsPagedAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

ExtractRequestParams (see page 5740) writes the current page number as a name/value pair.

See Also

ExtractRequestParams (see page 5530)

ExtractRequestParams (see page 5740)

57.1.65.2.3 TDefaultFieldsPagedAdapter.EndOfPage Method

Indicates whether the record index has gone past the last record in the current page.

Pascal

```
function EndOfPage: Boolean;
```

C++

```
__fastcall Boolean EndOfPage();
```

Description

EndOfPage tests for the end of page using PageSize (see page 5747), RecordIndex (see page 5748), and CurrentPage (see page 5746).

See Also

PageSize (see page 5747)

RecordIndex (see page 5748)

PageSize (see page 5747)

CurrentPage (see page 5746)

57.1.65.2.4 TDefaultFieldsPagedAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); override;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. ExtractRequestParams extracts the current page number from the request.

See Also

EncodeActionParams (see page 5740)

ExtractRequestParams (see page 5530)

57.1.65.2.5 TDefaultFieldsPagedAdapter.GetEOF Method

Indicates whether the record position has passed the last record.

Pascal

```
function GetEOF: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEOF();
```

Description

GetEOF calls the OnGetFirstRecord (see page 5749) event handler. If OnGetFirstRecord (see page 5749) is not implemented then the RecordIndex (see page 5748) is compared to the RecordCount (see page 5747). Descendant classes may override this method. For example, TDataSetAdapter tests the dataset's EOF.

See Also

OnGetEOF (see page 5748)

RecordIndex (see page 5748)

RecordCount (see page 5747)

57.1.65.2.6 TDefaultFieldsPagedAdapter.GetFirstRecord Method

Set the record position to the first record.

Pascal

```
function GetFirstRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetFirstRecord();
```

Description

GetFirstRecord calls the OnGetFirstRecord (see page 5749) event handler. Descendant classes may override this method. For example, TDataSetAdapter sets the record position of a dataset.

See Also

OnGetFirstRecord (see page 5749)

57.1.65.2.7 TDefaultFieldsPagedAdapter.GetNextRecord Method

Increments the record position.

Pascal

```
function GetNextRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNextRecord();
```

Description

GetNextRecord calls the OnGetNextRecord (see page 5749) event handler. Descendant classes may override this method. For example, TDataSetAdapter increments the record position of a dataset.

See Also

OnGetNextRecord (see page 5749)

57.1.65.2.8 TDefaultFieldsPagedAdapter.GetPageCount Method

Calculates the number of pages needed to display all adapter records.

Pascal

```
function GetPageCount: Integer;
```

C++

```
__fastcall int GetPageCount();
```

Description

GetPageCount calculates the page count using PageCount (see page 5747), PageSize (see page 5747), and RecordCount (see page 5747).

See Also

RecordCount (see page 5747)

PageCount (see page 5747)

PageSize (see page 5747)

57.1.65.2.9 TDefaultFieldsPagedAdapter.GetRecordCount Method

Returns the total number of records.

Pascal

```
function GetRecordCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

GetRecordCount calls the OnGetRecordCount (see page 5749) event handler. Descendant classes may override this method. For example, TDataSetAdapter returns the number of records in the dataset.

See Also

OnGetRecordCount (see page 5749)

RecordCount (see page 5747)

57.1.65.2.10 TDefaultFieldsPagedAdapter.GetRecordIndex Method

Returns the index of the current record.

Pascal

```
function GetRecordIndex: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordIndex();
```

Description

GetRecordIndex calls the OnGetRecordIndex (see page 5749) event handler.

See Also

OnGetRecordIndex (see page 5749)

57.1.65.2.11 TDefaultFieldsPagedAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows action classes that derive from TCustomPagedAdapterAction (see page 5708) in addition to those allowed by TCustomAdapter (see page 5519)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 5540)

ImplCanAddClass (see page 5201)

TCustomPagedAdapterAction (see page 5708)

57.1.65.2.12 TDefaultFieldsPagedAdapter.ImplEndIterator Method

Implements IteratorSupport's EndIterator (see page 5529) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

EndIterator (see page 5529) is called when record iteration is complete.

See Also

enditerator (see page 997)

57.1.65.2.13 TDefaultFieldsPagedAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList lists PrevPage, GotoPage, and NextPage actions.

See Also

TAdapterNextPageAction (see page 5358)

TAdapterGotoPageAction (see page 5280)

TAdapterPrevPageAction (see page 5384)

ImplGetActionsList (see page 5542)

57.1.65.2.14 TDefaultFieldsPagedAdapter.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration calls GetNextRecord (see page 5741) if the RecordIndex (see page 5748) can be incremented without running past the end of the current page. false is returned if RecordIndex (see page 5748) can't be incremented.

See Also

NextIteration (see page 997)

RecordIndex (see page 5748)

RecordCount (see page 5747)

GetNextRecord (see page 5741)

GetEOF (see page 5741)

EndOfPage (see page 5740)

57.1.65.2.15 TDefaultFieldsPagedAdapter.ImplNotifyDeactivate Method

Clears the current page variable.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The current page is reset to zero when the module is deactivated. This ensures that the current page set during the processing of an HTTP request will not affect another HTTP request. To keep track of the current page for a particular end-user, the current page is typically passed between end-user requests using action parameters.

See Also

ImplNotifyDeactivate (see page 5547)

EncodeActionParams (see page 5740)

ExtractRequestParams (see page 5740)

57.1.65.2.16 TDefaultFieldsPagedAdapter.ImplStartIterator Method

Implements IteratorSupport.StartIterator (see page 5553) to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator locates the first record in the current page using the GetFirstRecord (see page 5741) and SkipToPage (see page 5746) methods.

See Also

startiterator (see page 998)

ImplNextIteration (see page 5744)

GetFirstRecord (see page 5741)

SkipToPage (see page 5746)

CurrentPage (see page 5746)

57.1.65.2.17 TDefaultFieldsPagedAdapter.SetPageSize Method

Updates the pagesize variable.

Pascal

```
procedure SetPageSize(const Value: Integer);
```

C++

```
__fastcall SetPageSize(const int Value);
```

Description

If the pagesize variable is not zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the pagesize variable is changed back to zero then the these actions are removed.

See Also

PageSize (see page 5747)

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

57.1.65.2.18 TDefaultFieldsPagedAdapter.SkipToPage Method

Skips records until the current record is within the current page.

Pascal

```
procedure SkipToPage;
```

C++

```
__fastcall SkipToPage();
```

Description

SkipToPage calls GetNextRecord (see page 5741) to reach the first record in the current page.

See Also

CurrentPage (see page 5746)

GetNextRecord (see page 5741)

57.1.65.3 TDefaultFieldsPagedAdapter Properties

The properties of the TDefaultFieldsPagedAdapter class are listed here.

Public Properties

	Name	Description
	CurrentPage (see page 5746)	Sets or gets the current page index.
R	PageCount (see page 5747)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 5747)	Sets or gets the page size variable.
R	RecordCount (see page 5747)	Returns the total number of records.
R	RecordIndex (see page 5748)	Returns the index of the current record.

57.1.65.3.1 TDefaultFieldsPagedAdapter.CurrentPage Property

Sets or gets the current page index.

Pascal

```
property CurrentPage: Integer;
```

C++

```
__property int CurrentPage;
```

Description

Current page is a zero-based index that must be less than PageCount (see page 5747). Current page is used by the NextPage, GotoPage, and PrevPage actions to move between page.

See Also

PageCount (see page 5747)

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

EncodeActionParams (see page 5740)

ExtractRequestParams (see page 5740)

57.1.65.3.2 TDefaultFieldsPagedAdapter.PageCount Property

Calculates the number of pages needed to display all adapter records.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount returns the result of GetPageCount (see page 5742).

See Also

GetPageCount (see page 5742)

RecordCount (see page 5747)

PageCount

PageSize (see page 5747)

57.1.65.3.3 TDefaultFieldsPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

57.1.65.3.4 TDefaultFieldsPagedAdapter.RecordCount Property

Returns the total number of records.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

The RecordCount property returns the result of GetRecordCount (see page 5742).

See Also

GetRecordCount (see page 5742)

57.1.65.3.5 TDefaultFieldsPagedAdapter.RecordIndex Property

Returns the index of the current record.

Pascal

```
property RecordIndex: Integer;
```

C++

```
__property int RecordIndex;
```

Description

RecordIndex returns the result of GetRecordIndex (see page 5742).

See Also

OnGetRecordIndex (see page 5749)

GetRecordIndex (see page 5742)

57.1.65.4 TDefaultFieldsPagedAdapter Events

The events of the TDefaultFieldsPagedAdapter class are listed here.

Public Events

	Name	Description
	OnGetEOF (see page 5748)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5749)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5749)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5749)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5749)	Implement this event to identify the zero-based current record.

57.1.65.4.1 TDefaultFieldsPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount (see page 5749) event handler will be compared to the result of the OnGetRecordIndex (see page 5749) event handler.

57.1.65.4.2 TDefaultFieldsPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

57.1.65.4.3 TDefaultFieldsPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

57.1.65.4.4 TDefaultFieldsPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

57.1.65.4.5 TDefaultFieldsPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

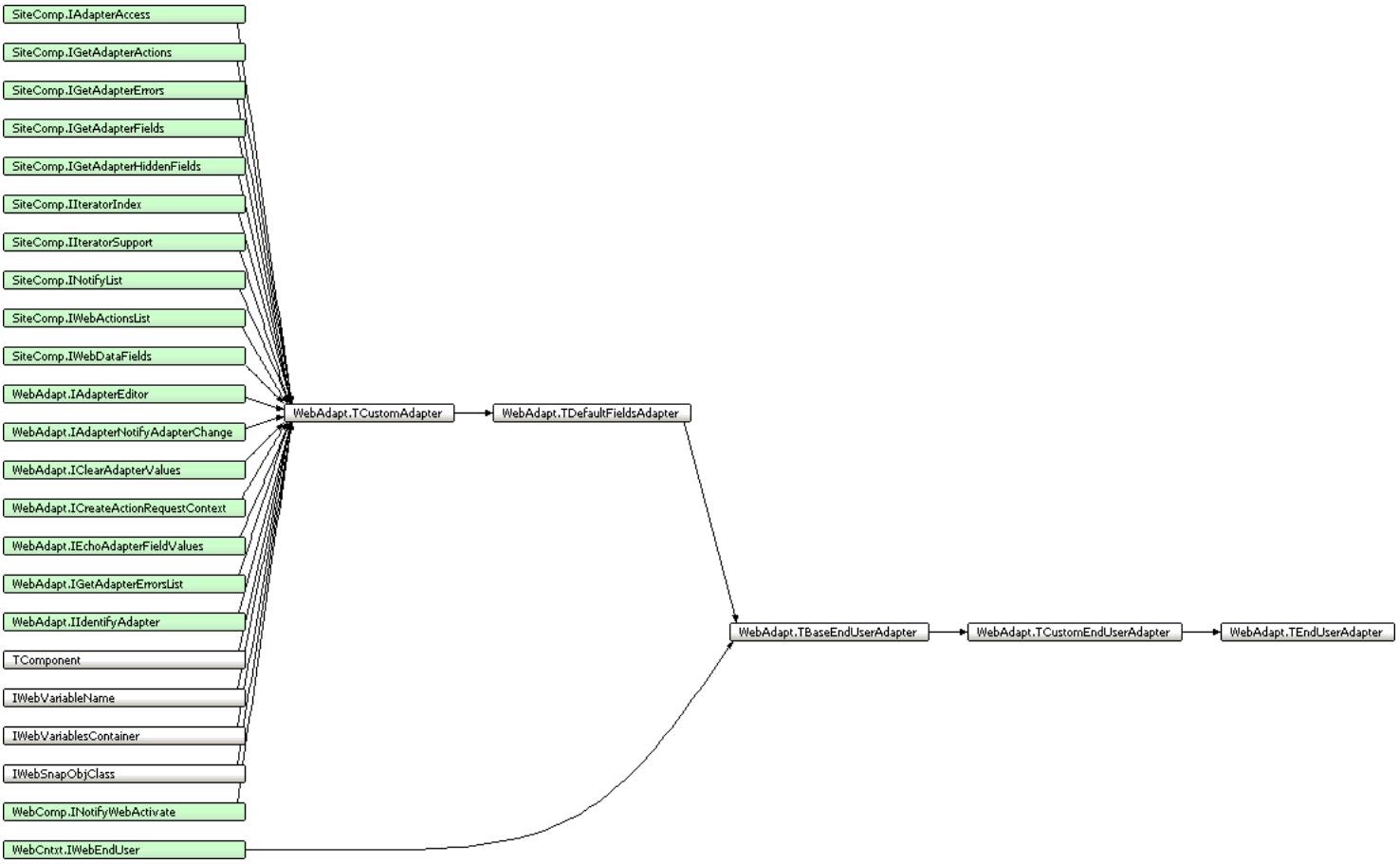
Description

OnGetRecordIndex is called to calculate the current page.

57.1.66 TEndUserAdapter Class

TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TEndUserAdapter = class(TCustomEndUserAdapter);
```

C++

```
class TEndUserAdapter : public TCustomEndUserAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter (see page 5779) to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter (see page 5779) handles login and logout actions on behalf of the end user by generating events where you can retrieve user information and execute user actions

TEndUserAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's

name and login state. Implement the OnGetDisplayName (see page 5759) and OnIsLoggedIn (see page 5760) events to provide values for these fields.

TEndUserAdapter contains two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to log the user out. Use the LoginPage (see page 5757) property to specify the name of the login page. Implement the OnLogin (see page 5761) and OnLogout (see page 5761) events to change the login state of an end user.

The EndUser script variable is used to by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 5760) event to check rights in an event handler.

57.1.66.1 TEndUserAdapter Members

The following tables list the members exposed by TEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5492)	Starts the server-side login process.

	ExecuteLogin (see page 5669)	Generates an OnLogin (see page 5672) event.
	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
	ExecuteLogout (see page 5493)	Performs server-side logout.
	ExecuteLogout (see page 5669)	Generates an OnLogout (see page 5673) event.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5493)	Retrieves the display name string.
	GetDisplayNameValue (see page 5494)	Gets the display name of the end user.
	GetDisplayNameValue (see page 5669)	Generates an OnGetDisplayName (see page 5671) event.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5495)	Determines if the user is logged in
	GetLoggedInValue (see page 5495)	Determines if the user is logged in.
	GetLoggedInValue (see page 5670)	Generates an OnIsLoggedIn (see page 5672) event.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
	GetUserID (see page 5496)	Gets the UserID from this TBaseEndUserAdapter (see page 5485).
	GetUserID (see page 5670)	Generates an OnGetUserID (see page 5671) event.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.

	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5501)	Processes the login for the given user.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.

	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5502)	Logs in a specified user.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5502)	Generates the login page.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
V	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5503)	Gets the user list.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetUserID (see page 5671)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.

	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5967)	Logs in a specified user.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.

	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5757)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5758)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5758)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5758)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5759)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5759)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 5760)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5760)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5761)	Logs in the current end user.
	OnLogout (see page 5761)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5762)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 5756)	Container for adapter actions components.
	Data (see page 5756)	Container for adapter field components.
	LoginPage (see page 5757)	Specifies the name of the page that acts as the login page.

57.1.66.2 TEndUserAdapter Properties

The properties of the TEndUserAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5756)	Container for adapter actions components.
	Data (see page 5756)	Container for adapter field components.
	LoginPage (see page 5757)	Specifies the name of the page that acts as the login page.

57.1.66.2.1 TEndUserAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.66.2.2 TEndUserAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.66.2.3 TEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

57.1.66.3 TEndUserAdapter Events

The events of the TEndUserAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5757)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5758)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5758)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5758)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5759)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5759)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 5760)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5760)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5761)	Logs in the current end user.
	OnLogout (see page 5761)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5762)	Allows implementation of a customized login page.

57.1.66.3.1 TEndUserAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.66.3.2 TEndUserAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.66.3.3 TEndUserAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.66.3.4 TEndUserAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.66.3.5 TEndUserAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.66.3.6 TEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser interface) requests the display name for the current end user, TCustomEndUserAdapter generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 5968)

EndUser (see page 5947)

WebContext (see page 5977)

57.1.66.3.7 TEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5492)

ImplHasExecuteRights (see page 5500)

ImplHasModifyRights (see page 5500)

ImplHasViewRights (see page 5501)

57.1.66.3.8 TEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser interface) inquires whether the current user has logged in, TCustomEndUserAdapter generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 5968)

EndUser (see page 5947)

WebContext (see page 5977)

OnLogin (see page 5672)

OnLogout (see page 5673)

57.1.66.3.9 TEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to allow the TCustomEndUserAdapter object to log users in. When a server-side script (or a caller of the IWebEndUser interface) attempts to log in a user, TCustomEndUserAdapter generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users in.

See Also

LogIn (see page 5967)

EndUser (see page 5947)

WebContext (see page 5977)

OnIsLoggedIn (see page 5672)

OnLogout (see page 5673)

TWebUserList (see page 6759)

OnGetUserID (see page 5671)

57.1.66.3.10 TEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write an OnLogout event handler to allow the TCustomEndUserAdapter object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users out.

See Also

OnIsLoggedIn (see page 5672)

OnLogin (see page 5672)

57.1.66.3.11 TEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5501)

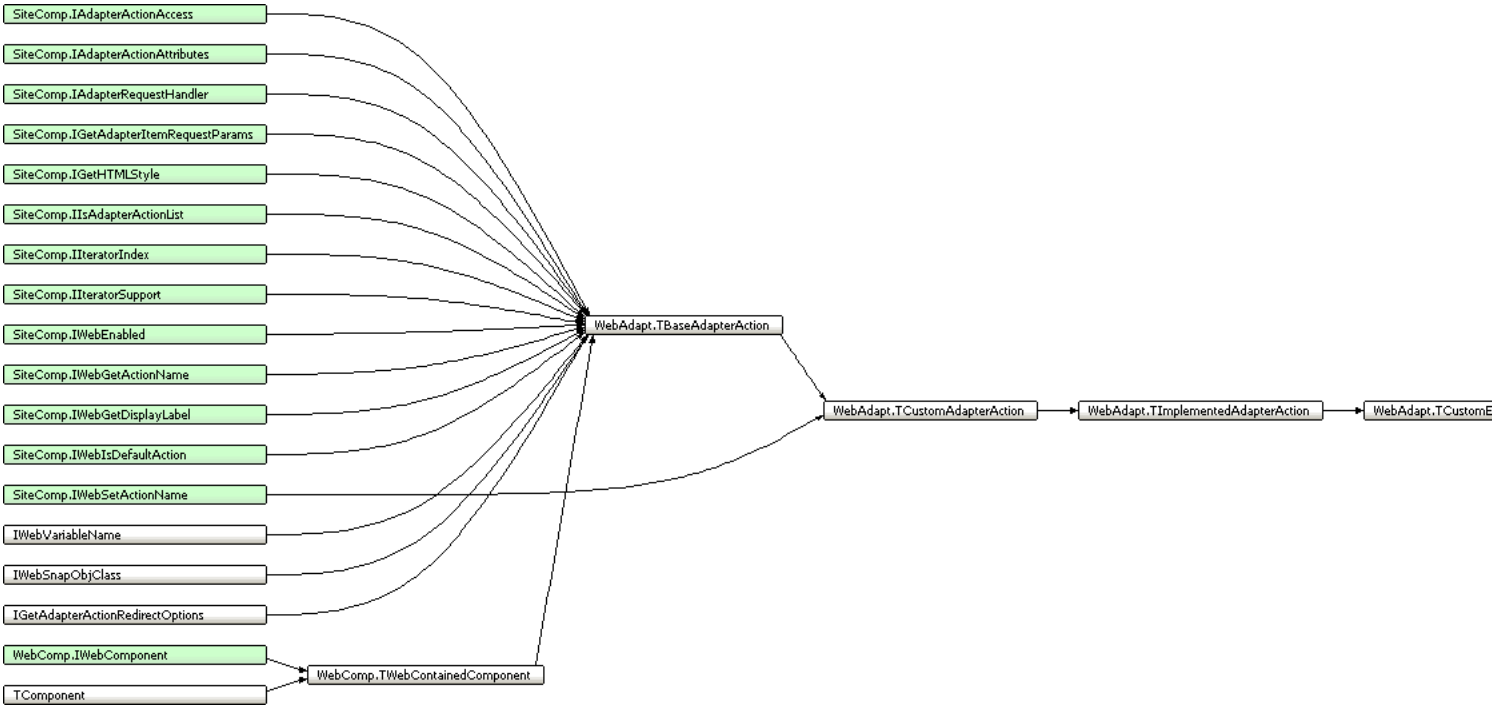
ProduceLoginPage (see page 5502)

CanProduceLoginPage (see page 5492)

57.1.67 TEndUserAdapterLoginFormAction Class

TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.

Class Hierarchy



Pascal

```
TEndUserAdapterLoginFormAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLoginFormAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This adapter action is typically executed when the end-user clicks on a "Login" hyperlink displayed at the top of a web page. The parent adapter's LoginPage property is used to specify the page displayed when this adapter action is executed.

See Also

TCustomEndUserAdapterAction (see page 5673)

TBaseEndUserAdapter (see page 5485)

TLoginFormAdapter (see page 5810)

TBaseEndUserAdapter (see page 5485)

57.1.67.1 TEndUserAdapterLoginFormAction Members

The following tables list the members exposed by TEndUserAdapterLoginFormAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5766)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.

	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
	ImplExecuteActionRequest (see page 5767)	Executes the adapter action.
	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
	ImplWebEnabled (see page 5767)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.

	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.
R	Adapter (see page 5678)	Returns the parent adapter of this adapter action.
R	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.

R	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5768)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5769)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5769)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5769)	Called before the adapter action response is generated.
	OnGetParams (see page 5770)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5768)	Suggests a display label for this adapter action.

57.1.67.2 TEndUserAdapterLoginFormAction Methods

The methods of the TEndUserAdapterLoginFormAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 5766)	Gets the name used to reference this adapter action with server-side script.
V	ImplExecuteActionRequest (see page 5767)	Executes the adapter action.
V	ImplWebEnabled (see page 5767)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.67.2.1 TEndUserAdapterLoginFormAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "LoginForm".

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.67.2.2 TEndUserAdapterLoginFormAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogin method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest

IActionResponse

ImplExecuteActionRequest (see page 5452)

ExecuteLogin (see page 5492)

Errors (see page 5559)

57.1.67.2.3 TEndUserAdapterLoginFormAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLoginForm function.

See Also

CanExecuteLoginForm (see page 5491)

WebEnabled (see page 5461)

57.1.67.3 TEndUserAdapterLoginFormAction Properties

The properties of the TEndUserAdapterLoginFormAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5768)	Suggests a display label for this adapter action.

57.1.67.3.1 TEndUserAdapterLoginFormAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.67.4 TEndUserAdapterLoginFormAction Events

The events of the TEndUserAdapterLoginFormAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5768)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5769)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5769)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5769)	Called before the adapter action response is generated.
	OnGetParams (see page 5770)	Specifies name/value pairs needed to execute this adapter action.

57.1.67.4.1 TEndUserAdapterLoginFormAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.67.4.2 TEndUserAdapterLoginFormAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.67.4.3 TEndUserAdapterLoginFormAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.67.4.4 TEndUserAdapterLoginFormAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.67.4.5 TEndUserAdapterLoginFormAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

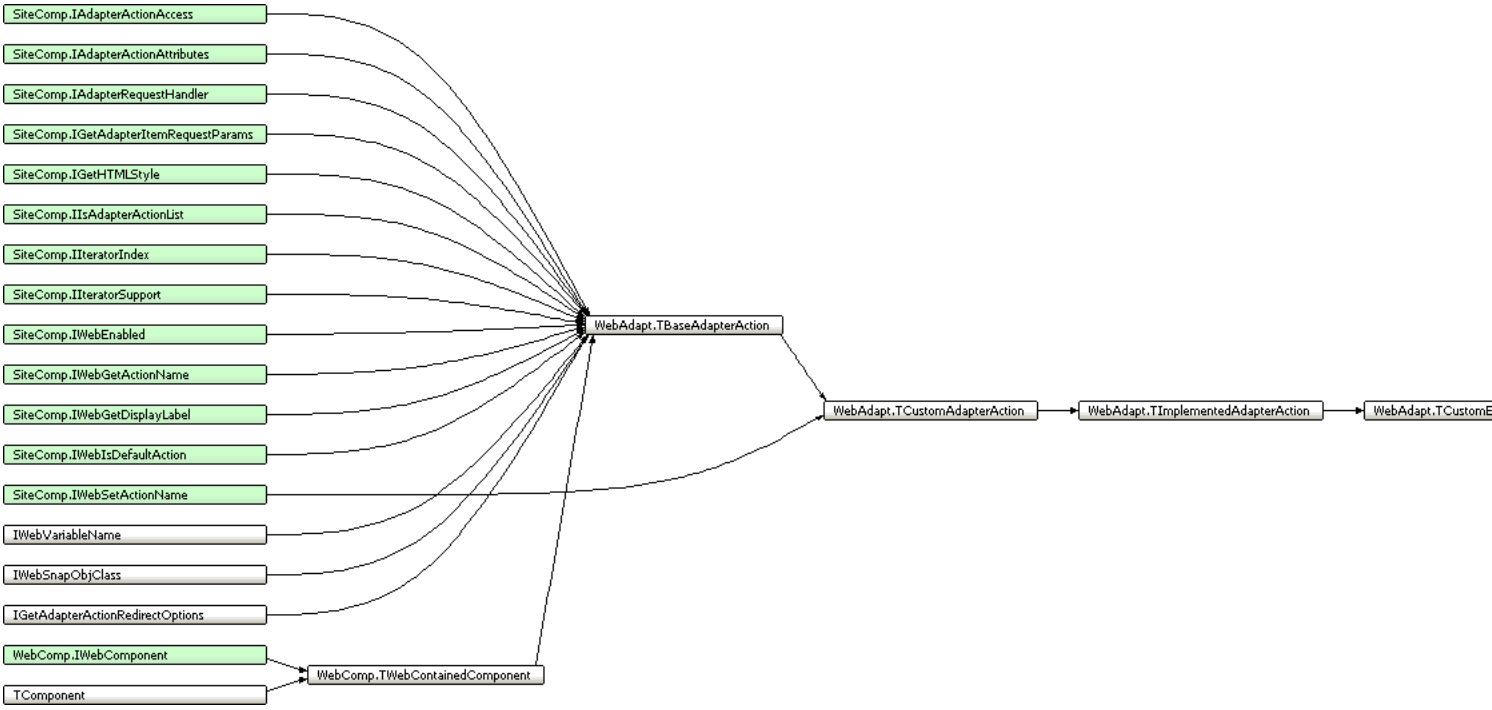
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.68 TEndUserAdapterLogoutAction Class

TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.

Class Hierarchy



Pascal

```
TEndUserAdapterLogoutAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLogoutAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This action is typically executed when the end-user clicks on a "Logout" hyperlink displayed at the top of a web page.

57.1.68.1 TEndUserAdapterLogoutAction Members

The following tables list the members exposed by TEndUserAdapterLogoutAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.

	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5775)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5775)	Executes the adapter action.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5453)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5570)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
V	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.

	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
V	ImplWebEnabled (see page 5775)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.
R	Adapter (see page 5678)	Returns the parent adapter of this adapter action.
R	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.

	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5777)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5777)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5777)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5778)	Called before the adapter action response is generated.
	OnGetParams (see page 5778)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5776)	Suggests a display label for this adapter action.

57.1.68.2 TEndUserAdapterLogoutAction Methods

The methods of the TEndUserAdapterLogoutAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5775)	Gets the name used to reference this adapter action with server-side script.

	ImplExecuteActionRequest (see page 5775)	Executes the adapter action.
	ImplWebEnabled (see page 5775)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

57.1.68.2.1 TEndUserAdapterLogoutAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Logout".

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.68.2.2 TEndUserAdapterLogoutAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogout method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest

IActionResponse

ImplExecuteActionRequest (see page 5452)

ExecuteLogout (see page 5493)

Errors (see page 5559)

57.1.68.2.3 TEndUserAdapterLogoutAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLogout function.

See Also

CanExecuteLogout (see page 5491)

WebEnabled (see page 5461)

57.1.68.3 TEndUserAdapterLogoutAction Properties

The properties of the TEndUserAdapterLogoutAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5776)	Suggests a display label for this adapter action.

57.1.68.3.1 TEndUserAdapterLogoutAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.68.4 TEndUserAdapterLogoutAction Events

The events of the TEndUserAdapterLogoutAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5777)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5777)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5777)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5778)	Called before the adapter action response is generated.
	OnGetParams (see page 5778)	Specifies name/value pairs needed to execute this adapter action.

57.1.68.4.1 TEndUserAdapterLogoutAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.68.4.2 TEndUserAdapterLogoutAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.68.4.3 TEndUserAdapterLogoutAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.68.4.4 TEndUserAdapterLogoutAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.68.4.5 TEndUserAdapterLogoutAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

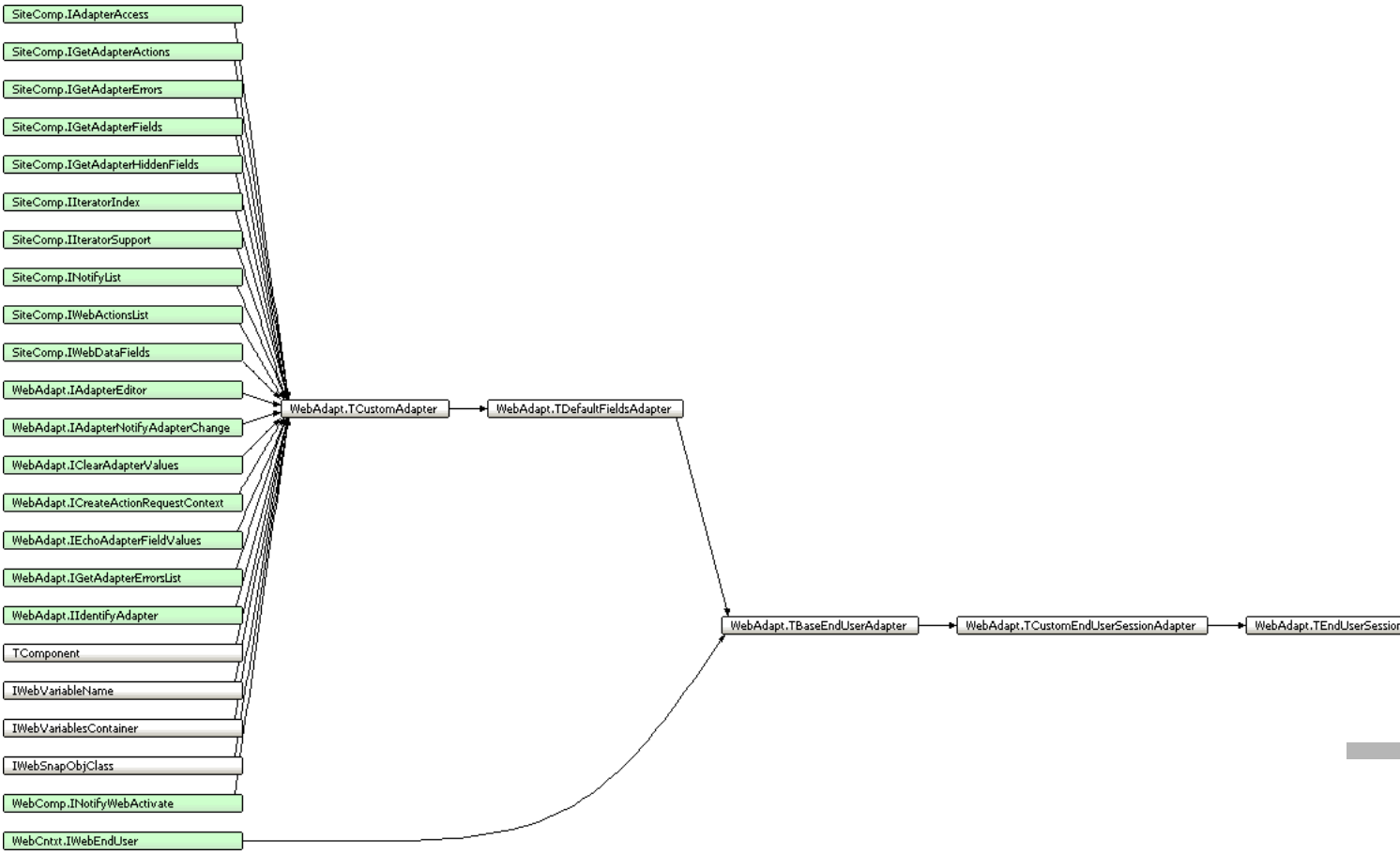
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.69 TEndUserSessionAdapter Class

TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TEndUserSessionAdapter = class(TCustomEndUserSessionAdapter);
```

C++

```
class TEndUserSessionAdapter : public TCustomEndUserSessionAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter handles login and logout actions on behalf of the end user. TEndUserSessionAdapter uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TEndUserSessionAdapter to work properly.

TEndUserSessionAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's name and login state. TEndUserSessionAdapter retrieves the values for these fields from a Web session, which is maintained by the TSessionsService component.

TEndUserSessionAdapter supports two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to log the user out. Use the LoginPage (see page 5786) property to specify the name of the login page. The default implementation of the logout action terminates the session.

The EndUser script variable is used by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserSessionAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter (see page 5750) calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 5788) event to check rights in an event handler.

See Also

TEndUserAdapter (see page 5750)

TSessionsService (see page 6722)

TLoginFormAdapter (see page 5810)

TWebUserList (see page 6759)

IWebEndUser (see page 5962)

Adding Login Support

57.1.69.1 TEndUserSessionAdapter Members

The following tables list the members exposed by TEndUserSessionAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5491)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5491)	Determines if the user is logged in.

	CanProduceLoginPage (see page 5492)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
	DoOnHasRights (see page 5492)	Determines whether the user has view rights.
	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	ExecuteLogin (see page 5492)	Starts the server-side login process.
	ExecuteLogin (see page 5685)	Logs in an end user.
	ExecuteLoginForm (see page 5493)	Creates the HTML login page.
	ExecuteLogout (see page 5493)	Performs server-side logout.
	ExecuteLogout (see page 5685)	Logs out the current user.
	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5493)	Retrieves the display name string.
	GetDisplayNameValue (see page 5494)	Gets the display name of the end user.
	GetDisplayNameValue (see page 5685)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5494)	Gets the end-user adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5495)	Determines if the user is logged in
	GetLoggedInValue (see page 5495)	Determines if the user is logged in.
	GetLoggedInValue (see page 5686)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 5495)	Gets the action component for the login form.
	GetLogoutAction (see page 5496)	Gets the action component for the logout form.
	GetUserID (see page 5496)	Gets the UserID from this TBaseEndUserAdapter (see page 5485).

	GetUserID (see page 5686)	Returns the user ID of the current user.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 5496)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5485).
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 5497)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5485).
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5497)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5485).
	ImplCanAddActionClass (see page 5497)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5498)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5498)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 5498)	Gets the display name for this TBaseEndUserAdapter (see page 5485).
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 5499)	Returns this TBaseEndUserAdapter (see page 5485).
	ImplGetFieldsList (see page 5499)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 5499)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5499)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetLogoutAction (see page 5500)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5485).
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5500)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5500)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 5501)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5501)	Processes the login for the given user.

	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5501)	Generate the HTML login page.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5502)	Logs in a specified user.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5502)	Generates the login page.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5503)	Gets the user list.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5967)	Logs in a specified user.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	UserID (see page 5687)	Specifies the user ID of the current end user.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5787)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5787)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5787)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5788)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5788)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5788)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5789)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 5785)	Container for adapter actions components.
	Data (see page 5786)	Container for adapter field components.
	LoginPage (see page 5786)	Specifies the name of the page that acts as the login page.

57.1.69.2 TEndUserSessionAdapter Properties

The properties of the TEndUserSessionAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5785)	Container for adapter actions components.
	Data (see page 5786)	Container for adapter field components.
	LoginPage (see page 5786)	Specifies the name of the page that acts as the login page.

57.1.69.2.1 TEndUserSessionAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.69.2.2 TEndUserSessionAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.69.2.3 TEndUserSessionAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

57.1.69.3 TEndUserSessionAdapter Events

The events of the TEndUserSessionAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5787)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5787)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5787)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5788)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5788)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5788)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5789)	Allows implementation of a customized login page.

57.1.69.3.1 TEndUserSessionAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.69.3.2 TEndUserSessionAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.69.3.3 TEndUserSessionAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.69.3.4 TEndUserSessionAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.69.3.5 TEndUserSessionAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.69.3.6 TEndUserSessionAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5492)

ImplHasExecuteRights (see page 5500)

ImplHasModifyRights (see page 5500)

ImplHasViewRights (see page 5501)

57.1.69.3.7 TEndUserSessionAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5501)

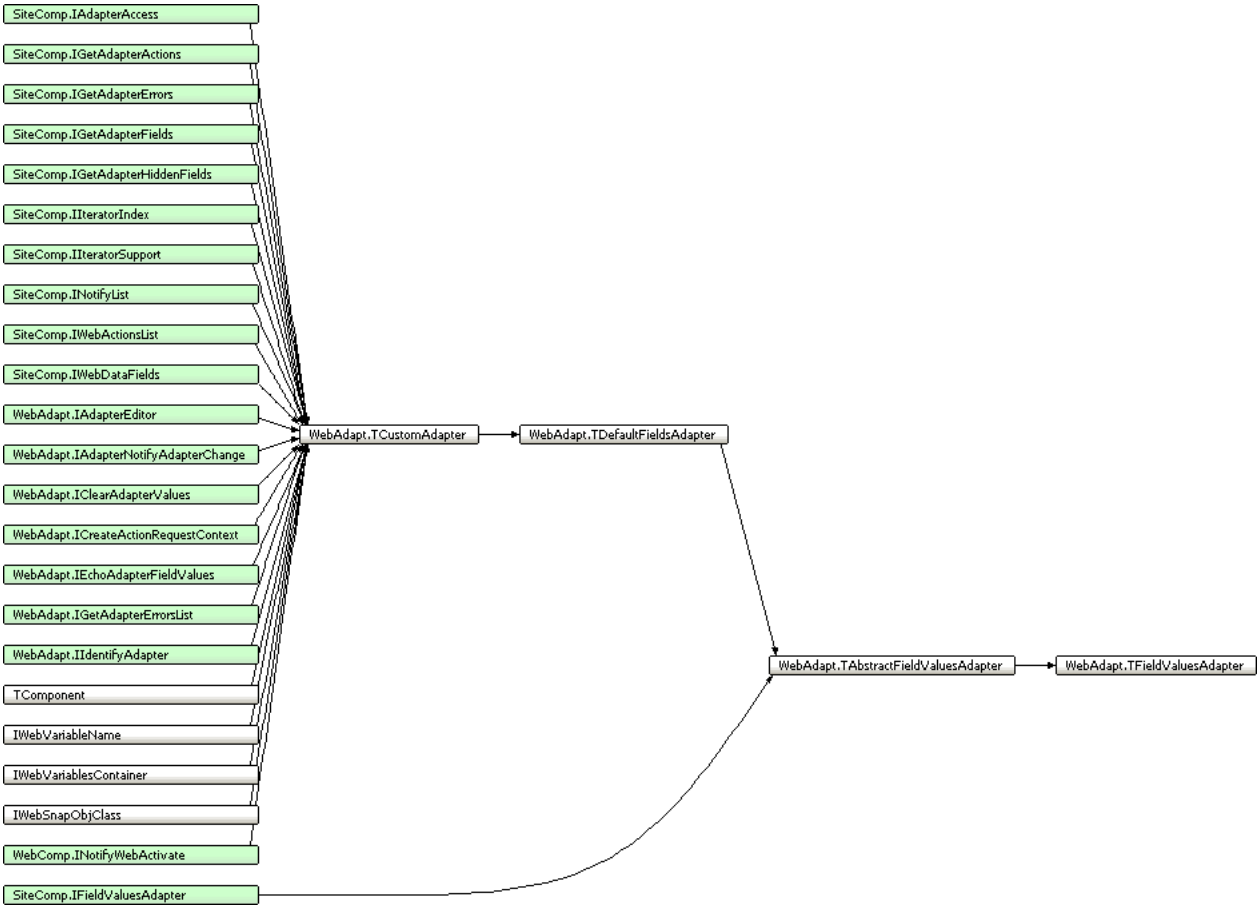
ProduceLoginPage (see page 5502)

CanProduceLoginPage (see page 5492)

57.1.70 TFieldValuesAdapter Class

TFieldValuesAdapter is an adapter that includes a single field value.

Class Hierarchy



Pascal

```
TFieldValuesAdapter = class(TAbstractFieldValuesAdapter);
```

C++

```
class TFieldValuesAdapter : public TAbstractFieldValuesAdapter;
```

File

WebAdapt

Description

Use TFieldValuesAdapter to represent a single field value, which is available as a field with the name 'ValueField'. The single field is represented by an adapter field object that is of type TFieldValuesValueField (see page 5799), or one of its descendants. This adapter field object is supplied to the TFieldValuesAdapter constructor.

The 'ValueField' field can have multiple values, such as when it represents a value in the column of a table. For example, the following server-side script iterates over the values of the 'ValueField' field and adds them to a table:

```
<table border=1>
<%
   var e = new Enumerator(FieldValuesAdapter1)
   for (; !e.atEnd(); e.moveNext())
   {
%>
```

```
  <tr><td><%=e.item().ValueField%></td></tr>
<%
   }
%>
</table>
```

See Also

TStringsValuesList (see page 5843)

TPagedAdapter (see page 5830)

TDataSetValuesList

TFieldValuesValueField (see page 5799)

57.1.70.1 TFieldValuesAdapter Members

The following tables list the members exposed by TFieldValuesAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetField (see page 5175)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetValueFieldClass (see page 5795)	Returns the class of the object that represents this adapter's ValueField field.
	GetValueIndex (see page 5176)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5176)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5796)	Provides the underlying implementation for the EndIterator (see page 5529) method.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetField (see page 5796)	Provides the underlying implementation for the GetField (see page 5175) method.
V	ImplGetFieldsList (see page 5177)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.
V	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetValueIndex (see page 5796)	Provides the underlying implementation for the GetValueIndex (see page 5176) method.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.

	ImplNextIteration (see page 5797)	Provides the underlying implementation for the NextIteration (see page 5551) method.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5797)	Provides the underlying implementation for the StartIterator (see page 5553) method.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	Field (see page 5798)	Indicates the object that represents this adapter's single field.
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.

	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5795)	This is Create, a member of class TFieldValuesAdapter.
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetField (see page 958)	This is GetField, a member of class IFieldValuesAdapter.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetValueIndex (see page 958)	This is GetValueIndex, a member of class IFieldValuesAdapter.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.
R	Actions (see page 5732)	Container for adapter actions components.
R	Data (see page 5558)	Data is the container for adapter field components.
R	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.

	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.70.2 TFieldValuesAdapter.Create Constructor

Pascal

```
constructor Create(AOwner: TComponent; AField: TObject); virtual; reintroduce;
```

C++

```
virtual __fastcall TFieldValuesAdapter(TComponent * AOwner, TObject * AField);
```

Description

This is Create, a member of class TFieldValuesAdapter.

57.1.70.3 TFieldValuesAdapter Methods

The methods of the TFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
	GetValueFieldClass (see page 5795)	Returns the class of the object that represents this adapter's ValueField field.
	ImplEndIterator (see page 5796)	Provides the underlying implementation for the EndIterator (see page 5529) method.
	ImplGetField (see page 5796)	Provides the underlying implementation for the GetField (see page 5175) method.
	ImplGetValueIndex (see page 5796)	Provides the underlying implementation for the GetValueIndex (see page 5176) method.
	ImplNextIteration (see page 5797)	Provides the underlying implementation for the NextIteration (see page 5551) method.
	ImplStartIterator (see page 5797)	Provides the underlying implementation for the StartIterator (see page 5553) method.

57.1.70.3.1 TFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the object that represents this adapter's ValueField field.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field (ValueField) that is the value of the Field (see page 5798) property. This is TFieldValuesValueField (see page 5799).

See Also

Field (see page 5798)

TFieldValuesValueField (see page 5799)

57.1.70.3.2 TFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 5529) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 5529) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values. This provides the underlying support that allows server-side script to iterate through adapter field values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator obtains an IWebGetFieldValues (see page 1034) interface for the adapter field that is the value of the Field (see page 5798) property and calls its EndValueIterator method, passing on the value of the OwnerData parameter.

See Also

EndIterator (see page 5529)

ImplStartIterator (see page 5797)

ImplNextIteration (see page 5797)

EndValueIterator (see page 1035)

Field (see page 5798)

57.1.70.3.3 TFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 5175) method.

Pascal

```
function ImplGetField: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetField();
```

Description

The GetField (see page 5175) method calls ImplGetField to obtain the object that represents this adapter's single field. This method returns the value of the Field (see page 5798) property.

See Also

GetField (see page 5175)

Field (see page 5798)

57.1.70.3.4 TFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 5176) method.

Pascal

```
function ImplGetValueIndex: Integer; override;
```

C++

```
virtual __fastcall int ImplGetValueIndex();
```

Description

The GetValueIndex (see page 5176) method calls ImplGetValueIndex to obtain the index of current value of the ValueField field. The first value has an index of 0, the second an index of 1, and so on.

This method returns the index of the current value, as indicated by the index that the GetIteratorIndex (see page 5537) method returns.

See Also

GetValueIndex (see page 5176)

GetIteratorIndex (see page 5537)

57.1.70.3.5 TFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 5551) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 5551) method calls ImplNextIteration when iterating over field values to determine whether there is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 5551) or to StartIterator (see page 5553).

ImplNextIteration returns true if it moves to the next field value, false if it can't change to a new field value (usually because the field is on its last value).

In TFieldValuesAdapter (see page 5789), ImplNextIteration obtains an IWebGetFieldValues (see page 1034) interface for the adapter field that is the value of the Field (see page 5798) property and calls its NextValueIteration method, passing on the value of the OwnerData parameter.

See Also

NextIteration (see page 5551)

ImplStartIterator (see page 5797)

ImplEndIterator (see page 5796)

NextValueIteration (see page 1036)

57.1.70.3.6 TFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 5553) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 5553) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Depending on the adapter field object, it may be supplied by the caller, or created by the adapter field object inside this method.

ImplStartIterator returns true if it positions the ValueField field on its first value, false if the adapter field can't supply a first value.

In TFieldValuesAdapter (see page 5789), ImplStartIterator obtains an IWebGetFieldValues (see page 1034) interface for the adapter field that is the value of the Field (see page 5798) property and calls its StartValueIterator method, passing on the value of the OwnerData parameter.

See Also

StartIterator (see page 5553)

ImplEndIterator (see page 5796)

ImplNextIteration (see page 5797)

StartValueIterator (see page 1036)

Field (see page 5798)

57.1.70.4 TFieldValuesAdapter Properties

The properties of the TFieldValuesAdapter class are listed here.

Protected Properties

	Name	Description
	Field (see page 5798)	Indicates the object that represents this adapter's single field.

57.1.70.4.1 TFieldValuesAdapter.Field Property

Indicates the object that represents this adapter's single field.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

Description

Read (see page 3727) Field to access the object that represents this adapter's ValueField field. Field is an instance of TFieldValuesValueField (see page 5799) or one of its descendants.

The TFieldValuesAdapter (see page 5789) constructor supplies the value for this property.

See Also

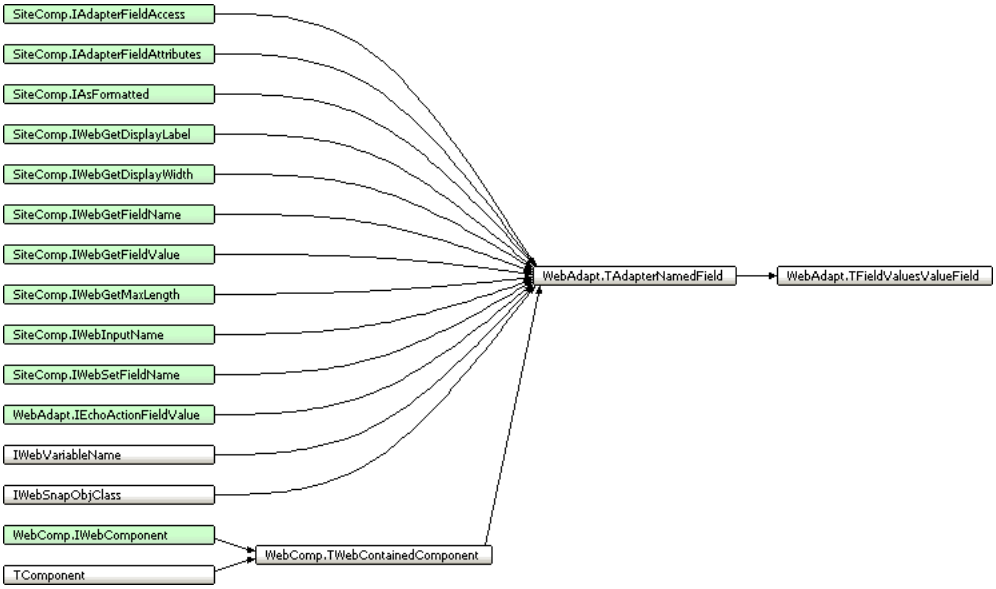
TFieldValuesValueField (see page 5799)

create (see page 5795)

57.1.71 TFieldValuesValueField Class

TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5789).

Class Hierarchy



Pascal

```
TFieldValuesValueField = class(TAdapterNamedField);
```

C++

```
class TFieldValuesValueField : public TAdapterNamedField;
```

File

WebAdapt

Description

The parent adapter, TFieldValuesAdapter (see page 5789), is used to iterate over the values associated with an adapter field. TFieldValuesValueField provides the value and display text of the current item.

See Also

TFieldValuesAdapter (see page 5789)

TBaseAdapterMultiValueField (see page 5475)

57.1.71.1 TFieldValuesValueField Members

The following tables list the members exposed by TFieldValuesValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.

	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5802)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5802)	Returns text used to display a read-only representation of the value associated with this adapter field.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5803)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
R	Adapter (see page 5356)	Returns the parent adapter component.
R	Field (see page 5803)	Returns the adapter field that is using TFieldValueAdapter.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.

R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.71.2 TFieldValuesValueField Methods

The methods of the TFieldValuesValueField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5802)	Returns the name used to access this adapter field using server-side script.
	ImplAsFormatted (see page 5802)	Returns text used to display a read-only representation of the value associated with this adapter field.
	ImplGetValue (see page 5803)	Returns the value associated with this adapter field.

57.1.71.2.1 TFieldValuesValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. This allows server-side script to access this adapter field using the parent adapter's ValueField property.

See Also

ImplGetVariableName (see page 5352)

ImplSetFieldName (see page 5354)

GetDefaultFieldName (see page 5345)

57.1.71.2.2 TFieldValuesValueField.ImplAsFormatted Method

Returns text used to display a read-only representation of the value associated with this adapter field.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted determines whether the adapter field using TFieldValuesAdapter (see page 5789) has a values list. If a values list is available, the values list is used to look up a value name to return. If there is no values list or no value name, the result of ImplGetValue (see page 5803) is returned. Server-side script accesses the result of this function using the DisplayText property of this adapter field. ImplAsFormatted overrides TAdapterNamedField (see page 5339)'s ImplAsFormatted method.

See Also

Value (see page 5357)

ImplAsFormatted (see page 5349)

GetAdapterValuesList (see page 5414)

ImplGetValue (see page 5803)

57.1.71.2.3 TFieldValuesValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetCurrentValue. GetCurrentValue is implemented by the adapter field using TFieldValuesAdapter (see page 5789). Server-side script accesses the result of this function using the Value (see page 5357) property of this adapter field.

See Also

GetCurrentValue (see page 5481)

Value (see page 5357)

57.1.71.3 TFieldValuesValueField Properties

The properties of the TFieldValuesValueField class are listed here.

Protected Properties

	Name	Description
	Field (see page 5803)	Returns the adapter field that is using TFieldValueAdapter.

57.1.71.3.1 TFieldValuesValueField.Field Property

Returns the adapter field that is using TFieldValueAdapter.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

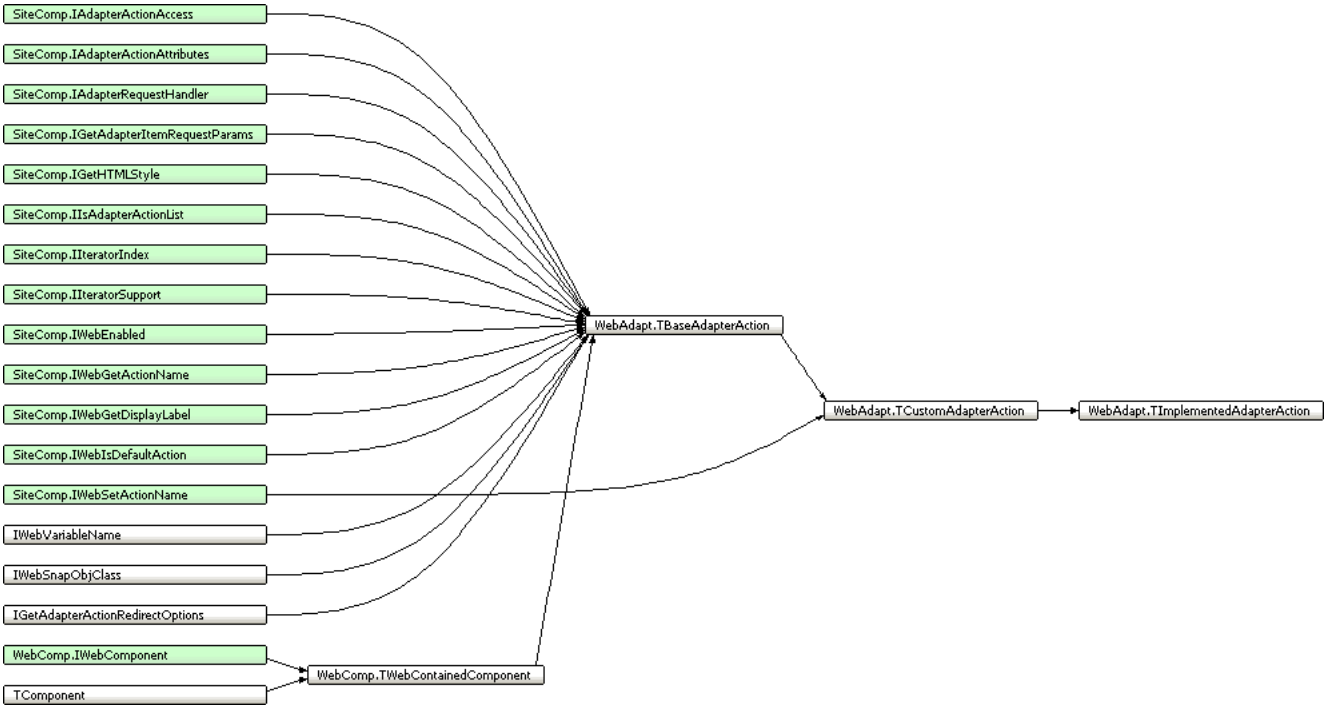
Description

Adapter (see page 5356) fields use the TFieldValuesAdapter (see page 5789) component to iterate over the values associated with the adapter field. The Field property returns the adapter field that is using the TFieldValuesAdapter (see page 5789).

57.1.72 TImplementedAdapterAction Class

TImplementedAdapterAction is a base class for adapter actions.

Class Hierarchy



Pascal

```
TImplementedAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TImplementedAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

Derive from TImplementedAdapterAction to define an adapter action component that works with specific adapter component types. Classes derived from TImplementedAdapterAction can only be added to adapter components that allow the particular class. On the other hand, classes derived from TCustomAdapterAction (see page 5565) (such as TAdapter (see page 5179)) can be added to any adapter component class.

See Also

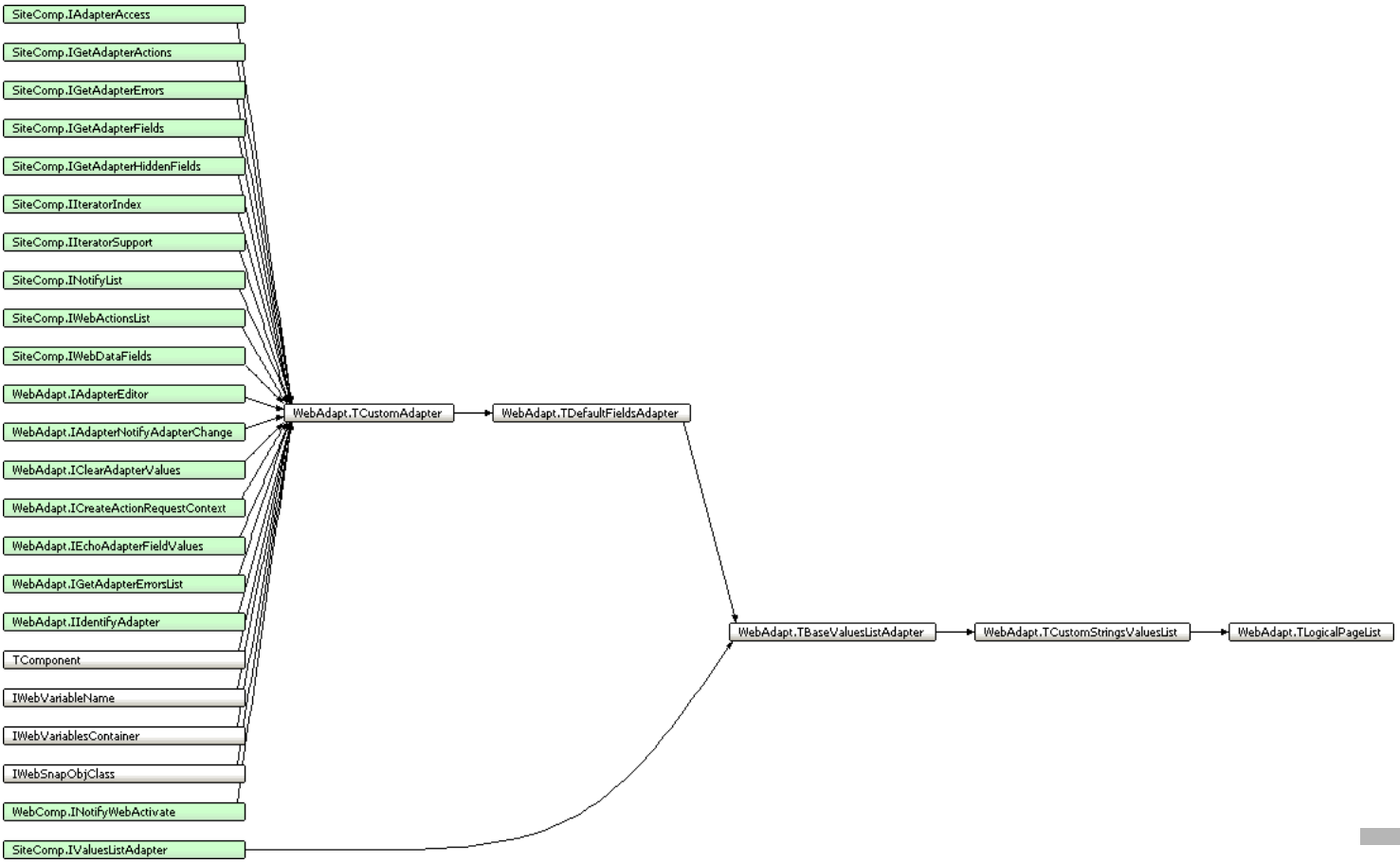
TCustomAdapterAction (see page 5565)

TAdapter (see page 5179)

TAdapterActions (see page 5198)

57.1.73 TLogicalPageList Class

Class Hierarchy



Pascal

```
TLogicalPageList = class(TCustomStringsValuesList);
```

C++

```
class TLogicalPageList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

This is class WebAdapt.TLogicalPageList.

57.1.73.1 TLogicalPageList Members

The following tables list the members exposed by TLogicalPageList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5510)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5510)	Looks up an image for a value.
	GetListName (see page 5511)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5511)	Looks up a name for a value.
	GetListValue (see page 5511)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.

	GetNameFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetValueFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5512)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5512)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 5720)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5513)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 5513)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 5513)	Looks up an image for a value.
	ImplGetListName (see page 5514)	Responsible returning the name of the name/value pair.
	ImplGetListName (see page 5721)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5514)	Looks up a name for a value.
	ImplGetListNameOfValue (see page 5721)	Looks up a name for a value.
	ImplGetListValue (see page 5514)	Responsible for returning the value of the name/value pair.
	ImplGetListValue (see page 5721)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5722)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.

	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5722)	Clears the iteration index.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 5722)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	PrepareStrings (see page 5723)	Customizes the content of the strings list.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5723)	This flag indicates whether the strings list has been prepared or customized.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.

	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 5724)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5810)	Create is the constructor for TCustomStringsValuesList.
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
V	Destroy (see page 5719)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetListImage (see page 1015)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1016)	Returns the image object associated with a specified value.
	GetListName (see page 1016)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1016)	Returns the name associated with a specified value.
	GetListValue (see page 1017)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5558)	Actions is the container for adapter actions components.

	Actions (see page 5732)	Container for adapter actions components.
	Data (see page 5558)	Data is the container for adapter field components.
	Data (see page 5733)	Container for adapter field components.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	Strings (see page 5723)	This string list defines the list of name/value pairs.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

57.1.73.2 TLogicalPageList.Create Constructor

Create is the constructor for TCustomStringsValuesList.

Pascal

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

C++

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

Description

Create creates the strings list.

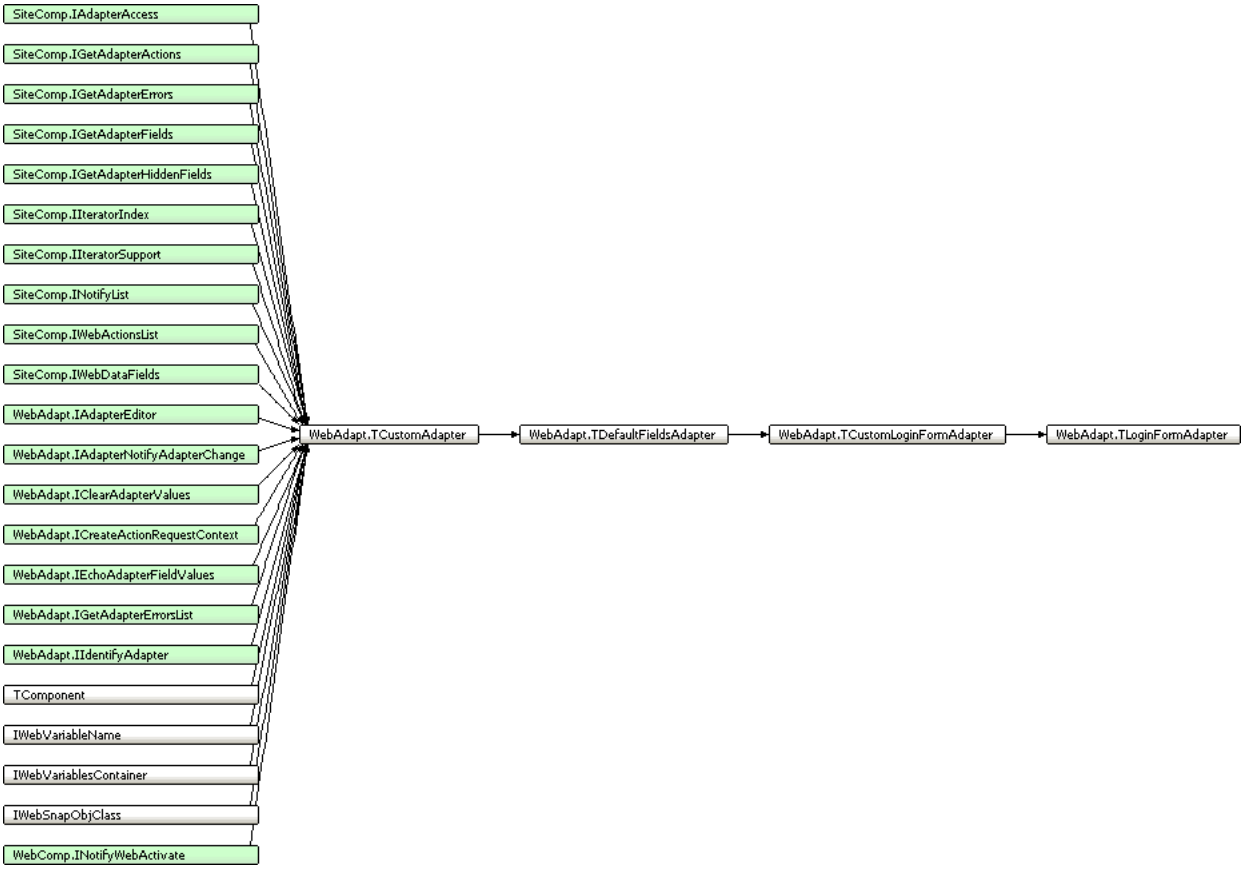
See Also

Strings (see page 5723)

57.1.74 TLoginFormAdapter Class

TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.

Class Hierarchy



Pascal

```
TLoginFormAdapter = class(TCustomLoginFormAdapter);
```

C++

```
class TLoginFormAdapter : public TCustomLoginFormAdapter;
```

File

WebAdapt

Description

Use TLoginFormAdapter to generate a login form. TLoginFormAdapter has default fields for user name, password, and next page. The next page field lets the user select a Web page to see after logging in. TLoginFormAdapter has a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 5750) or TEndUserSessionAdapter (see page 5779), TLoginFormAdapter calls the end user adapter's Login method to log the user in. You can also implement an OnLogin (see page 5820) event handler to provide custom login support.

To build a simple login form

1. Create (see page 5523) a new Web page module that contains a TAdapterPageProducer component.
2. Add a TLoginFormAdapter component to that module.
3. Right click on the TAdapterPageProducer object and display the Web page editor.

4. In the Web page editor, add an adapter form, and to that form add an adapter field group and an adapter command group.
5. Select the field group, and set its Adapter property to the TLoginFormAdapter object.
6. Select the command group, and set its display component to the adapter field group.

You can embellish this simple form as you like. When you are done, connect the form to an end user adapter by setting the end user adapter's LoginPage property to the name of the page module.

To provide complete login support, add TWebUserList, TEndUserSessionAdapter (see page 5779), and TSessionsService components to the web application module to identify user names and passwords and to keep track of end user login state.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

TAdapterPageProducer (see page 1076)

TWebUserList (see page 6759)

TSessionsService (see page 6722)

TAdapterControlGroup (see page 6312)

TAdapterFieldGroup (see page 6367)

57.1.74.1 TLoginFormAdapter Members

The following tables list the members exposed by TLoginFormAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 5693)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5693)	Logs in the user after the user name and password have been validated.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.

	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextPage (see page 5694)	Returns the value of the NextPage (see page 5699) property.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 5694)	Provides the underlying implementation for the CanAddActionClass (see page 5527) method.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5695)	Provides the underlying implementation for the CanAddFieldClass (see page 5527) method.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 5695)	Provides the underlying implementation for the GetActionsList (see page 5533) method.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetFieldsList (see page 5696)	Provides the underlying implementation for the GetFieldsList (see page 5536) method.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.

	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5696)	Provides the underlying implementation for the NotifyDeactivate (see page 5552) method.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	RaiseBlankPassword (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5697)	Raises an EAdapterFieldException (see page 5167) with a message that the supplied user name is blank.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5697)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5698)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5698)	Validates the name that the user supplied to the login form.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	NextPage (see page 5699)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5700)	Indicates the password that the user supplied to the login form.
R	UserName (see page 5700)	Indicates the name that the user supplied to the login form.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5818)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5818)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5819)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5819)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5819)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 5820)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5820)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5821)	Occurs when the login form adapter must validate the name that the user supplied.

Published Properties

	Name	Description
	Actions (see page 5816)	Container for adapter actions components.
	Data (see page 5817)	Container for adapter field components.
	PasswordRequired (see page 5817)	Indicates whether the user must supply a password to log in.

57.1.74.2 TLoginFormAdapter Properties

The properties of the TLoginFormAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5816)	Container for adapter actions components.
	Data (see page 5817)	Container for adapter field components.
	PasswordRequired (see page 5817)	Indicates whether the user must supply a password to log in.

57.1.74.2.1 TLoginFormAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.74.2.2 TLoginFormAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.74.2.3 TLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword event handler does not handle the validation of the password, then the login form adapter checks that the value of Password is not an empty string. If Password is an empty string, the login form adapter raises an EAdapterFieldException exception when it tries to validate the password. If the OnValidatePassword event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's Login action.

See Also

Password (see page 5700)

OnValidatePassword (see page 5702)

TLoginFormAdapterLoginAction (see page 5821)

EAdapterFieldException (see page 5167)

57.1.74.3 TLoginFormAdapter Events

The events of the TLoginFormAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5818)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5818)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5819)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5819)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5819)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 5820)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5820)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5821)	Occurs when the login form adapter must validate the name that the user supplied.

57.1.74.3.1 TLoginFormAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.74.3.2 TLoginFormAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.74.3.3 TLoginFormAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.74.3.4 TLoginFormAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.74.3.5 TLoginFormAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.74.3.6 TLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword and an OnValidateUserName event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's Login method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 5702)

OnValidateUserName (see page 5702)

LogIn (see page 5967)

TWebUserList (see page 6759)

UserName (see page 5700)

57.1.74.3.7 TLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName property.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName event, and then to log the user in using an OnLogin event (and, if the application includes an end user adapter, by calling the end user adapter's Login method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets

the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired property is true.

See Also

OnLogin (see page 5701)

OnValidateUserName (see page 5702)

Password (see page 5700)

UserName (see page 5700)

PasswordRequired (see page 5700)

57.1.74.3.8 TLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword event and before the OnLogin event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException exception if the value of UserName is an empty string. If the OnValidateUserName event handler finds the user name to be invalid, it should raise an exception.

See Also

OnLogin (see page 5701)

OnValidatePassword (see page 5702)

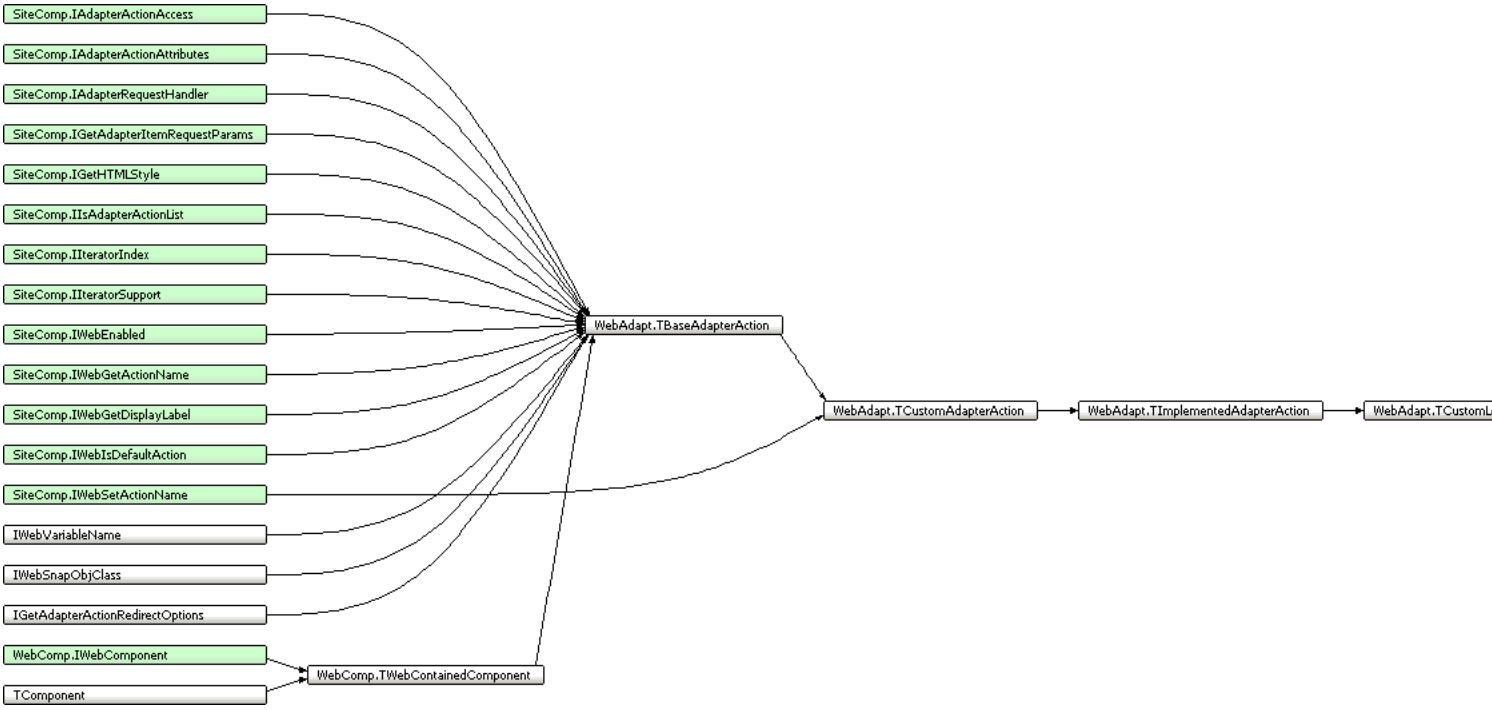
UserName (see page 5700)

EAdapterFieldException (see page 5167)

57.1.75 TLoginFormAdapterLoginAction Class

TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5810).

Class Hierarchy



Pascal

```
TLoginFormAdapterLoginAction = class(TCustomLoginFormAdapterAction);
```

C++

```
class TLoginFormAdapterLoginAction : public TCustomLoginFormAdapterAction;
```

File

WebAdapt

Description

Add TLoginFormAdapterLoginAction to TLoginFormAdapter (see page 5810). The login form typically contains three fields: user name, password, and next page.

See Also

TCustomLoginFormAdapter (see page 5687)

TAdapterNextPageField (see page 5367)

TAdapterUserNameField (see page 5422)

TAdapterPasswordField (see page 5379)

57.1.75.1 TLoginFormAdapterLoginAction Members

The following tables list the members exposed by TLoginFormAdapterLoginAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5440)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5440)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5441)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5441)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5442)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5442)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5443)	Implements IIteratorSupport (see page 996)'s EndIterator method.
	ExecuteActionRequest (see page 5443)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5444)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5444)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5444)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5826)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5445)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5445)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5445)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5446)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5446)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 5446)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5447)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5447)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5448)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5448)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5448)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5449)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5449)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5450)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5450)	Suggests a view mode for this adapter action.
	GetVisible (see page 5450)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5451)	Returns the class type for this adapter field.
	HandleRequest (see page 5451)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5451)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5452)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5452)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5443) method.
V	ImplExecuteActionRequest (see page 5826)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5453)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5570)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5827)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5453)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5454)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5454)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5454)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5455)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5448) method.
V	ImplGetSuccessOptions (see page 5455)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5571)	Indicates whether to perform an HTTP redirect after successfully executing this action.

	ImplGetVariableName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5456)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5456)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5456)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5457)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5459) method.
V	ImplIsDefaultAction (see page 5457)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5458)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5460) method.
	ImplSetActionName (see page 5571)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5458)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5461) method.
V	ImplWebEnabled (see page 5458)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5571)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5459)	Implements IIteratorIndex (see page 994)'s InIterator method.
	IsDefaultAction (see page 5459)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5460)	Implements IIteratorSupport (see page 996)'s NextIteration method.
V	NextPage (see page 5827)	Identifies the name of the next page to display after login.
V	ParentChanged (see page 5460)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5572)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5460)	Set the value of the DisplayLabel (see page 5462) property.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.
	StartIterator (see page 5461)	Implements IIteratorSupport (see page 996)'s StartIterator method.
	WebEnabled (see page 5461)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5462)	Provides access to the parent adapter component.
R	Adapter (see page 5707)	Returns the adapter that acts as a parent to this action.
R	Errors (see page 5463)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5574)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5575)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	CreateRequestContext (see page 955)	Creates an object that describes the adapter action parameters and adapter field values.

	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 966)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 974)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputStyle (see page 974)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 991)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 975)	Returns the preferred mode of an adapter field.
	GetVisible (see page 926)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 955)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 925)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1042)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 997)	Advances to the objects next item.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
	WebEnabled (see page 1026)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1029)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5462)	Identifies the name of this action when using server-side script.
	ActionName (see page 5572)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5463)	Defines the rights required to execute this action.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 991)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5463)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5464)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 995)	Indicates the index of the current item.
	RedirectOptions (see page 5464)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 926)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5828)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5829)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5829)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5830)	Called before the adapter action response is generated.
	OnGetParams (see page 5830)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5828)	Suggests a display label for this adapter action.

57.1.75.2 TLoginFormAdapterLoginAction Methods

The methods of the TLoginFormAdapterLoginAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5826)	Gets the name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 5826)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5827)	Provides the name/value pairs needed to execute this adapter action.
	NextPage (see page 5827)	Identifies the name of the next page to display after login.

57.1.75.2.1 TLoginFormAdapterLoginAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Login".

See Also

GetActionName (see page 5444)

GetDefaultActionName (see page 5445)

57.1.75.2.2 TLoginFormAdapterLoginAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest uses the user name and password passed in the HTTP request to log the user in. Exceptions are raised if the user name or password contain invalid values or if the user name/password is not known. If an exception is raised the exception message is added to the parent adapter's error list and the echo flag is set to true. If the login is successful then ImplExecuteActionRequest causes an HTTP redirect to the page specified in the next page field.

See Also

ImplExecuteActionRequest (see page 5452)

IActionRequest

IActionResponse

ValidatePassword (see page 5698)

ValidateUserName (see page 5698)

ValidateUser (see page 1059)

Errors (see page 5559)

EchoActionFieldValues (see page 5559)

57.1.75.2.3 TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams adds a name/value pair identifying the name of the next page that should be displayed after login. When the login action is executed, the name/value specified by ImplGetAdapterItemRequestParams will be used to generate the HTTP response unless the login form contains a next page field. In that case, the value entered by the end-user will be used to the generate the HTTP response.

See Also

NextPage (see page 5699)

RequestedPageName (see page 5949)

ImplGetAdapterItemRequestParams (see page 5453)

IAdapterItemRequestParams (see page 947)

TAdapterNextPageField (see page 5367)

57.1.75.2.4 TLoginFormAdapterLoginAction.NextPage Method

Identifies the name of the next page to display after login.

Pascal

```
function NextPage: string; virtual;
```

C++

```
virtual __fastcall UnicodeString NextPage();
```

Description

This value is used when generating the login form. NextPage returns the result of TCustomLoginFormAdapter.NextPage.

See Also

NextPage (see page 5699)

57.1.75.3 TLoginFormAdapterLoginAction Properties

The properties of the TLoginFormAdapterLoginAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5828)	Suggests a display label for this adapter action.

57.1.75.3.1 TLoginFormAdapterLoginAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5453)

SetDisplayLabel (see page 5460)

57.1.75.4 TLoginFormAdapterLoginAction Events

The events of the TLoginFormAdapterLoginAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5828)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5829)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5829)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5830)	Called before the adapter action response is generated.
	OnGetParams (see page 5830)	Specifies name/value pairs needed to execute this adapter action.

57.1.75.4.1 TLoginFormAdapterLoginAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5441)

OnAfterExecuteAction (see page 5561)

OnBeforeExecute (see page 5465)

57.1.75.4.2 TLoginFormAdapterLoginAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.75.4.3 TLoginFormAdapterLoginAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5442)

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.75.4.4 TLoginFormAdapterLoginAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5442)

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.75.4.5 TLoginFormAdapterLoginAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

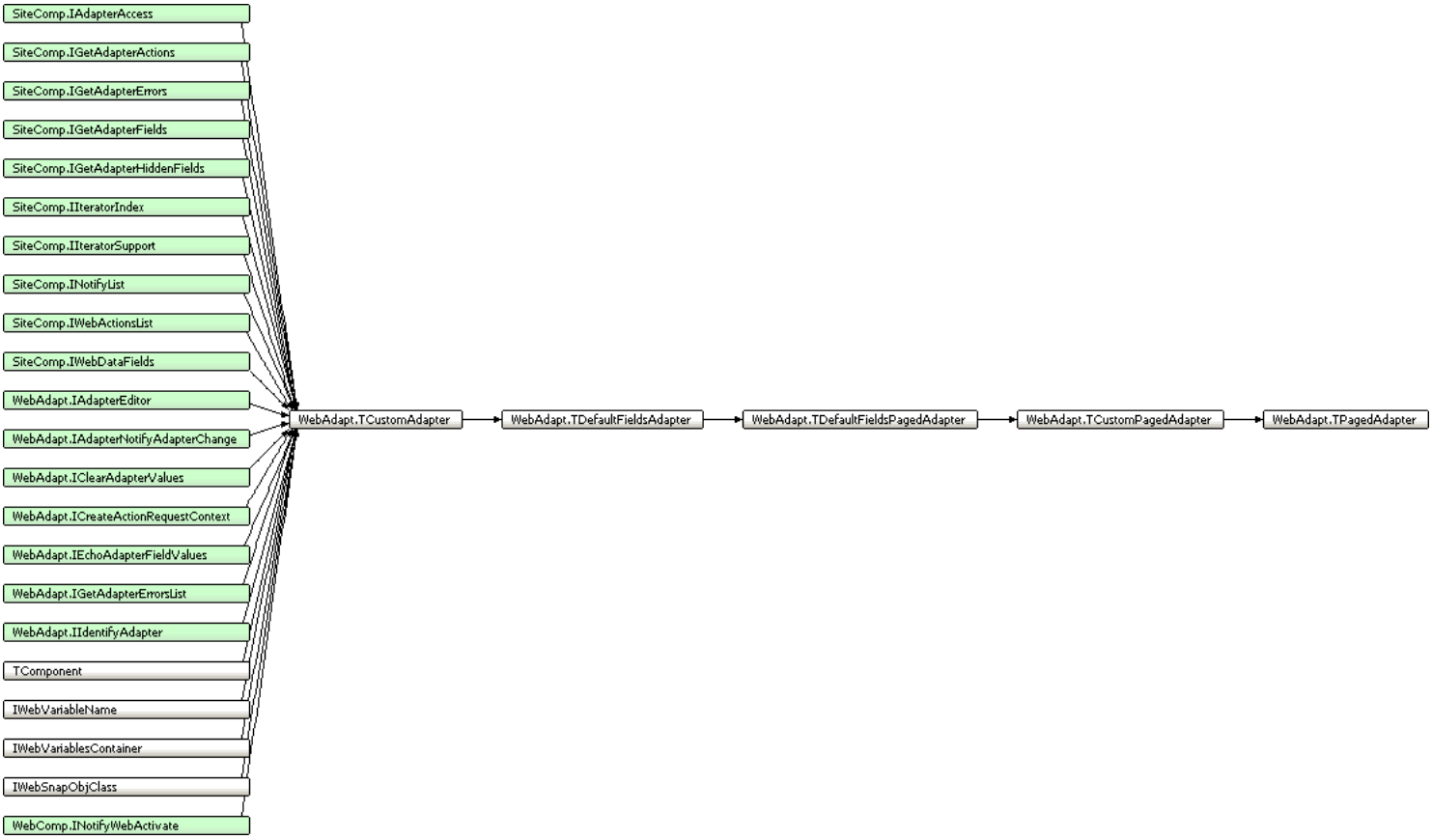
See Also

ImplGetAdapterItemRequestParams (see page 5453)

57.1.76 TPagedAdapter Class

TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.

Class Hierarchy



Pascal

```
TPagedAdapter = class(TCustomPagedAdapter);
```

C++

```
class TPagedAdapter : public TCustomPagedAdapter;
```

File

WebAdapt

Description

Use TPagedAdapter like TAdapter (see page 5179) to define a custom interface to application logic. Set the PageSize (see page 5837) property to indicate the number of rows that a page displays. When the PageSize (see page 5837) is not 0, the TPagedAdapter component automatically adds built-in actions called TAdapterNextPageAction (see page 5358), TAdapterPrevPageAction (see page 5384), and TAdapterGotoPageAction (see page 5280) for moving between pages. To generate HTML buttons for these actions, use TAdapterPageProducer.

The TDataSetAdapter component also supports paging.

57.1.76.1 TPagedAdapter Members

The following tables list the members exposed by TPagedAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultActions (see page 5739)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
V	EncodeActionParams (see page 5740)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 5740)	Indicates whether the record index has gone past the last record in the current page.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
V	ExtractRequestParams (see page 5740)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
V	GetEOF (see page 5741)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetFirstRecord (see page 5741)	Set the record position to the first record.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.

	GetNextRecord (see page 5741)	Increments the record position.
	GetPageCount (see page 5742)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5742)	Returns the total number of records.
	GetRecordIndex (see page 5742)	Returns the index of the current record.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 5743)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 5743)	Implements IteratorSupport's EndIterator (see page 5529) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 5743)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5543)	Returns an action or field.
	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5744)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5744)	Clears the current page variable.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.

	ImplStartIterator (see page 5745)	Implements IteratorSupport.StartIterator (see page 5553) to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	SetPageSize (see page 5745)	Updates the pagesize variable.
	SkipToPage (see page 5746)	Skips records until the current record is within the current page.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.

	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	CurrentPage (see page 5746)	Sets or gets the current page index.
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
R	PageCount (see page 5747)	Calculates the number of pages needed to display all adapter records.
R	RecordCount (see page 5747)	Returns the total number of records.
R	RecordIndex (see page 5748)	Returns the index of the current record.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5838)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5838)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5839)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5839)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5839)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5840)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5840)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5841)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5841)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5841)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5842)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5842)	Implement this method to increment the iteration index.

	OnGetRecordCount (see page 5842)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5842)	Implement this event to identify the zero-based current record.

Published Properties

	Name	Description
	Actions (see page 5836)	Container for adapter actions components.
	Data (see page 5836)	Container for adapter field components.
	ModifyAccess (see page 5837)	This is ModifyAccess, a member of class TPagedAdapter.
	PageSize (see page 5837)	Sets or gets the page size variable.
	ViewAccess (see page 5837)	Identifies the rights required to view this adapter's fields.

57.1.76.2 TPagedAdapter Properties

The properties of the TPagedAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5836)	Container for adapter actions components.
	Data (see page 5836)	Container for adapter field components.
	ModifyAccess (see page 5837)	This is ModifyAccess, a member of class TPagedAdapter.
	PageSize (see page 5837)	Sets or gets the page size variable.
	ViewAccess (see page 5837)	Identifies the rights required to view this adapter's fields.

57.1.76.2.1 TPagedAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.76.2.2 TPagedAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.76.2.3 TPagedAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TPagedAdapter.

57.1.76.2.4 TPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 5358)

TAdapterPrevPageAction (see page 5384)

TAdapterGotoPageAction (see page 5280)

57.1.76.2.5 TPagedAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6759)

HasViewAccess (see page 5539)

implhasviewaccess (see page 5546)

57.1.76.3 TPagedAdapter Events

The events of the TPagedAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5838)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5838)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5839)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5839)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5839)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5840)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5840)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5841)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5841)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5841)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5842)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5842)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5842)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5842)	Implement this event to identify the zero-based current record.

57.1.76.3.1 TPagedAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5465)

OnBeforeExecuteAction (see page 5563)

57.1.76.3.2 TPagedAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5465)

OnBeforeGetActionResponse (see page 5563)

57.1.76.3.3 TPagedAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnBeforeUpdateFields (see page 5564)

Errors (see page 5559)

57.1.76.3.4 TPagedAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5555)

OnBeforeValidateFields (see page 5564)

Errors (see page 5559)

57.1.76.3.5 TPagedAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5465)

OnAfterExecuteAction (see page 5561)

57.1.76.3.6 TPagedAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5466)

OnAfterGetActionResponse (see page 5562)

57.1.76.3.7 TPagedAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5555)

OnAfterUpdateFields (see page 5562)

Errors (see page 5559)

57.1.76.3.8 TPagedAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5555)

OnAfterValidateFields (see page 5563)

Errors (see page 5559)

57.1.76.3.9 TPagedAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5466)

57.1.76.3.10 TPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount event handler will be compared to the result of the OnGetRecordIndex event handler.

57.1.76.3.11 TPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

57.1.76.3.12 TPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

57.1.76.3.13 TPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

57.1.76.3.14 TPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

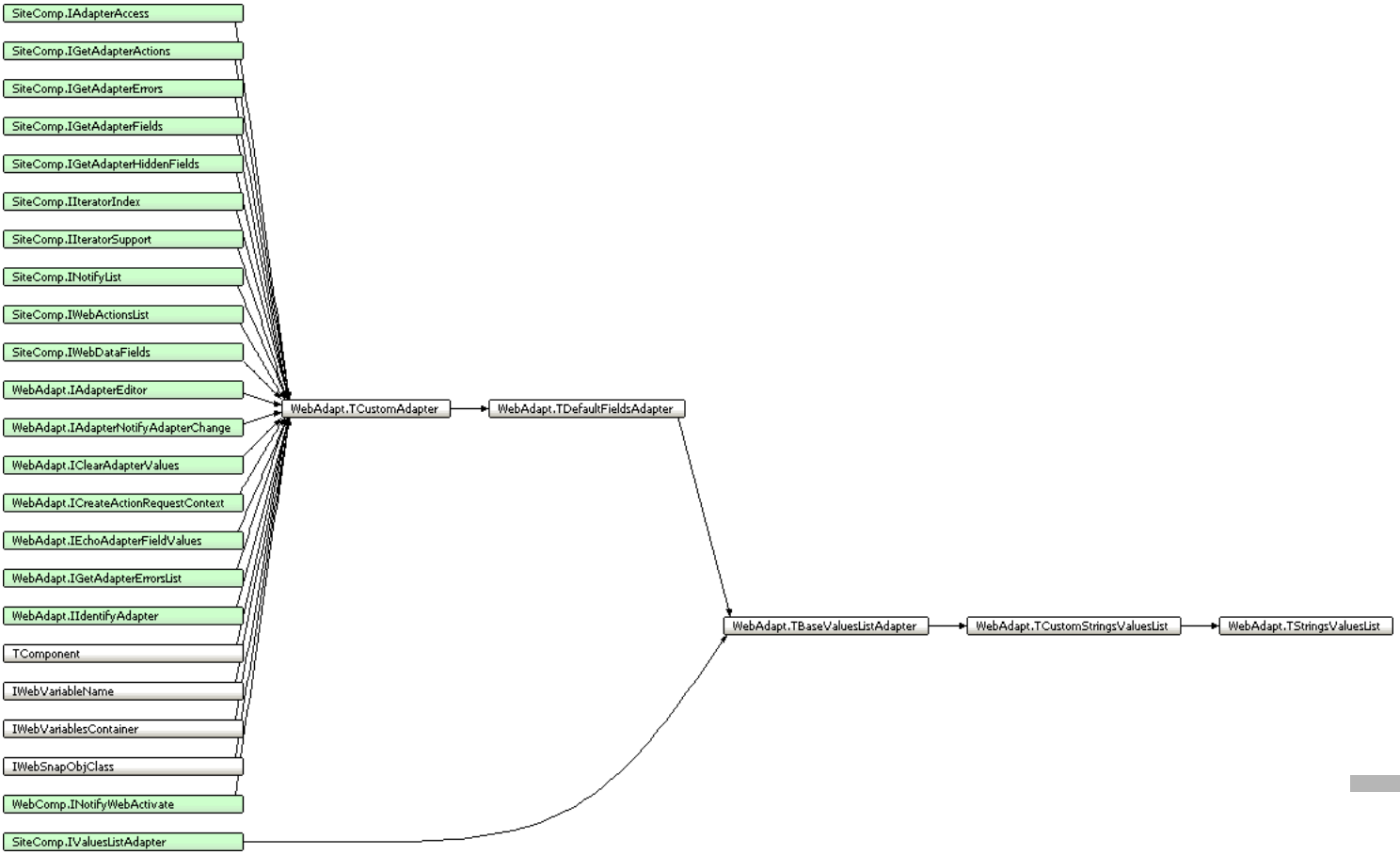
Description

OnGetRecordIndex is called to calculate the current page.

57.1.77 TStringsValuesList Class

TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.

Class Hierarchy



Pascal

```
TStringsValuesList = class(TCustomStringsValuesList);
```

C++

```
class TStringsValuesList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

TStringsValuesList defines a list of possible values for an adapter field by assigning the TStringsValuesList component to the ValuesList property of the adapter field. Typically, the values are used to populate HTML elements, such as a select list or radio buttons, when generating HTML to edit an adapter field. The TAdapterPageProducer will automatically generate appropriate HTML/JavaScript for an editable field when the ValuesList is set.

57.1.77.1 TStringsValuesList Members

The following tables list the members exposed by TStringsValuesList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5526)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5729)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5730)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5526)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5526)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5527)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5527)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5527)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 5528)	Clears adapter state information.
	ClearErrors (see page 5528)	Clears the content of the adapter's Errors (see page 5559) property.
	CreateActionRequestContext (see page 5528)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5529)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5730)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5529)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5529)	Implements IIteratorSupport (see page 996)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5530)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5530)	Returns an action component with a matching action name.
	FindField (see page 5530)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5531)	Finds an action with a matching variable name.
	FindNamedField (see page 5531)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5531)	Finds a field or action with a matching variable name.
	FindVariable (see page 5532)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5532)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5533)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5730)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5533)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5533)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5534)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 5534)	<insert text here>
	GetAdapterFields (see page 5534)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5534)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5535)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5535)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5535)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5731)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5536)	Returns the value of the echo flag.
	GetFieldsList (see page 5536)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5536)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5537)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5510)	Returns an image associated with the current name/value pair.

	GetListImageOfValue (see page 5510)	Looks up an image for a value.
	GetListName (see page 5511)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5511)	Looks up a name for a value.
	GetListValue (see page 5511)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetNameFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5512)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	GetValueFieldClass (see page 5720)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5537)	Returns an action or field.
	GetVariableCount (see page 5538)	Returns the total count of actions and fields.
	GetVariableName (see page 5538)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5538)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 5538)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 5512)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 5539)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5539)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 5539)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 5539)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5540)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5512)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5540)	Clears adapter state information.
V	ImplEndIterator (see page 5541)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the data records of the adapter.
V	ImplEndIterator (see page 5720)	Implements IIteratorSupport (see page 996)'s EndIterator (see page 5529) method to support iterating over the records of the adapter.
V	ImplFindVariable (see page 5541)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5541)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5542)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5542)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5513)	Adds default field components to this adapter.
V	ImplGetFieldsList (see page 5542)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5543)	Implements IIteratorIndex (see page 994)'s GetIteratorIndex (see page 5537) method to support iterating over the data records of the adapter.
V	ImplGetListImage (see page 5513)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 5513)	Looks up an image for a value.
V	ImplGetListName (see page 5514)	Responsible returning the name of the name/value pair.
V	ImplGetListName (see page 5721)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 5514)	Looks up a name for a value.
V	ImplGetListNameOfValue (see page 5721)	Looks up a name for a value.
V	ImplGetListValue (see page 5514)	Responsible for returning the value of the name/value pair.
V	ImplGetListValue (see page 5721)	Responsible for returning the value of the name/value pair.
V	ImplGetVariable (see page 5543)	Returns an action or field.
V	ImplGetVariableCount (see page 5544)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5544)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5544)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5731)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5544)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5731)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5545)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5545)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5546)	This is ImplHasViewAccess, a member of class TCustomAdapter.

	ImplInIterator (see page 5546)	Implements IIteratorIndex (see page 994)'s InIterator (see page 5549) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5546)	Implements IIteratorSupport (see page 996)'s NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5722)	Implements IteratorSupport's NextIteration (see page 5551) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5547)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5547)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5722)	Clears the iteration index.
	ImplSetActions (see page 5547)	Initializes the Actions (see page 5558) property.
	ImplSetActions (see page 5732)	Initializes the Actions (see page 5732) property.
	ImplSetData (see page 5548)	Initializes the Data (see page 5558) property.
	ImplSetData (see page 5732)	Initializes the Data (see page 5733) property.
	ImplSetEchoActionFieldValues (see page 5548)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5548)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 5722)	Implements IIteratorSupport (see page 996)'s StartIterator (see page 5553) method to support iterating over the records of the adapter.
	InIterator (see page 5549)	Implements IIteratorIndex (see page 994)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5549)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5549)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5550)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 5550)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5551)	Implements IIteratorSupport (see page 996)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5551)	Clears references to deleted components.
	NotifyActivate (see page 5551)	Called when the web module is activated.
	NotifyAdapterChange (see page 5551)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5552)	Called when the web module is deactivated.
	PrepareStrings (see page 5723)	Customizes the content of the strings list.
	SetActions (see page 5552)	Initializes the Actions (see page 5558) property.
	SetData (see page 5553)	Initializes the Data (see page 5558) property.
	SetEchoActionFieldValues (see page 5553)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5553)	Implements IIteratorSupport (see page 996)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5554)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5554)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5555)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5555)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5556)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5557)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5557)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5557)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5560)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5723)	This flag indicates whether the strings list has been prepared or customized.
	VisibleActions (see page 5561)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 5561)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5561)	Occurs after an adapter action has executed.

	OnAfterGetActionResponse (see page 5562)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5562)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5563)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5563)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5563)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5564)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5564)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5564)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5565)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5523)	Constructor for TCustomAdapter (see page 5519).
	Create (see page 5719)	Create is the constructor for TCustomStringsValuesList (see page 5714).
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5523)	Destructor for TCustomAdapter (see page 5519).
	Destroy (see page 5719)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 960)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 962)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 963)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 964)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 964)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 995)	Returns the value of the IteratorIndex (see page 995) property.
	GetListImage (see page 1015)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1016)	Returns the image object associated with a specified value.
	GetListName (see page 1016)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1016)	Returns the name associated with a specified value.
	GetListValue (see page 1017)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 924)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 924)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 924)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 995)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.
V	UpdateRecords (see page 5556)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5559)	Modifies the adapter's echo flag.
R	Errors (see page 5559)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5560)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 995)	Indicates the index of the current item.
	ModifyAccess (see page 5560)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 5560)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnPrepareStrings (see page 5849)	Implement OnPrepareStrings to customize the content of the strings list.

Published Properties

	Name	Description
	Actions (see page 5848)	Container for adapter actions components.
	Data (see page 5849)	Container for adapter field components.
	Strings (see page 5849)	This string list defines the list of name/value pairs.

57.1.77.2 TStringsValuesList Properties

The properties of the TStringsValuesList class are listed here.

Published Properties

	Name	Description
	Actions (see page 5848)	Container for adapter actions components.
	Data (see page 5849)	Container for adapter field components.
	Strings (see page 5849)	This string list defines the list of name/value pairs.

57.1.77.2.1 TStringsValuesList.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5220)

57.1.77.2.2 TStringsValuesList.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5226)

57.1.77.2.3 TStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

57.1.77.3 TStringsValuesList Events

The events of the TStringsValuesList class are listed here.

Published Events

	Name	Description
	OnPrepareStrings (see page 5849)	Implement OnPrepareStrings to customize the content of the strings list.

57.1.77.3.1 TStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

TNotifyEvent

PrepareStrings (see page 5723)

57.1.78 TUpdateFileList Class

TUpdateFileList is a container for uploaded files.

Class Hierarchy

Pascal

```
TUpdateFileList = class;
```

C++

```
class TUpdateFileList;
```

File

WebAdapt

Description

TUpdateFileList contains the files that are being uploaded to a particular adapter field. TUpdateFileList is passed as a parameter to event handlers that operate on a list of uploaded files.

See Also

TCustomAdapterFileField (see page 5590)

TCustomAdapterFileField (see page 5590)

57.1.78.1 TUpdateFileList Members

The following tables list the members exposed by TUpdateFileList.

Public Methods

	Name	Description
	Create (see page 5850)	Creates an instance of TUpdateFileList (see page 5850).

Public Properties

	Name	Description
R	Count (see page 5851)	Returns the count of uploaded files.
R	Files (see page 5851)	Returns a particular uploaded file.

57.1.78.2 TUpdateFileList.Create Constructor

Creates an instance of TUpdateFileList (see page 5850).

Pascal

```
constructor Create(AValue: IActionFieldValue);
```

C++

```
__fastcall TUpdateFileList(IActionFieldValue AValue);
```

Description

```
constructor Create(AValue: IActionFieldValueIActionFieldValue);
```

Create initializes an instance of TUpdateFileList (see page 5850) and calls the inherited Create method.

57.1.78.3 TUpdateFileList Properties

The properties of the TUpdateFileList class are listed here.

Public Properties

	Name	Description
	Count (see page 5851)	Returns the count of uploaded files.
	Files (see page 5851)	Returns a particular uploaded file.

57.1.78.3.1 TUpdateFileList.Count Property

Returns the count of uploaded files.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count with the Files (see page 5851) property to loop through the uploaded files.

See Also

Files (see page 5851)

57.1.78.3.2 TUpdateFileList.Files Property

Returns a particular uploaded file.

Pascal

```
property Files [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Files[int I];
```

Description

Use Files with the Count (see page 5851) property to loop through the uploaded files.

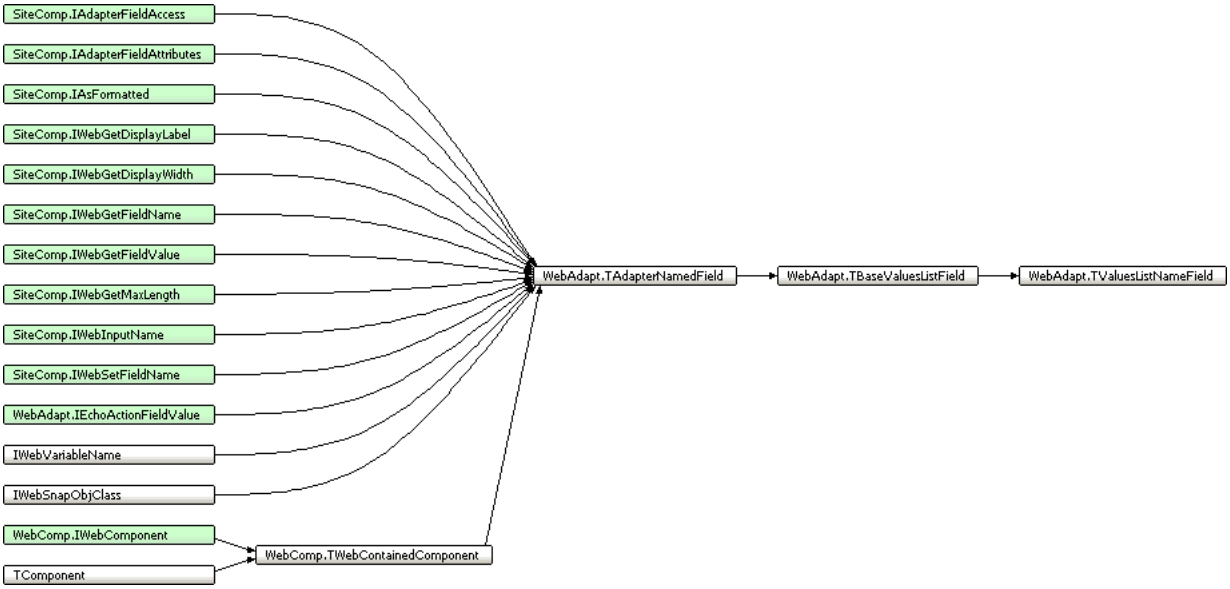
See Also

Count (see page 5851)

57.1.79 TValuesListNameField Class

TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.

Class Hierarchy



Pascal

```
TValuesListNameField = class(TBaseValuesListField);
```

C++

```
class TValuesListNameField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListNameField is an adapter field that returns the name of the current list item.

See Also

TBaseValuesListField (see page 5515)

TStringsValuesList (see page 5843)

TValuesListValueField (see page 5855)

57.1.79.1 TValuesListNameField Members

The following tables list the members exposed by TValuesListNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5518)	Retrieves the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5855)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5855)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
R	Adapter (see page 5356)	Returns the parent adapter component.
R	Adapter (see page 5518)	Retrieves the parent adapter component.
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.

	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.79.2 TValuesListNameField Methods

The methods of the TValuesListNameField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5855)	Returns the name used to access this adapter field using server-side script.
	ImplGetValue (see page 5855)	Returns the value associated with this adapter field.

57.1.79.2.1 TValuesListNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'NameField'. Server-side script can access this adapter field using the NameField property.

57.1.79.2.2 TValuesListNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListName function to retrieve the name of the current list item.

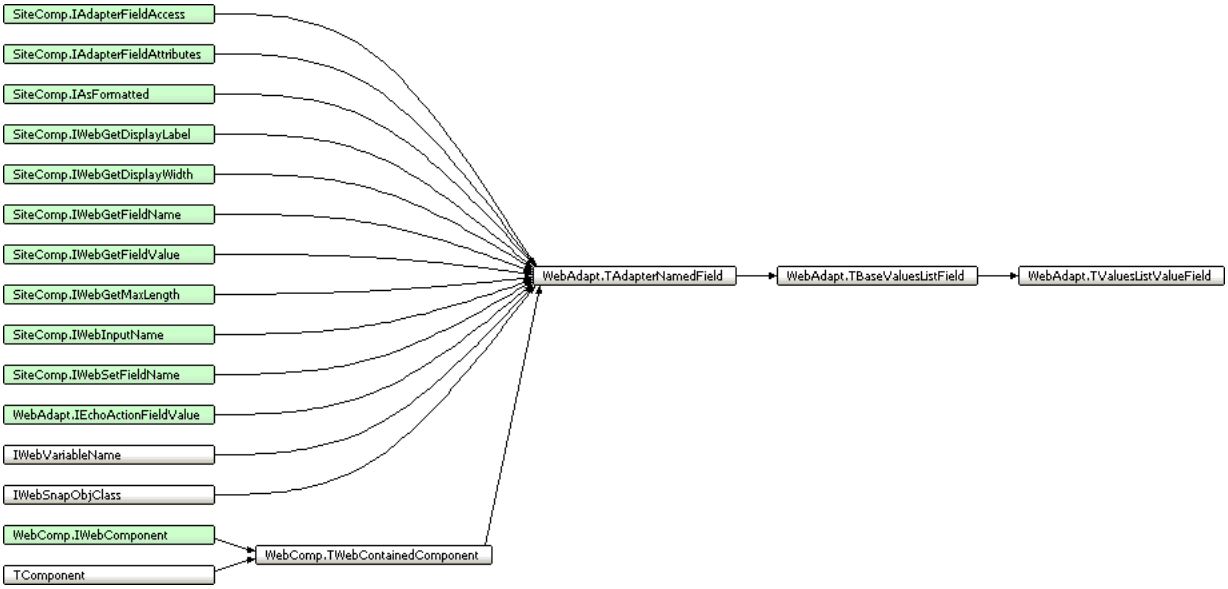
See Also

GetListName (see page 5511)

57.1.80 TValuesListValueField Class

TValuesListValueField is an adapter field used to access the Value (see page 5357) of the current list item of a values list adapter.

Class Hierarchy



Pascal

```
TValuesListValueField = class(TBaseValuesListField);
```

C++

```
class TValuesListValueField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListValueField is an adapter field that returns the value of the current list item.

See Also

TBaseValuesListField (see page 5515)

TCustomStringsValuesList (see page 5714)

TValuesListValueField

57.1.80.1 TValuesListValueField Members

The following tables list the members exposed by TValuesListValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5343)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5343)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5344)	Checks end-user rights to view the adapter field value.

	FullyQualifyInputName (see page 5344)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5344)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5518)	Retrieves the parent adapter component.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5345)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5859)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5345)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5346)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5346)	Returns the value of the echo flag.
	GetInputName (see page 5346)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5347)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5347)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5347)	Gets the value of this adapter field as a Variant (see page 3430).
	GetVariableName (see page 5348)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5348)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5348)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5349)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5349)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5349)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5349)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5350)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5350)	Returns the value of the echo flag.
	ImplGetInputName (see page 5350)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5351)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5859)	Returns the value associated with this adapter field.
	ImplGetVariableName (see page 5352)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5352)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5353)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5353)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5353)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5354)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5354)	Handles movement of the component to a different parent.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5354)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5355)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	Adapter (see page 5356)	Returns the parent adapter component.
	Adapter (see page 5518)	Retrieves the parent adapter component.
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 956)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDisplayLabel (see page 1030)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1031)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetInputName (see page 1041)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1037)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 942)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1033)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 942)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 941)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 941)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5355)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5356)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5356)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5357)	Identifies the name of this field when using server-side script.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1041)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5357)	Identifies the rights required to update this field.
R	Required (see page 943)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1034)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5357)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5358)	Identifies the rights required to view this field.
R	Visible (see page 943)	This is Visible, a member of class IAdapterFieldAttributes.

57.1.80.2 TValuesListValueField Methods

The methods of the TValuesListValueField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5859)	Returns the name used to access this adapter field using server-side script.
	ImplGetValue (see page 5859)	Returns the value associated with this adapter field.

57.1.80.2.1 TValuesListValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. Server-side script can access this adapter field using the ValueField property.

57.1.80.2.2 TValuesListValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListValue function to retrieve the value of the current list item.

See Also

GetListValue (see page 5511)

57.2 Interfaces

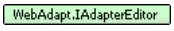
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterEditor (see page 5860)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 5861)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 5862)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 5863)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 5864)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 5865)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 5866)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 5868)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 5869)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 5871)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 5872)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 5873)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 5874)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 5875)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 5875)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 5876)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 5877)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 5878)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 5879)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 5879)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 5880)	This is class WebAdapt.IWebGetFieldPrevValue.

57.2.1 IAdapterEditor Interface

Class Hierarchy



Pascal

```
IAdapterEditor = interface;
```

C++

```
__interface IAdapterEditor;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterEditor.

57.2.1.1 IAdapterEditor Members

The following tables list the members exposed by IAdapterEditor.

Public Methods

	Name	Description
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.

57.2.1.2 IAdapterEditor Methods

The methods of the IAdapterEditor class are listed here.

Public Methods

	Name	Description
	CanAddActionClass (see page 5861)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5861)	This is CanAddFieldClass, a member of class IAdapterEditor.

57.2.1.2.1 IAdapterEditor.CanAddActionClass Method

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddActionClass, a member of class IAdapterEditor.

57.2.1.2.2 IAdapterEditor.CanAddFieldClass Method

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

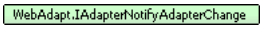
```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddFieldClass, a member of class IAdapterEditor.

57.2.2 IAdapterNotifyAdapterChange Interface

Class Hierarchy



Pascal

```
IAdapterNotifyAdapterChange = interface;
```

C++

```
__interface IAdapterNotifyAdapterChange;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterNotifyAdapterChange.

57.2.2.1 IAdapterNotifyAdapterChange Members

The following tables list the members exposed by IAdapterNotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

57.2.2.2 IAdapterNotifyAdapterChange Methods

The methods of the IAdapterNotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 5862)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

57.2.2.2.1 IAdapterNotifyAdapterChange.NotifyAdapterChange Method

Pascal

```
procedure NotifyAdapterChange;
```

C++

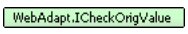
```
__fastcall NotifyAdapterChange();
```

Description

This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

57.2.3 ICheckOrigValue Interface

Class Hierarchy



Pascal

```
ICheckOrigValue = interface;
```

C++

```
__interface ICheckOrigValue;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckOrigValue.

57.2.3.1 ICheckOrigValue Members

The following tables list the members exposed by ICheckOrigValue.

Public Methods

	Name	Description
	CheckOrigValue (see page 5863)	This is CheckOrigValue, a member of class ICheckOrigValue.

57.2.3.2 ICheckOrigValue Methods

The methods of the ICheckOrigValue class are listed here.

Public Methods

	Name	Description
	CheckOrigValue (see page 5863)	This is CheckOrigValue, a member of class ICheckOrigValue.

57.2.3.2.1 ICheckOrigValue.CheckOrigValue Method

Pascal

```
procedure CheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

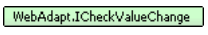
```
__fastcall CheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckOrigValue, a member of class ICheckOrigValue.

57.2.4 ICheckValueChange Interface

Class Hierarchy



Pascal

```
ICheckValueChange = interface;
```

C++

```
__interface ICheckValueChange;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckValueChange.

57.2.4.1 ICheckValueChange Members

The following tables list the members exposed by ICheckValueChange.

Public Methods

	Name	Description
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.

57.2.4.2 ICheckValueChange Methods

The methods of the ICheckValueChange class are listed here.

Public Methods

	Name	Description
	CheckValueChange (see page 5864)	This is CheckValueChange, a member of class ICheckValueChange.

57.2.4.2.1 ICheckValueChange.CheckValueChange Method

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckValueChange, a member of class ICheckValueChange.

57.2.5 IClearAdapterValues Interface

Class Hierarchy

WebAdapt.IClearAdapterValues

Pascal

```
IClearAdapterValues = interface;
```

C++

```
__interface IClearAdapterValues;
```

File

WebAdapt

Description

This is class WebAdapt.IClearAdapterValues.

57.2.5.1 IClearAdapterValues Members

The following tables list the members exposed by IClearAdapterValues.

Public Methods

	Name	Description
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.

57.2.5.2 IClearAdapterValues Methods

The methods of the IClearAdapterValues class are listed here.

Public Methods

	Name	Description
	ClearAdapterValues (see page 5865)	This is ClearAdapterValues, a member of class IClearAdapterValues.

57.2.5.2.1 IClearAdapterValues.ClearAdapterValues Method

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

This is ClearAdapterValues, a member of class IClearAdapterValues.

57.2.6 ICreateActionRequestContext Interface

Class Hierarchy

WebAdapt.ICreateActionRequestContext

Pascal

```
ICreateActionRequestContext = interface;
```

C++

```
__interface ICreateActionRequestContext;
```

File

WebAdapt

Description

This is class WebAdapt.ICreateActionRequestContext.

57.2.6.1 ICreateActionRequestContext Members

The following tables list the members exposed by ICreateActionRequestContext.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

57.2.6.2 ICreateActionRequestContext Methods

The methods of the ICreateActionRequestContext class are listed here.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 5866)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

57.2.6.2.1 ICreateActionRequestContext.CreateActionRequestContext Method

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

57.2.7 IEchoActionFieldValue Interface

Class Hierarchy

WebAdapt.IEchoActionFieldValue

Pascal

```
IEchoActionFieldValue = interface;
```

C++

```
__interface IEchoActionFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoActionFieldValue.

57.2.7.1 IEchoActionFieldValue Members

The following tables list the members exposed by IEchoActionFieldValue.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.7.2 IEchoActionFieldValue Methods

The methods of the IEchoActionFieldValue class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 5867)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 5867)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.7.2.1 IEchoActionFieldValue.GetEchoActionFieldValue Method

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.7.2.2 IEchoActionFieldValue.SetEchoActionFieldValue Method

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.7.3 IEchoActionFieldValue Properties

The properties of the IEchoActionFieldValue class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 5868)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.7.3.1 IEchoActionFieldValue.EchoActionFieldValue Property

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

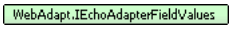
```
__property Boolean EchoActionFieldValue;
```

Description

This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

57.2.8 IEchoAdapterFieldValues Interface

Class Hierarchy



Pascal

```
IEchoAdapterFieldValues = interface;
```

C++

```
__interface IEchoAdapterFieldValues;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoAdapterFieldValues.

57.2.8.1 IEchoAdapterFieldValues Members

The following tables list the members exposed by IEchoAdapterFieldValues.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.8.2 IEchoAdapterFieldValues Methods

The methods of the IEchoAdapterFieldValues class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 5869)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

	SetEchoActionFieldValues (see page 5869)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.8.2.1 IEchoAdapterFieldValues.GetEchoActionFieldValues Method

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.8.2.2 IEchoAdapterFieldValues.SetEchoActionFieldValues Method

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.8.3 IEchoAdapterFieldValues Properties

The properties of the IEchoAdapterFieldValues class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5869)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.8.3.1 IEchoAdapterFieldValues.EchoActionFieldValues Property

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

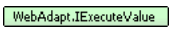
Description

This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

57.2.9 IExecuteValue Interface

IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.

Class Hierarchy



Pascal

```
IExecuteValue = interface;
```

C++

```
__interface IExecuteValue;
```

File

WebAdapt

Description

Adapter fields implement the IExecuteValue interface when they are executable. When a field is executable, HTTP requests can supply field values that are used to execute a command rather than update a value. For example, an HTML form may contain a check box or radio button control that indicates an action the user wants to take. When the form is submitted, the values for the field that underlies the check box or radio buttons indicate the user's choice. When the WebSnap application receives the HTTP request, it uses the IExecuteValue interface to call the field's ExecuteValue (see page 5870) method.

Note: Method declarations that use IExecuteValue use the _di_IExecuteValue type instead. This type is a DelphiInterface (see page 3263) wrapper around the IExecuteValue interface:

Note: typedef System::DelphiInterface (see page 3263)< IExecuteValue > _di_IExecuteValue;

See Also

TCustomAdapter (see page 5519)

57.2.9.1 IExecuteValue Members

The following tables list the members exposed by IExecuteValue.

Public Methods

	Name	Description
	ExecuteValue (see page 5870)	Executes the field based on a value supplied in the current HTTP request.

57.2.9.2 IExecuteValue Methods

The methods of the IExecuteValue class are listed here.

Public Methods

	Name	Description
	ExecuteValue (see page 5870)	Executes the field based on a value supplied in the current HTTP request.

57.2.9.2.1 IExecuteValue.ExecuteValue Method

Executes the field based on a value supplied in the current HTTP request.

Pascal

```
procedure ExecuteValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ExecuteValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

An adapter calls ExecuteValue when responding to an HTTP request that supplies values from executable fields.

AActionRequest is the current request object. ExecuteValue uses this interface to extract the value to use when executing the field.

AFieldIndex is the index of the target field in the list of field values that the request object supplies.

See Also

UpdateFieldsExecuteValues (see page 5554)

57.2.10 IFullyQualifyInputNames Interface

Class Hierarchy

WebAdapt.IFullyQualifyInputNames

Pascal

```
IFullyQualifyInputNames = interface;
```

C++

```
__interface IFullyQualifyInputNames;
```

File

WebAdapt

Description

This is class WebAdapt.IFullyQualifyInputNames.

57.2.10.1 IFullyQualifyInputNames Members

The following tables list the members exposed by IFullyQualifyInputNames.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 5871)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

57.2.10.2 IFullyQualifyInputNames Methods

The methods of the IFullyQualifyInputNames class are listed here.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 5871)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

57.2.10.2.1 IFullyQualifyInputNames.FullyQualifyInputNames Method

Pascal

```
function FullyQualifyInputNames: Boolean;
```

C++

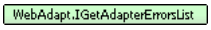
```
__fastcall Boolean FullyQualifyInputNames();
```

Description

This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

57.2.11 IGetAdapterErrorsList Interface

Class Hierarchy



Pascal

```
IGetAdapterErrorsList = interface;
```

C++

```
__interface IGetAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterErrorsList.

57.2.11.1 IGetAdapterErrorsList Members

The following tables list the members exposed by IGetAdapterErrorsList.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

57.2.11.2 IGetAdapterErrorsList Methods

The methods of the IGetAdapterErrorsList class are listed here.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 5872)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

57.2.11.2.1 IGetAdapterErrorsList.GetAdapterErrorsList Method

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

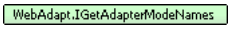
```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

57.2.12 IGetAdapterModeNames Interface

Class Hierarchy



Pascal

```
IGetAdapterModeNames = interface;
```

C++

```
__interface IGetAdapterModeNames;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterModeNames.

57.2.12.1 IGetAdapterModeNames Members

The following tables list the members exposed by IGetAdapterModeNames.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 5873)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

57.2.12.2 IGetAdapterModeNames Methods

The methods of the IGetAdapterModeNames class are listed here.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 5873)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

57.2.12.2.1 IGetAdapterModeNames.GetAdapterModeNames Method

Pascal

```
procedure GetAdapterModeNames(AStrings: TStrings);
```

C++

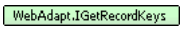
```
__fastcall GetAdapterModeNames(TStrings AStrings);
```

Description

This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

57.2.13 IGetRecordKeys Interface

Class Hierarchy



Pascal

```
IGetRecordKeys = interface;
```

C++

```
__interface IGetRecordKeys;
```

File

WebAdapt

Description

This is class WebAdapt.IGetRecordKeys.

57.2.13.1 IGetRecordKeys Members

The following tables list the members exposed by IGetRecordKeys.

Public Methods

	Name	Description
	GetRecordKeys (see page 5874)	This is GetRecordKeys, a member of class IGetRecordKeys.

57.2.13.2 IGetRecordKeys Methods

The methods of the IGetRecordKeys class are listed here.

Public Methods

	Name	Description
	GetRecordKeys (see page 5874)	This is GetRecordKeys, a member of class IGetRecordKeys.

57.2.13.2.1 IGetRecordKeys.GetRecordKeys Method

Pascal

```
procedure GetRecordKeys(AParams: TStrings; var AFullyQualify: Boolean);
```

C++

```
__fastcall GetRecordKeys(TStrings AParams, Boolean AFullyQualify);
```

Description

This is GetRecordKeys, a member of class IGetRecordKeys.

57.2.14 IIdentifyAdapter Interface

Class Hierarchy

WebAdapt.IIdentifyAdapter

Pascal

```
IIdentifyAdapter = interface;
```

C++

```
__interface IIdentifyAdapter;
```

File

WebAdapt

Description

This is class WebAdapt.IIdentifyAdapter.

57.2.15 ISetSubComponents Interface

Class Hierarchy

WebAdapt.ISetSubComponents

Pascal

```
ISetSubComponents = interface;
```

C++

```
__interface ISetSubComponents;
```

File

WebAdapt

Description

This is class WebAdapt.ISetSubComponents.

57.2.15.1 ISetSubComponents Members

The following tables list the members exposed by ISetSubComponents.

Public Methods

	Name	Description
	SetActions (see page 5876)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 5876)	This is SetData, a member of class ISetSubComponents.

57.2.15.2 ISetSubComponents Methods

The methods of the ISetSubComponents class are listed here.

Public Methods

	Name	Description
	SetActions (see page 5876)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 5876)	This is SetData, a member of class ISetSubComponents.

57.2.15.2.1 ISetSubComponents.SetActions Method

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

This is SetActions, a member of class ISetSubComponents.

57.2.15.2.2 ISetSubComponents.SetData Method

Pascal

```
procedure SetData(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetData(const TAdapterSubComponent Value);
```

Description

This is SetData, a member of class ISetSubComponents.

57.2.16 ISetWebFieldValue Interface

Class Hierarchy

WebAdapt.ISetWebFieldValue

Pascal

```
ISetWebFieldValue = interface;
```

C++

```
__interface ISetWebFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.ISetWebFieldValue.

57.2.16.1 ISetWebFieldValue Members

The following tables list the members exposed by ISetWebFieldValue.

Public Methods

	Name	Description
	SetValue (see page 5877)	This is SetValue, a member of class ISetWebFieldValue.

57.2.16.2 ISetWebFieldValue Methods

The methods of the ISetWebFieldValue class are listed here.

Public Methods

	Name	Description
	SetValue (see page 5877)	This is SetValue, a member of class ISetWebFieldValue.

57.2.16.2.1 ISetWebFieldValue.SetValue Method

Pascal

```
procedure SetValue(const AValue: Variant);
```

C++

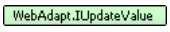
```
__fastcall SetValue(const Variant AValue);
```

Description

This is SetValue, a member of class ISetWebFieldValue.

57.2.17 IUpdateValue Interface

Class Hierarchy



Pascal

```
IUpdateValue = interface;
```

C++

```
__interface IUpdateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IUpdateValue.

57.2.17.1 IUpdateValue Members

The following tables list the members exposed by IUpdateValue.

Public Methods

	Name	Description
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.

57.2.17.2 IUpdateValue Methods

The methods of the IUpdateValue class are listed here.

Public Methods

	Name	Description
	UpdateValue (see page 5878)	This is UpdateValue, a member of class IUpdateValue.

57.2.17.2.1 IUpdateValue.UpdateValue Method

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

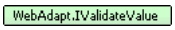
```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is UpdateValue, a member of class IUpdateValue.

57.2.18 IValidateValue Interface

Class Hierarchy



Pascal

```
IValidateValue = interface;
```

C++

```
__interface IValidateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IValidateValue.

57.2.18.1 IValidateValue Members

The following tables list the members exposed by IValidateValue.

Public Methods

	Name	Description
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

57.2.18.2 IValidateValue Methods

The methods of the IValidateValue class are listed here.

Public Methods

	Name	Description
	ValidateValue (see page 5879)	This is ValidateValue, a member of class IValidateValue.

57.2.18.2.1 IValidateValue.ValidateValue Method

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

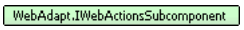
```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is ValidateValue, a member of class IValidateValue.

57.2.19 IWebActionsSubcomponent Interface

Class Hierarchy



Pascal

```
IWebActionsSubcomponent = interface;
```

C++

```
__interface IWebActionsSubcomponent;
```

File

WebAdapt

Description

This is class WebAdapt.IWebActionsSubcomponent.

57.2.20 IWebFieldsSubcomponent Interface

Class Hierarchy

Pascal

```
IWebFieldsSubcomponent = interface;
```

C++

```
__interface IWebFieldsSubcomponent;
```

File

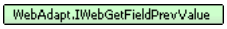
WebAdapt

Description

This is class WebAdapt.IWebFieldsSubcomponent.

57.2.21 IWebGetFieldPrevValue Interface

Class Hierarchy



Pascal

```
IWebGetFieldPrevValue = interface;
```

C++

```
__interface IWebGetFieldPrevValue;
```

File

WebAdapt

Description

This is class WebAdapt.IWebGetFieldPrevValue.

57.2.21.1 IWebGetFieldPrevValue Members

The following tables list the members exposed by IWebGetFieldPrevValue.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 5881)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 5881)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

Public Properties

	Name	Description
R	NeedPrevValue (see page 5881)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
R	PrevValue (see page 5881)	This is PrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.2 IWebGetFieldPrevValue Methods

The methods of the IWebGetFieldPrevValue class are listed here.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 5881)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 5881)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.2.1 IWebGetFieldPrevValue.GetNeedPrevValue Method

Pascal

```
function GetNeedPrevValue: Boolean;
```

C++

```
__fastcall Boolean GetNeedPrevValue();
```

Description

This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.2.2 IWebGetFieldPrevValue.GetPrevValue Method

Pascal

```
function GetPrevValue: Variant;
```

C++

```
__fastcall Variant GetPrevValue();
```

Description

This is GetPrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.3 IWebGetFieldPrevValue Properties

The properties of the IWebGetFieldPrevValue class are listed here.

Public Properties

	Name	Description
	NeedPrevValue (see page 5881)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
	PrevValue (see page 5881)	This is PrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.3.1 IWebGetFieldPrevValue.NeedPrevValue Property

Pascal

```
property NeedPrevValue: Boolean;
```

C++

```
__property Boolean NeedPrevValue;
```

Description

This is NeedPrevValue, a member of class IWebGetFieldPrevValue.

57.2.21.3.2 IWebGetFieldPrevValue.PrevValue Property

Pascal

```
property PrevValue: Variant;
```

C++

```
__property Variant PrevValue;
```

Description

This is PrevValue, a member of class IWebGetFieldPrevValue.

57.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddDefaultActions (see page 5883)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 5883)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 5884)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 5884)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 5884)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 5885)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 5885)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 5885)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 5886)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 5886)	This is function WebAdapt.RaiseMultipleValuesException.

57.3.1 WebAdapt.AddDefaultActions Function

Pascal

```
procedure AddDefaultActions(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultActions(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultActions.

57.3.2 WebAdapt.AddDefaultFields Function

Pascal

```
procedure AddDefaultFields(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultFields(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultFields.

57.3.3 WebAdapt.FindAdapterAction Function

Pascal

```
function FindAdapterAction(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterAction(TWebComponentList AWebComponents, const UnicodeString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterAction.

57.3.4 WebAdapt.FindAdapterField Function

Pascal

```
function FindAdapterField(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterField(TWebComponentList AWebComponents, const UnicodeString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterField.

57.3.5 WebAdapt.FindAdapterInParent Function

Pascal

```
function FindAdapterInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindAdapterInParent(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterInParent.

57.3.6 WebAdapt.MakeAdapterRequestIdentifier Function

Pascal

```
function MakeAdapterRequestIdentifier(AComponent: TComponent): string;
```

C++

```
UnicodeString MakeAdapterRequestIdentifier(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.MakeAdapterRequestIdentifier.

57.3.7 WebAdapt.MakeValidAdapterFieldName Function

Pascal

```
function MakeValidAdapterFieldName(const AFieldName: string): string;
```

C++

```
UnicodeString MakeValidAdapterFieldName(const UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.MakeValidAdapterFieldName.

57.3.8 WebAdapt.RaiseFileUploadNotSupported Function

Pascal

```
procedure RaiseFileUploadNotSupported(AFieldName: string);
```

C++

```
RaiseFileUploadNotSupported(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseFileUploadNotSupported.

57.3.9 WebAdapt.RaiseMultipleFilesException Function

Pascal

```
procedure RaiseMultipleFilesException(AFieldName: string);
```

C++

```
RaiseMultipleFilesException(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleFilesException.

57.3.10 WebAdapt.RaiseMultipleValuesException Function

Pascal

```
procedure RaiseMultipleValuesException(AFieldName: string);
```

C++

```
RaiseMultipleValuesException(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleValuesException.

57.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterHiddenFieldOption (see page 5887)	This is record WebAdapt.TAdapterHiddenFieldOption.
	TIteratorMethod (see page 5887)	This is record WebAdapt.TIteratorMethod.

57.4.1 WebAdapt.TAdapterHiddenFieldOption Enumeration

Pascal

```
TAdapterHiddenFieldOption = (
  hoOrigValues,
  hoMode,
  hoRecordKeys
);
```

C++

```
enum TAdapterHiddenFieldOption {
  hoOrigValues,
  hoMode,
  hoRecordKeys
};
```

File

WebAdapt

Description

This is record WebAdapt.TAdapterHiddenFieldOption.

57.4.2 WebAdapt.TIteratorMethod Enumeration

Pascal

```
TIteratorMethod = (
  itStart,
  itNext,
  itEnd
);
```

C++

```
enum TIteratorMethod {
  itStart,
  itNext,
  itEnd
};
```

File

WebAdapt

Description

This is record WebAdapt.TIteratorMethod.

57.5 Types

The following table lists types in this documentation.

Types

Name	Description
IActionFieldValue (see page 5890)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 5890)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 5890)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 5891)	This is type WebAdapt.TActionExecuteEvent.
TActionExecuteHandledEvent (see page 5891)	This is type WebAdapt.TActionExecuteHandledEvent.
TActionGetEnabledEvent (see page 5891)	This is type WebAdapt.TActionGetEnabledEvent.
TActionGetParamsEvent (see page 5892)	This is type WebAdapt.TActionGetParamsEvent.
TActionRequestFieldsEvent (see page 5892)	This is type WebAdapt.TActionRequestFieldsEvent.
TActionRequestFieldsHandledEvent (see page 5892)	This is type WebAdapt.TActionRequestFieldsHandledEvent.
TAdapterActionExecuteEvent (see page 5893)	This is type WebAdapt.TAdapterActionExecuteEvent.
TAdapterActionExecuteHandledEvent (see page 5893)	This is type WebAdapt.TAdapterActionExecuteHandledEvent.
TAdapterActionGetParamsEvent (see page 5893)	This is type WebAdapt.TAdapterActionGetParamsEvent.
TAdapterActionsClass (see page 5894)	This is type WebAdapt.TAdapterActionsClass.
TAdapterFieldBooleanValueEvent (see page 5894)	This is type WebAdapt.TAdapterFieldBooleanValueEvent.
TAdapterFieldFileEvent (see page 5894)	This is type WebAdapt.TAdapterFieldFileEvent.
TAdapterFieldFileListEvent (see page 5895)	This is type WebAdapt.TAdapterFieldFileListEvent.
TAdapterFieldGetBooleanEvent (see page 5895)	This is type WebAdapt.TAdapterFieldGetBooleanEvent.
TAdapterFieldGetMultiValueEvent (see page 5895)	This is type WebAdapt.TAdapterFieldGetMultiValueEvent.
TAdapterFieldGetMultiValueStringEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.
TAdapterFieldGetStringEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetStringEvent.
TAdapterFieldGetValueCountEvent (see page 5896)	This is type WebAdapt.TAdapterFieldGetValueCountEvent.
TAdapterFieldGetValueEvent (see page 5897)	This is type WebAdapt.TAdapterFieldGetValueEvent.
TAdapterFieldHasValueEvent (see page 5897)	This is type WebAdapt.TAdapterFieldHasValueEvent.
TAdapterFieldsClass (see page 5897)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 5898)	This is type WebAdapt.TAdapterFieldStringEvent.
TAdapterFieldStringsEvent (see page 5898)	This is type WebAdapt.TAdapterFieldStringsEvent.
TAdapterFieldValidateValueEvent (see page 5898)	This is type WebAdapt.TAdapterFieldValidateValueEvent.
TAdapterFieldValueEvent (see page 5899)	This is type WebAdapt.TAdapterFieldValueEvent.
TAdapterHiddenFieldOptions (see page 5899)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 5899)	This is type WebAdapt.TEndUserGetNameEvent.
TEndUserGetUserIDEvent (see page 5900)	This is type WebAdapt.TEndUserGetUserIDEvent.
TEndUserHandledEvent (see page 5900)	This is type WebAdapt.TEndUserHandledEvent.
TEndUserIsLoggedInEvent (see page 5900)	This is type WebAdapt.TEndUserIsLoggedInEvent.
TEndUserRightsEvent (see page 5901)	This is type WebAdapt.TEndUserRightsEvent.
TEndUserUserIDEvent (see page 5901)	This is type WebAdapt.TEndUserUserIDEvent.
TGetEOFEvent (see page 5901)	This is type WebAdapt.TGetEOFEvent.
TGetRecordCountEvent (see page 5902)	This is type WebAdapt.TGetRecordCountEvent.
TGetRecordEvent (see page 5902)	This is type WebAdapt.TGetRecordEvent.
TGetRecordIndexEvent (see page 5902)	This is type WebAdapt.TGetRecordIndexEvent.
THandledEvent (see page 5903)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 5903)	This is type WebAdapt.TImageFieldGetHREFEvent.
TImageFieldGetImageEvent (see page 5903)	This is type WebAdapt.TImageFieldGetImageEvent.
TImageFieldGetParamsEvent (see page 5904)	This is type WebAdapt.TImageFieldGetParamsEvent.
TIterateRecordsEvent (see page 5904)	This is type WebAdapt.TIterateRecordsEvent.

TLoginUserIDEvent (see page 5904)	This is type WebAdapt.TLoginUserIDEvent.
TObjectList (see page 5905)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 5905)	This is type WebAdapt.TValidateAdapterFieldEvent.
TWebFormGetPageNameEvent (see page 5905)	This is type WebAdapt.TWebFormGetPageNameEvent.

57.5.1 WebAdapt.IActionFieldValue Type

Pascal

```
IActionFieldValue = AdaptReq.IActionFieldValue;
```

C++

```
AdaptReq.IActionFieldValue IActionFieldValue;
```

File

WebAdapt

Description

This is type WebAdapt.IActionFieldValue.

57.5.2 WebAdapt.IActionRequest Type

Pascal

```
IActionRequest = AdaptReq.IActionRequest;
```

C++

```
AdaptReq.IActionRequest IActionRequest;
```

File

WebAdapt

Description

This is type WebAdapt.IActionRequest.

57.5.3 WebAdapt.TAbstractWebRequestFile Type

Pascal

```
TAbstractWebRequestFile = HTTPApp.TAbstractWebRequestFile;
```

C++

```
HTTPApp.TAbstractWebRequestFile TAbstractWebRequestFile;
```

File

WebAdapt

Description

This is type WebAdapt.TAbstractWebRequestFile.

57.5.4 WebAdapt.TActionExecuteEvent Type

Pascal

```
TActionExecuteEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionExecuteEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionExecuteEvent.

57.5.5 WebAdapt.TActionExecuteHandledEvent Type

Pascal

```
TActionExecuteHandledEvent = procedure (Sender: TObject; Params: TStrings; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var Handled: Boolean) ( TActionExecuteHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionExecuteHandledEvent.

57.5.6 WebAdapt.TActionGetEnabledEvent Type

Pascal

```
TActionGetEnabledEvent = procedure (Sender: TObject; var Enabled: Boolean) of object;
```

C++

```
(Sender: TObject; var Enabled: Boolean) ( TActionGetEnabledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionGetEnabledEvent.

57.5.7 WebAdapt.TActionGetParamsEvent Type

Pascal

```
TActionGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionGetParamsEvent.

57.5.8 WebAdapt.TActionRequestFieldsEvent Type

Pascal

```
TActionRequestFieldsEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList) (
TActionRequestFieldsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionRequestFieldsEvent.

57.5.9 WebAdapt.TActionRequestFieldsHandledEvent Type

Pascal

```
TActionRequestFieldsHandledEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList; var Handled:
Boolean) ( TActionRequestFieldsHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionRequestFieldsHandledEvent.

57.5.10 WebAdapt.TAdapterActionExecuteEvent Type

Pascal

```
TAdapterActionExecuteEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionExecuteEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionExecuteEvent.

57.5.11 WebAdapt.TAdapterActionExecuteHandledEvent Type

Pascal

```
TAdapterActionExecuteHandledEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) ( TAdapterActionExecuteHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionExecuteHandledEvent.

57.5.12 WebAdapt.TAdapterActionGetParamsEvent Type

Pascal

```
TAdapterActionGetParamsEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionGetParamsEvent.

57.5.13 WebAdapt.TAdapterActionsClass Type

Pascal

```
TAdapterActionsClass = class of TAdapterActions;
```

C++

```
TAdapterActions TAdapterActionsClass;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionsClass.

57.5.14 WebAdapt.TAdapterFieldBooleanValueEvent Type

Pascal

```
TAdapterFieldBooleanValueEvent = procedure (Sender: TObject; Value: Boolean) of object;
```

C++

```
(Sender: TObject; Value: Boolean) ( TAdapterFieldBooleanValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldBooleanValueEvent.

57.5.15 WebAdapt.TAdapterFieldFileEvent Type

Pascal

```
TAdapterFieldFileEvent = procedure (Sender: TObject; AFile: TAbstractWebRequestFile) of object;
```

C++

```
(Sender: TObject; AFile: TAbstractWebRequestFile) ( TAdapterFieldFileEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldFileEvent.

57.5.16 WebAdapt.TAdapterFieldFileListEvent Type

Pascal

```
TAdapterFieldFileListEvent = procedure (Sender: TObject; Files: TUpdateFileList) of object;
```

C++

```
(Sender: TObject; Files: TUpdateFileList) ( TAdapterFieldFileListEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldFileListEvent.

57.5.17 WebAdapt.TAdapterFieldGetBooleanEvent Type

Pascal

```
TAdapterFieldGetBooleanEvent = procedure (Sender: TObject; var Value: Boolean) of object;
```

C++

```
(Sender: TObject; var Value: Boolean) ( TAdapterFieldGetBooleanEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetBooleanEvent.

57.5.18 WebAdapt.TAdapterFieldGetMultiValueEvent Type

Pascal

```
TAdapterFieldGetMultiValueEvent = procedure (Sender: TObject; Index: Integer; var Value:
Variant) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: Variant) ( TAdapterFieldGetMultiValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetMultiValueEvent.

57.5.19 WebAdapt.TAdapterFieldGetMultiValueStringEvent Type

Pascal

```
TAdapterFieldGetMultiValueStringEvent = procedure (Sender: TObject; Index: Integer; var Value:
string) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: string) (
TAdapterFieldGetMultiValueStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.

57.5.20 WebAdapt.TAdapterFieldGetStringEvent Type

Pascal

```
TAdapterFieldGetStringEvent = procedure (Sender: TObject; var Value: string) of object;
```

C++

```
(Sender: TObject; var Value: string) ( TAdapterFieldGetStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetStringEvent.

57.5.21 WebAdapt.TAdapterFieldGetValueCountEvent Type

Pascal

```
TAdapterFieldGetValueCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TAdapterFieldGetValueCountEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetValueCountEvent.

57.5.22 WebAdapt.TAdapterFieldGetValueEvent Type

Pascal

```
TAdapterFieldGetValueEvent = procedure (Sender: TObject; var Value: Variant) of object;
```

C++

```
(Sender: TObject; var Value: Variant) ( TAdapterFieldGetValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetValueEvent.

57.5.23 WebAdapt.TAdapterFieldHasValueEvent Type

Pascal

```
TAdapterFieldHasValueEvent = procedure (Sender: TObject; const Value: Variant; var HasValue:
Boolean) of object;
```

C++

```
(Sender: TObject; const Value: Variant; var HasValue: Boolean) ( TAdapterFieldHasValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldHasValueEvent.

57.5.24 WebAdapt.TAdapterFieldsClass Type

Pascal

```
TAdapterFieldsClass = class of TAdapterFields;
```

C++

```
TAdapterFields TAdapterFieldsClass;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldsClass.

57.5.25 WebAdapt.TAdapterFieldStringEvent Type

Pascal

```
TAdapterFieldStringEvent = procedure (Sender: TObject; Value: string) of object;
```

C++

```
(Sender: TObject; Value: string) ( TAdapterFieldStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldStringEvent.

57.5.26 WebAdapt.TAdapterFieldStringsEvent Type

Pascal

```
TAdapterFieldStringsEvent = procedure (Sender: TObject; Values: TStrings) of object;
```

C++

```
(Sender: TObject; Values: TStrings) ( TAdapterFieldStringsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldStringsEvent.

57.5.27 WebAdapt.TAdapterFieldValidateValueEvent Type

Pascal

```
TAdapterFieldValidateValueEvent = procedure (Sender: TObject; AValue: IActionFieldValue; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; AValue: IActionFieldValue; var Handled: Boolean) (
TAdapterFieldValidateValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldValidateValueEvent.

57.5.28 WebAdapt.TAdapterFieldValueEvent Type

Pascal

```
TAdapterFieldValueEvent = procedure (Sender: TObject; Value: Variant) of object;
```

C++

```
(Sender: TObject; Value: Variant) ( TAdapterFieldValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldValueEvent.

57.5.29 WebAdapt.TAdapterHiddenFieldOptions Type

Pascal

```
TAdapterHiddenFieldOptions = set of TAdapterHiddenFieldOption;
```

C++

```
TAdapterHiddenFieldOption TAdapterHiddenFieldOptions;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterHiddenFieldOptions.

57.5.30 WebAdapt.TEndUserGetNameEvent Type

Pascal

```
TEndUserGetNameEvent = procedure (Sender: TObject; var Name: string) of object;
```

C++

```
(Sender: TObject; var Name: string) ( TEndUserGetNameEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserGetNameEvent.

57.5.31 WebAdapt.TEndUserGetUserIDEvent Type

Pascal

```
TEndUserGetUserIDEvent = procedure (Sender: TObject; var UserID: Variant) of object;
```

C++

```
(Sender: TObject; var UserID: Variant) ( TEndUserGetUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserGetUserIDEvent.

57.5.32 WebAdapt.TEndUserHandledEvent Type

Pascal

```
TEndUserHandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( TEndUserHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserHandledEvent.

57.5.33 WebAdapt.TEndUserIsLoggedInEvent Type

Pascal

```
TEndUserIsLoggedInEvent = procedure (Sender: TObject; var IsLoggedIn: Boolean) of object;
```

C++

```
(Sender: TObject; var IsLoggedIn: Boolean) ( TEndUserIsLoggedInEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserIsLoggedInEvent.

57.5.34 WebAdapt.TEndUserRightsEvent Type

Pascal

```
TEndUserRightsEvent = procedure (Sender: TObject; Rights: TStrings; var HasRights, Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Rights: TStrings; var HasRights, Handled: Boolean) ( TEndUserRightsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserRightsEvent.

57.5.35 WebAdapt.TEndUserUserIDEvent Type

Pascal

```
TEndUserUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TEndUserUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserUserIDEvent.

57.5.36 WebAdapt.TGetEOFEvent Type

Pascal

```
TGetEOFEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetEOFEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetEOFEvent.

57.5.37 WebAdapt.TGetRecordCountEvent Type

Pascal

```
TGetRecordCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TGetRecordCountEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordCountEvent.

57.5.38 WebAdapt.TGetRecordEvent Type

Pascal

```
TGetRecordEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetRecordEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordEvent.

57.5.39 WebAdapt.TGetRecordIndexEvent Type

Pascal

```
TGetRecordIndexEvent = procedure (Sender: TObject; var Index: Integer) of object;
```

C++

```
(Sender: TObject; var Index: Integer) ( TGetRecordIndexEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordIndexEvent.

57.5.40 WebAdapt.THandledEvent Type

Pascal

```
THandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( THandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.THandledEvent.

57.5.41 WebAdapt.TImageFieldGetHREFEvent Type

Pascal

```
TImageFieldGetHREFEvent = procedure (Sender: TObject; var HREF: string) of object;
```

C++

```
(Sender: TObject; var HREF: string) ( TImageFieldGetHREFEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetHREFEvent.

57.5.42 WebAdapt.TImageFieldGetImageEvent Type

Pascal

```
TImageFieldGetImageEvent = procedure (Sender: TObject; Params: TStrings; var MimeType: string;
var Image: TStream; var Owned: Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var MimeType: string; var Image: TStream; var Owned:
Boolean) ( TImageFieldGetImageEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetImageEvent.

57.5.43 WebAdapt.TImageFieldGetParamsEvent Type

Pascal

```
TImageFieldGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TImageFieldGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetParamsEvent.

57.5.44 WebAdapt.TIterateRecordsEvent Type

Pascal

```
TIterateRecordsEvent = procedure (Sender: TObject; Action: TIteratorMethod; var Eof: Boolean)
of object;
```

C++

```
(Sender: TObject; Action: TIteratorMethod; var Eof: Boolean) ( TIterateRecordsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TIterateRecordsEvent.

57.5.45 WebAdapt.TLoginUserIDEvent Type

Pascal

```
TLoginUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TLoginUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TLoginUserIDEvent.

57.5.46 WebAdapt.TObjectList Type

Pascal

```
TObjectList = Contnrs.TObjectList;
```

C++

```
Contnrs.TObjectList TObjectList;
```

File

WebAdapt

Description

This is type WebAdapt.TObjectList.

57.5.47 WebAdapt.TValidateAdapterFieldEvent Type

Pascal

```
TValidateAdapterFieldEvent = procedure (Sender: TObject; Value: Variant; var Handled: Boolean)
of object;
```

C++

```
(Sender: TObject; Value: Variant; var Handled: Boolean) ( TValidateAdapterFieldEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TValidateAdapterFieldEvent.

57.5.48 WebAdapt.TWebFormGetPageNameEvent Type

Pascal

```
TWebFormGetPageNameEvent = procedure (Sender: TObject; APageIndex: Integer; var Value: string)
of object;
```

C++

```
(Sender: TObject; APageIndex: Integer; var Value: string) ( TWebFormGetPageNameEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TWebFormGetPageNameEvent.

57.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 5906)	This is variable WebAdapt.NotifyPossibleHTMLChange.

57.6.1 WebAdapt.NotifyPossibleHTMLChange Variable

Pascal

```
NotifyPossibleHTMLChange: procedure;
```

C++

```
procedure NotifyPossibleHTMLChange;
```

File

WebAdapt

Description

This is variable WebAdapt.NotifyPossibleHTMLChange.

57.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sActionRequestLong (see page 5907)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 5907)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 5908)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 5908)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 5908)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 5909)	This is constant WebAdapt.sLoginUserName.
sNextPageAction (see page 5909)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 5909)	This is constant WebAdapt.sPrevPageAction.

57.7.1 WebAdapt.sActionRequestLong Constant

Pascal

```
sActionRequestLong = '__action';
```

C++

```
sActionRequestLong = '__action';
```

File

WebAdapt

Description

This is constant WebAdapt.sActionRequestLong.

57.7.2 WebAdapt.sCurrentPage Constant

Pascal

```
sCurrentPage = '_cp';
```

C++

```
sCurrentPage = '_cp';
```

File

WebAdapt

Description

This is constant WebAdapt.sCurrentPage.

57.7.3 WebAdapt.sGotoPageAction Constant

Pascal

```
sGotoPageAction = 'GotoPage';
```

C++

```
sGotoPageAction = 'GotoPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sGotoPageAction.

57.7.4 WebAdapt.sImageRequest Constant

Pascal

```
sImageRequest = '__img';
```

C++

```
sImageRequest = '__img';
```

File

WebAdapt

Description

This is constant WebAdapt.sImageRequest.

57.7.5 WebAdapt.sLoginPassword Constant

Pascal

```
sLoginPassword = 'Password';
```

C++

```
sLoginPassword = 'Password';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginPassword.

57.7.6 WebAdapt.sLoginUserName Constant

Pascal

```
sLoginUserName = 'UserName';
```

C++

```
sLoginUserName = 'UserName';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginUserName.

57.7.7 WebAdapt.sNextPageAction Constant

Pascal

```
sNextPageAction = 'NextPage';
```

C++

```
sNextPageAction = 'NextPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sNextPageAction.

57.7.8 WebAdapt.sPrevPageAction Constant

Pascal

```
sPrevPageAction = 'PrevPage';
```

C++

```
sPrevPageAction = 'PrevPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sPrevPageAction.

58 WebBroker Namespace

This is namespace WebBroker.

Topics

Name	Description
Classes (see page 5912)	The following table lists classes in this documentation.
Types (see page 5917)	The following table lists types in this documentation.
Variables (see page 5918)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebApplication (see page 5912)	TWebApplication encapsulates a Web server application.

Types

Name	Description
THandleShutdownException (see page 5917)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 5917)	This is type WebBroker.TServerExceptionEvent.

Variables

Name	Description
Application (see page 5918)	This is variable WebBroker.Application.
HandleShutdownException (see page 5918)	This is variable WebBroker.HandleShutdownException.

58.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebApplication (see page 5912)	TWebApplication encapsulates a Web server application.

58.1.1 TWebApplication Class

TWebApplication encapsulates a Web server application.

Class Hierarchy

Pascal

```
TWebApplication = class(TWebRequestHandler);
```

C++

```
class TWebApplication : public TWebRequestHandler;
```

File

WebBroker

Description

TWebApplication provides the fundamental behavior of a Web server application. TWebApplication provides methods to create TWebRequest and TWebResponse objects that represent HTTP request messages and their responses, pass these objects to the dispatcher so that the response can be filled in, and send the response back to the Web client.

Each Web server project automatically declares an Application (see page 5918) variable of type TWebApplication as the instance of the application. For CGI applications, this variable is initialized to a TCGIApplication object in the CGIApp unit. For Apache applications, this variable is initialized to a TApacheApplication object in the ApacheApp unit. For ISAPI applications, this variable is initialized to a TISAPIApplication object in the ISAPIApp unit.

The Application (see page 5918) variable is added to a project by selecting File | New (see page 3712) from the menu of the IDE main window and selecting Web Server Application (see page 5918) in the New (see page 3712) Items dialog. Do not add the Forms unit to the uses list of the project (Delphi) or include Forms.hpp in the project source file (C++). These actions declare a conflicting Application (see page 5918) variable of type TApplication.In addition, do not add the ApacheApp or ISAPIApp unit to a CGI application or the CGIApp unit to an Apache, ISAPI or NSAPI application. The CGIApp unit, the ISAPIApp unit, and the ApacheApp unit each create the initial instance of the Application (see page 5918) variable. Including more than one of these units in the uses list causes the memory for the object created in the first unit to be lost.

See Also

Application (see page 5918)

TCGIApplication

TISAPIApplication

TApacheApplication

TWebDispatcher

TWebRequest

TWebResponse

58.1.1.1 TWebApplication Members

The following tables list the members exposed by TWebApplication.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6602)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6603)	Deactivates the Web module of the application.
	HandleRequest (see page 6603)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6602)	Adds a Web module factory to the request handler.
V	Create (see page 5913)	Creates an instance of TWebApplication (see page 5912).
V	CreateForm (see page 5914)	Registers the single Web module with the Web application.
V	Destroy (see page 5914)	Destroys an instance of TWebApplication (see page 5912).
	HandleException (see page 6603)	Responds to exceptions related to TWebRequestHandler (see page 6600) objects.
V	Initialize (see page 5915)	Introduces an interface for a method that initializes the application.
V	Run (see page 5915)	Executes the Web application.

Public Properties

	Name	Description
R	ActiveCount (see page 6604)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6605)	Determines whether the Web modules used by the application are cached for re-use.
R	InactiveCount (see page 6605)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6606)	Sets or returns the MaxConnections used by the socket.
	Title (see page 5916)	Specifies a title for the Web application.
	WebModuleClass (see page 6606)	Contains a reference to a Web module.
R	WebModuleFactory (see page 6607)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 6607)	The number of Web module factories in use by the request handler.

58.1.1.2 TWebApplication.Create Constructor

Creates an instance of TWebApplication (see page 5912).

Pascal

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

C++

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

Description

Do not call Create. The single Web application object named Application (see page 5918) is instantiated automatically when the CGIApp unit, the ISAPIApp unit, or the ApacheApp unit is included in the uses list of the project.

Create allocates the helper objects used by TWebApplication (see page 5912).

See Also

Create

58.1.1.3 TWebApplication.Destroy Destructor

Destroys an instance of TWebApplication (see page 5912).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebApplication();
```

Description

Never call the Destroy or Free method. The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web modules created by the application.

See Also

System::TObject::Free (see page 3382)

Run (see page 5915)

58.1.1.4 TWebApplication Methods

The methods of the TWebApplication class are listed here.

Public Methods

	Name	Description
	CreateForm (see page 5914)	Registers the single Web module with the Web application.
	Initialize (see page 5915)	Introduces an interface for a method that initializes the application.
	Run (see page 5915)	Executes the Web application.

58.1.1.4.1 TWebApplication.CreateForm Method

Registers the single Web module with the Web application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the Web module for the Web application.

CreateForm registers the class of the Web module for the application.

See Also

TWebModule

58.1.1.4.2 TWebApplication.Initialize Method

Introduces an interface for a method that initializes the application.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

The Initialize method for TWebApplication (see page 5912) does nothing. Initialize is the first method called for every project. Descendants of TWebApplication (see page 5912) that must perform any initialization before the application starts running override this method for that purpose.

Note: Although Initialize is the first method called in the project source code, it is not the first code that is executed in an application.

See Also

Run (see page 5915)

58.1.1.4.3 TWebApplication.Run Method

Executes the Web application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

When the application is executed, the application's Run method is calledautomatically.

The Run method for TWebApplication (see page 5912) sets up the default error handler and shut down sequence for the application. Descendants of TWebApplication (see page 5912) override the Run method to receive HTTP request messages, create the TWebRequest and TWebResponse objects for incoming requests, and call the HandleRequest method to process those requests.

See Also

TWebRequest

TWebResponse

58.1.1.5 TWebApplication Properties

The properties of the TWebApplication class are listed here.

Public Properties

	Name	Description
	Title (see page 5916)	Specifies a title for the Web application.

58.1.1.5.1 TWebApplication.Title Property

Specifies a title for the Web application.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Use the Title property to assign a title to your Web application. This title can be used in HTTP messages that the application sends to identify the application. The default value is taken from the executable DLL/EXE .so filename at runtime.

Do not confuse Title with the Name property. Name is the identifier for the Web application when it is referred to in code.

See Also

Name

58.2 Types

The following table lists types in this documentation.

Types

Name	Description
THandleShutdownException (see page 5917)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 5917)	This is type WebBroker.TServerExceptionEvent.

58.2.1 WebBroker.THandleShutdownException Type

Pascal

```
THandleShutdownException = procedure (E: Exception);
```

C++

```
(E: Exception) ( THandleShutdownException)();
```

File

WebBroker

Description

This is type WebBroker.THandleShutdownException.

58.2.2 WebBroker.TServerExceptionEvent Type

Pascal

```
TServerExceptionEvent = procedure (E: Exception; wr: TWebResponse) of object;
```

C++

```
(E: Exception; wr: TWebResponse) ( TServerExceptionEvent)();
```

File

WebBroker

Description

This is type WebBroker.TServerExceptionEvent.

58.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 5918)	This is variable WebBroker.Application.
HandleShutdownException (see page 5918)	This is variable WebBroker.HandleShutdownException.

58.3.1 WebBroker.Application Variable

Pascal

```
Application: TWebApplication = nil;
```

C++

```
TWebApplication Application = nil;
```

File

WebBroker

Description

This is variable WebBroker.Application.

58.3.2 WebBroker.HandleShutdownException Variable

Pascal

```
HandleShutdownException: THandleShutdownException;
```

C++

```
THandleShutdownException HandleShutdownException;
```

File

WebBroker

Description

This is variable WebBroker.HandleShutdownException.

59 WebBrokerSOAP Namespace

This is namespace WebBrokerSOAP.

Topics

Name	Description
Classes (see page 5920)	The following table lists classes in this documentation.
Functions (see page 5925)	The following table lists functions in this documentation.

Classes

	Name	Description
	THTTPSoapDispatcher (see page 5920)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Functions

	Name	Description
	GetSOAPWebModule (see page 5925)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

59.1 Classes

The following table lists classes in this documentation.

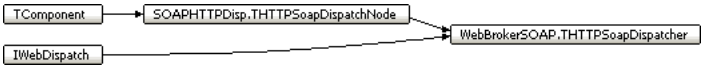
Classes

	Name	Description
	THTTPSoapDispatcher (see page 5920)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

59.1.1 THTTPSoapDispatcher Class

THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Class Hierarchy



Pascal

```
THTTPSoapDispatcher = class(THTTPSoapDispatchNode, IWebDispatch);
```

C++

```
class THTTPSoapDispatcher : public THTTPSoapDispatchNode, public IWebDispatch;
```

File

WebBrokerSOAP

Description

Use THTTPSoapDispatcher in an application that publishes a Web Service using SOAP. THTTPSoapDispatcher responds to all SOAP-based calls made to the invokable interfaces that are registered in the application. These interfaces are registered by calling the RegisterInterface method of the invocation registry (InvRegistry).

THTTPSoapDispatcher only acts as a dispatcher, receiving incoming messages and forwarding them on to another component that handles the task of interpreting and executing them. The component that interprets and executes SOAP message is identified by the Dispatcher property. Dispatcher is the interface of an invoker component, which interprets the SOAP message, identifies the invokable interface that is the target of the call, executes the call, and assembles the content of the response message.

THTTPSoapDispatcher automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the SOAP dispatcher without requiring the use of Web action items.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

IInvokable (see page 3505)

InvRegistry

TInvokableClassRegistry

Using the SOAP Application Wizard

Writing Servers that Support Web Services

59.1.1.1 THTTPSoapDispatcher Members

The following tables list the members exposed by THTTPSoapDispatcher.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1197)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	DispatchEnabled (see page 5922)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 5923)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 5923)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 5923)	Responds to an incoming message and generates the appropriate response message.
V	Notification (see page 1198)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
V	Create (see page 5921)	Creates a new THTTPSoapDispatcher (see page 5920) object.
V	Destroy (see page 5922)	Frees the memory associated with the THTTPSoapDispatcher (see page 5920) object.
V	DispatchSOAP (see page 1197)	Forwards the SOAP message on to the invoker for interpretation and execution.

Published Properties

	Name	Description
	Dispatcher (see page 1199)	Specifies the interface of the component that interprets SOAP messages.
	WebDispatch (see page 5924)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

59.1.1.2 THTTPSoapDispatcher.Create Constructor

Creates a new THTTPSoapDispatcher (see page 5920) object.

Pascal

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

C++

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

Description

Most applications do not need to call the THTTPSoapDispatcher (see page 5920) constructor. Typically the HTTP SOAP dispatcher is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the THTTPSoapDispatcher (see page 5920) instance.

See Also

Owner

59.1.1.3 THTTPSoapDispatcher.Destroy Destructor

Frees the memory associated with the THTTPSoapDispatcher (see page 5920) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPSoapDispatcher();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the THTTPSoapDispatcher (see page 5920) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

59.1.1.4 THTTPSoapDispatcher Methods

The methods of the THTTPSoapDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchEnabled (see page 5922)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 5923)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 5923)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 5923)	Responds to an incoming message and generates the appropriate response message.

59.1.1.4.1 THTTPSoapDispatcher.DispatchEnabled Method

Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In THTTPSoapDispatcher (see page 5920), DispatchEnabled always returns true, because the HTTP SOAP dispatcher can't be disabled.

See Also

WebDispatch (see page 5924)

59.1.1.4.2 THTTPSoapDispatcher.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the Mask property of the TWebDispatch object specified by the WebDispatch (see page 5924) property.

See Also

WebDispatch (see page 5924)

Mask

59.1.1.4.3 THTTPSoapDispatcher.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the MethodType property of the TWebDispatch object specified by the WebDispatch (see page 5924) property.

See Also

MethodType

WebDispatch (see page 5924)

59.1.1.4.4 THTTPSoapDispatcher.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of

the WebDispatch (see page 5924) property. DispatchRequest passes the request on to the component whose interface is the value of the Dispatcher property. That object interprets the request message, calls the appropriate invokable interface, and returns the content of the response message.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming SOAP message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully passes the request on to an invoker and that invoker handles the request and fills out the Response parameter. It returns false if it could not handle the request.

See Also

Dispatcher (see page 1199)

WebDispatch (see page 5924)

59.1.1.5 THTTPSoapDispatcher Properties

The properties of the THTTPSoapDispatcher class are listed here.

Published Properties

	Name	Description
	WebDispatch (see page 5924)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

59.1.1.5.1 THTTPSoapDispatcher.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the SOAP messages to which this component responds.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which the Web Service application responds, as well as the values allowed in the types of messages it accepts. By default, WebDispatch enables the HTTP SOAP dispatcher to respond to any URL whose path portion begins with the string "soap".

Note: You can't use WebDispatch to disable the THTTPSoapDispatcher (see page 5920) object. Setting its Enabled property to false does nothing.

See Also

TWebModule

TWebDispatcher

59.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPWebModule (see page 5925)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

59.2.1 WebBrokerSOAP.GetSOAPWebModule Function

Returns the Web module that contains the SOAP dispatcher that is processing the current request.

Pascal

```
function GetSOAPWebModule: TWebModule;
```

C++

```
TWebModule GetSOAPWebModule();
```

File

WebBrokerSOAP

Description

Call GetSOAPWebModule to access the application Web module of a Web Services application. This is the Web module that contains the SOAP dispatcher (THTTPSoapDispatcher (see page 5920)).

See Also

THTTPSOAPDispatcher

60 WebCntxt Namespace

This is namespace WebCntxt.

Topics

Name	Description
Classes (see page 5929)	The following table lists classes in this documentation.
Interfaces (see page 5962)	The following table lists interfaces in this documentation.
Functions (see page 5977)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5978)	The following table lists structs, records, enums in this documentation.
Types (see page 5979)	The following table lists types in this documentation.
Variables (see page 5981)	The following table lists variables in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 5929)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 5932)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 5935)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 5950)	TWebContext represents the current state of a WebSnap application.

Functions

	Name	Description
	WebContext (see page 5977)	Returns the current Web context

Interfaces

	Name	Description
	IWebEndUser (see page 5962)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 5970)	IWebPageInfo describes a Web page in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TFindPageOption (see page 5978)	This is record WebCntxt.TFindPageOption.

Types

Name	Description
TFindPageOptions (see page 5979)	This is type WebCntxt.TFindPageOptions.

TGetWebContextProc (see page 5979)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 5980)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 5980)	This is type WebCntxt.TSetWebContextProc.

Variables

Name	Description
GetModuleFileNameProc (see page 5981)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 5981)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 5982)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 5982)	This is variable WebCntxt.SetWebContextProc.

60.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 5929)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 5932)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 5935)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 5950)	TWebContext represents the current state of a WebSnap application.

60.1.1 TAbstractAdapterRequest Class

TAbstractAdapterRequest is the base class for adapter request objects.

Class Hierarchy

Pascal

```
TAbstractAdapterRequest = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterRequest : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterRequest is the base class for the object that adapters use to obtain information about the current HTTP request message. An application interacts with an adapter request object through interfaces implemented by various descendants of TAbstractAdapterRequest. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 5935)

IActionRequest

IAdapterModes

IactionFieldValues

IEnumActionFieldValueRecords

IAdapterRecordKeys

IActionFieldValuesOfAdapter

IAdapterDispatchParams (see page 929)

Iadaptermodes

iactionfieldvalues

ienumactionfieldvaluerecords

iadapterrecordkeys

iactionfieldvaluesofadapter

60.1.1.1 TAbstractAdapterRequest Members

The following tables list the members exposed by TAbstractAdapterRequest.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Request (see page 5932)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5931)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

60.1.1.2 TAbstractAdapterRequest.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterRequest();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

60.1.1.3 TAbstractAdapterRequest Properties

The properties of the TAbstractAdapterRequest class are listed here.

Protected Properties

	Name	Description
	Request (see page 5932)	This is Request, a member of class TAbstractAdapterRequest.

60.1.1.3.1 TAbstractAdapterRequest.Request Property

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

This is Request, a member of class TAbstractAdapterRequest.

60.1.2 TAbstractAdapterResponse Class

TAbstractAdapterResponse is the base class for adapter response objects.

Class Hierarchy

Pascal

```
TAbstractAdapterResponse = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterResponse : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterResponse is the base class for the object that adapters use to obtain information about the current HTTP response message. An application interacts with an adapter response object through interfaces implemented by various descendants of TAbstractAdapterResponse. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 5935)

IActionResponse

60.1.2.1 TAbstractAdapterResponse Members

The following tables list the members exposed by TAbstractAdapterResponse.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Response (see page 5934)	This is Response, a member of class TAbstractAdapterResponse.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5934)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.

	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

60.1.2.2 TAbstractAdapterResponse.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterResponse();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

60.1.2.3 TAbstractAdapterResponse Properties

The properties of the TAbstractAdapterResponse class are listed here.

Protected Properties

	Name	Description
R	Response (see page 5934)	This is Response, a member of class TAbstractAdapterResponse.

60.1.2.3.1 TAbstractAdapterResponse.Response Property

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

This is Response, a member of class TAbstractAdapterResponse.

60.1.3 TAbstractWebContext Class

TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.

Class Hierarchy

Pascal

```
TAbstractWebContext = class(TObject);
```

C++

```
class TAbstractWebContext : public TObject;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 5977) function.

See Also

WebContext (see page 5977)

TWebContext (see page 5950)

60.1.3.1 TAbstractWebContext Members

The following tables list the members exposed by TAbstractWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5942)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 5942)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 5942)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 5942)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 5943)	
	GetPageInfo (see page 5943)	
	GetRequestedPageName (see page 5943)	
	GetSession (see page 5943)	
	GetWebModules (see page 5944)	
	GetWebRequest (see page 5944)	
	GetWebResponse (see page 5944)	
	SetAdapterRequest (see page 5944)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 5945)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 5945)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 5945)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5937)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 5937)	Frees an instance of TAbstractWebContext (see page 5935).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
A	FindApplicationModule (see page 5938)	Returns the application module instance.
A	FindDefaultPageInfo (see page 5939)	Retrieves the Web page information object for a specified class of Web module.
A	FindModuleClass (see page 5939)	Returns an instance of a Web module, given its class reference.
A	FindModuleName (see page 5940)	Returns an instance of a Web module, given its class name.
A	FindPage (see page 5940)	Retrieves a Web page module given its name.
A	FindPageInfo (see page 5941)	Retrieves a Web page information object given the name of the corresponding Web page module.
A	FindVariableContainer (see page 5941)	Returns an instance of a Web data module, given its class name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AdapterRequest (see page 5946)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 5946)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5947)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 5947)	Provides access to information about the end user that sent the current request.

R	PageCount (see page 5947)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 5948)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 5948)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5949)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 5949)	Represents the current HTTP response that the application is assembling.
R	Session (see page 5949)	Provides access to the current Web session.
R	WebModules (see page 5950)	Lists all the Modules in the WebSnap application.

60.1.3.2 TAbstractWebContext.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractWebContext();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

60.1.3.3 TAbstractWebContext.Destroy Destructor

Frees an instance of TAbstractWebContext (see page 5935).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TAbstractWebContext (see page 5935) destructor unregisters itself with the global WebContext (see page 5977) function so that the WebContext (see page 5977) function never returns a Web context that no longer exists.

See Also

WebContext (see page 5977)

60.1.3.4 TAbstractWebContext Methods

The methods of the TAbstractWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5942)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 5942)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 5942)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 5942)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 5943)	
	GetPageInfo (see page 5943)	
	GetRequestedPageName (see page 5943)	
	GetSession (see page 5943)	
	GetWebModules (see page 5944)	
	GetWebRequest (see page 5944)	
	GetWebResponse (see page 5944)	
	SetAdapterRequest (see page 5944)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 5945)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 5945)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 5945)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 5938)	Returns the application module instance.
	FindDefaultPageInfo (see page 5939)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5939)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5940)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5940)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5941)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5941)	Returns an instance of a Web data module, given its class name.

60.1.3.4.1 TAbstractWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; virtual;
abstract;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext) = 0;
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 5950)

TWebAppPageModule (see page 6554)

TWebAppDataModule (see page 6550)

60.1.3.4.2 TAbstractWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 5941)

60.1.3.4.3 TAbstractWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass) = 0;
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 5940)

FindPage (see page 5940)

60.1.3.4.4 TAbstractWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AClass) = 0;
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 5939) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 5939)

FindPage (see page 5940)

FindVariableContainer (see page 5941)

60.1.3.4.5 TAbstractWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPage(const UnicodeString APage, TFindPageOptions AOptions,
TObject * AComponent) = 0;
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 5940)

FindPageInfo (see page 5941)

FindModuleClass (see page 5939)

60.1.3.4.6 TAbstractWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const UnicodeString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 5940)

FindDefaultPageInfo (see page 5939)

Pages (see page 5948)

60.1.3.4.7 TAbstractWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const UnicodeString AIdent) = 0;
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an

instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 5940)

60.1.3.4.8 TAbstractWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest() = 0;
```

Description

This is GetAdapterRequest, a member of class TAbstractWebContext.

60.1.3.4.9 TAbstractWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse() = 0;
```

Description

This is GetAdapterResponse, a member of class TAbstractWebContext.

60.1.3.4.10 TAbstractWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetDispatchedPageName() = 0;
```

Description

This is GetDispatchedPageName, a member of class TAbstractWebContext.

60.1.3.4.11 TAbstractWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; virtual; abstract;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser() = 0;
```

Description

This is GetEndUser, a member of class TAbstractWebContext.

60.1.3.4.12 TAbstractWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetPageCount() = 0;
```

See Also

GetPageCount

GetPageInfo (see page 5943)

PageCount (see page 5947)

60.1.3.4.13 TAbstractWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; virtual; abstract;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I) = 0;
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.3.4.14 TAbstractWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetRequestedPageName() = 0;
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.3.4.15 TAbstractWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession() = 0;
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.3.4.16 TAbstractWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules() = 0;
```

See Also

WebModules (see page 5950)

60.1.3.4.17 TAbstractWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; virtual; abstract;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest() = 0;
```

See Also

GetWebResponse (see page 5944)

Request (see page 5948)

60.1.3.4.18 TAbstractWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; virtual; abstract;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse() = 0;
```

See Also

GetWebRequest (see page 5944)

Response (see page 5949)

60.1.3.4.19 TAbstractWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf) = 0;
```

Description

This is SetAdapterRequest, a member of class TAbstractWebContext.

60.1.3.4.20 TAbstractWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf) = 0;
```

Description

This is SetAdapterResponse, a member of class TAbstractWebContext.

60.1.3.4.21 TAbstractWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetDispatchedPageName(const UnicodeString APageName) = 0;
```

Description

This is SetDispatchedPageName, a member of class TAbstractWebContext.

60.1.3.4.22 TAbstractWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetRequestedPageName(const UnicodeString APageName) = 0;
```

Description

This is SetRequestedPageName, a member of class TAbstractWebContext.

60.1.3.5 TAbstractWebContext Properties

The properties of the TAbstractWebContext class are listed here.

Public Properties

	Name	Description
R	AdapterRequest (see page 5946)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 5946)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5947)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 5947)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 5947)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 5948)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 5948)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5949)	Indicates the name of the Web page module to which the current request is addressed.

R	Response (see page 5949)	Represents the current HTTP response that the application is assembling.
R	Session (see page 5949)	Provides access to the current Web session.
R	WebModules (see page 5950)	Lists all the Modules in the WebSnap application.

60.1.3.5.1 TAbstractWebContext.AdapterRequest Property

Provides access to the adapter request object that manages access to the current request.

Pascal

```
property AdapterRequest: IUnknown;
```

C++

```
__property IUnknown AdapterRequest;
```

Description

AdapterRequest is the interface of the object that adapters use to obtain information about the current HTTP request message. The adapter request object obtains its information from the TWebRequest object that is the value of the Request (see page 5948) property.

Note: In Delphi, this interface can typically be cast to an IActionRequest interface.

Note: In C++, you can generally obtain an IActionRequest interface from this interface.

See Also

AdapterResponse (see page 5946)

TAbstractAdapterRequest (see page 5929)

Request (see page 5948)

IActionRequest

60.1.3.5.2 TAbstractWebContext.AdapterResponse Property

Provides access to the adapter response object that manages access to the current response.

Pascal

```
property AdapterResponse: IUnknown;
```

C++

```
__property IUnknown AdapterResponse;
```

Description

AdapterResponse is the interface of the object that adapters use to obtain information about the current HTTP response message. The adapter response object obtains its information from the TWebResponse object that is the value of the Response (see page 5949) property.

Note: In Delphi, this interface can typically be cast to an IActionResponse interface.

Note: In C++, you can generally obtain an IActionResponse interface from this interface.

See Also

AdapterRequest (see page 5946)

TAbstractAdapterResponse (see page 5932)

Response (see page 5949)

IActionResponse

60.1.3.5.3 TAbstractWebContext.DispatchedPageName Property

Specifies the name of the Web page module that is producing content to respond to the current request.

Pascal

```
property DispatchedPageName: string;
```

C++

```
__property UnicodeString DispatchedPageName;
```

Description

Read (see page 3727) DispatchedPageName to determine the Web page module that is handling the current HTTP request message. Although this is typically the same as the page that the RequestedPageName (see page 5949) property specifies, it can differ if the request was redirected to another Web page or if its use was delayed because the caller had not yet logged in.

See Also

RequestedPageName (see page 5949)

FindPageInfo (see page 5941)

pagename

60.1.3.5.4 TAbstractWebContext.EndUser Property

Provides access to information about the end user that sent the current request.

Pascal

```
property EndUser: IWebEndUser;
```

C++

```
__property IWebEndUser EndUser;
```

Description

Use EndUser to obtain information about the caller that sent the current request. You can use the IWebEndUser (see page 5962) interface to determine whether the end user has logged in, whether the user has access rights to perform specified operations on pages that have a specified access string, the display name for the end user, and so on.

EndUser is nil (Delphi) or NULL (C++) if the WebSnap application does not include an end user adapter such as TEndUserAdapter or TEndUserSessionAdapter.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

60.1.3.5.5 TAbstractWebContext.PageCount Property

Indicates the number of registered Web pages available in the WebSnap application.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount indicates the number of entries in the Pages (see page 5948) property array. Use this value as an upper bound for the index when iterating over all the pages in the WebSnap application.

See Also

Pages (see page 5948)

60.1.3.5.6 TAbstractWebContext.Pages Property

Lists the page information interfaces of the registered Web pages available in the WebSnap application.

Pascal

```
property Pages [I: Integer]: IWebPageInfo;
```

C++

```
__property IWebPageInfo Pages[int I];
```

Description

Use Pages to obtain an IWebPageInfo (see page 5970) interface for a registered Web page module. The IWebPageInfo (see page 5970) interface describes the characteristics of a Web page, including its name, title, HREF, description, an indication of whether users must log in to access the page, an indication of whether the current user has access rights to view the page, and so on.

Pages provides indexed access to page information interfaces. To access the information about a Web page given its name, use the FindPageInfo (see page 5941) method instead. To access the information given the class of the Web page module, use the FindDefaultPageInfo (see page 5939) method instead. To access the page module itself (rather than the page information interface), use the WebModules (see page 5950) property instead.

See Also

PageCount (see page 5947)

FindPageInfo (see page 5941)

WebModules (see page 5950)

FindDefaultPageInfo (see page 5939)

60.1.3.5.7 TAbstractWebContext.Request Property

Represents the current HTTP request that the application is handling.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3727) Request to obtain information about the current HTTP request message. The request object provides

access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 5949)

60.1.3.5.8 TAbstractWebContext.RequestedPageName Property

Indicates the name of the Web page module to which the current request is addressed.

Pascal

```
property RequestedPageName: string;
```

C++

```
__property UnicodeString RequestedPageName;
```

Description

When the page dispatcher receives an HTTP request message, it sets RequestedPageName to the name of the target Web page module. This is taken from the pathinfo portion of the request message's target URL.

Note: There is no guarantee that RequestedPageName refers to a registered Web page module, because this property is set before the dispatcher verifies that the requested page exists.

See Also

DispatchedPageName (see page 5947)

FindPageInfo (see page 5941)

Request (see page 5948)

TPageDispatcher (see page 6210)

60.1.3.5.9 TAbstractWebContext.Response Property

Represents the current HTTP response that the application is assembling.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3727) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 5948) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 5948)

60.1.3.5.10 TAbstractWebContext.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Use Session to get or set information about the current session. Sessions store information about an end user that is needed for a short period of time. This can be used for such tasks as login/logout support, recording user preferences, and so on.

Note: If you want to use a session to store end user information, the WebSnap application should include a TSessionsService component.

See Also

TSessionsService (see page 6722)

60.1.3.5.11 TAbstractWebContext.WebModules Property

Lists all the Modules in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

See Also

Pages (see page 5948)

FindModuleClass (see page 5939)

FindModuleName (see page 5940)

FindApplicationModule (see page 5938)

FindModuleClass (see page 5939)

60.1.4 TWebContext Class

TWebContext represents the current state of a WebSnap application.

Class Hierarchy

Pascal

```
TWebContext = class(TAbstractWebContext);
```

C++

```
class TWebContext : public TAbstractWebContext;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 5977) function.

TWebContext provides the implementation of the abstract or (in C++ terminology) pure virtual methods introduced by its immediate ancestor, TAbstractWebContext (see page 5935).

See Also

WebContext (see page 5977)

Dispatching Requests and Responses

60.1.4.1 TWebContext Members

The following tables list the members exposed by TWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5958)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 5958)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 5958)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 5958)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 5959)	
	GetPageInfo (see page 5959)	
	GetRequestedPageName (see page 5959)	
	GetSession (see page 5959)	
	GetWebModules (see page 5960)	
	GetWebRequest (see page 5960)	
	GetWebResponse (see page 5960)	
	SetAdapterRequest (see page 5960)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 5961)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 5961)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 5961)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5952)	Creates an initializes an instance of TWebContext (see page 5950).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 5953)	Frees an instance of TWebContext (see page 5950).

	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindApplicationModule (see page 5954)	Returns the application module instance.
	FindDefaultPageInfo (see page 5955)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5955)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5956)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5956)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5957)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5957)	Returns an instance of a Web data module, given its class name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AdapterRequest (see page 5946)	Provides access to the adapter request object that manages access to the current request.
	AdapterResponse (see page 5946)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5947)	Specifies the name of the Web page module that is producing content to respond to the current request.
	EndUser (see page 5947)	Provides access to information about the end user that sent the current request.
	PageCount (see page 5947)	Indicates the number of registered Web pages available in the WebSnap application.
	Pages (see page 5948)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
	Request (see page 5948)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5949)	Indicates the name of the Web page module to which the current request is addressed.
	Response (see page 5949)	Represents the current HTTP response that the application is assembling.
	Session (see page 5949)	Provides access to the current Web session.
	WebModules (see page 5950)	Lists all the Modules in the WebSnap application.

60.1.4.2 TWebContext.Create Constructor

Creates an initializes an instance of TWebContext (see page 5950).

Pascal

```
constructor Create(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest; AResponse:
TWebResponse; ASession: TAbstractWebSession);
```

C++

```
__fastcall TWebContext(TAbstractWebModuleList AWebModuleList, TWebRequest ARequest,
TWebResponse AResponse, TAbstractWebSession ASession);
```

Description

Do not create instances of TWebContext (see page 5950). WebSnap applications create a global Web context automatically to represent the current state of the application. To access this global Web context object, use the global WebContext (see page 5977) function.

The TWebContext (see page 5950) constructor registers itself with the global WebContext (see page 5977) function so that the application can access the Web context using that function, and initializes the properties of the Web context to the values passed in as parameters.

AWebModuleList is the component that manages the Web modules in the application. It becomes the value of the WebModules (see page 5950) property.

ARequest is the current HTTP request message. It becomes the value of the Request (see page 5948) property.

AResponse is the current response object that the application is filling out. It becomes the value of the Response (see page 5949) property.

ASession is the Session (see page 5949) object associated with ARequest. It becomes the value of the Session (see page 5949) property.

See Also

WebContext (see page 5977)

WebModules (see page 5950)

Request (see page 5948)

Response (see page 5949)

Session (see page 5949)

60.1.4.3 TWebContext.Destroy Destructor

Frees an instance of TWebContext (see page 5950).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TWebContext (see page 5950) destructor unregisters itself with the global WebContext (see page 5977) function so that the WebContext (see page 5977) function never returns a Web context that no longer exists.

See Also

WebContext (see page 5977)

60.1.4.4 TWebContext Methods

The methods of the TWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5958)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 5958)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 5958)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 5958)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 5959)	
	GetPageInfo (see page 5959)	
	GetRequestedPageName (see page 5959)	
	GetSession (see page 5959)	
	GetWebModules (see page 5960)	
	GetWebRequest (see page 5960)	
	GetWebResponse (see page 5960)	
	SetAdapterRequest (see page 5960)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 5961)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 5961)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 5961)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 5954)	Returns the application module instance.
	FindDefaultPageInfo (see page 5955)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5955)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5956)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5956)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5957)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5957)	Returns an instance of a Web data module, given its class name.

60.1.4.4.1 TWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext);
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 5950)

TWebAppPageModule (see page 6554)

TWebAppDataModule (see page 6550)

60.1.4.4.2 TWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 5957)

60.1.4.4.3 TWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 5956)

FindPage (see page 5956)

60.1.4.4.4 TWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AName);
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 5955) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 5955)

FindPage (see page 5956)

FindVariableContainer (see page 5957)

60.1.4.4.5 TWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPage(const UnicodeString APage, TFindPageOptions AOptions,
TObject * AComponent);
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 5956)

FindPageInfo (see page 5957)

FindModuleClass (see page 5955)

60.1.4.4.6 TWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const UnicodeString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 5956)

FindDefaultPageInfo (see page 5955)

Pages (see page 5948)

60.1.4.4.7 TWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const UnicodeString AIdent);
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an

instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 5956)

60.1.4.4.8 TWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest();
```

Description

This is GetAdapterRequest, a member of class TWebContext.

60.1.4.4.9 TWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse();
```

Description

This is GetAdapterResponse, a member of class TWebContext.

60.1.4.4.10 TWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDispatchedPageName();
```

Description

This is GetDispatchedPageName, a member of class TWebContext.

60.1.4.4.11 TWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; override;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser();
```

Description

This is GetEndUser, a member of class TWebContext.

60.1.4.4.12 TWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; override;
```

C++

```
virtual __fastcall int GetPageCount();
```

See Also

GetPageCount (see page 5943)

GetPageInfo (see page 5943)

PageCount (see page 5947)

60.1.4.4.13 TWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; override;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I);
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.4.4.14 TWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetRequestedPageName();
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.4.4.15 TWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; override;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession();
```

See Also

GetPageCount (see page 5943)

Pages (see page 5948)

60.1.4.4.16 TWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; override;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules();
```

See Also

WebModules (see page 5950)

60.1.4.4.17 TWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; override;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest();
```

See Also

GetWebResponse (see page 5944)

Request (see page 5948)

60.1.4.4.18 TWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; override;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse();
```

See Also

GetWebRequest (see page 5944)

Response (see page 5949)

60.1.4.4.19 TWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf);
```

Description

This is SetAdapterRequest, a member of class TWebContext.

60.1.4.4.20 TWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf);
```

Description

This is SetAdapterResponse, a member of class TWebContext.

60.1.4.4.21 TWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetDispatchedPageName(const UnicodeString APageName);
```

Description

This is SetDispatchedPageName, a member of class TWebContext.

60.1.4.4.22 TWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetRequestedPageName(const UnicodeString APageName);
```

Description

This is SetRequestedPageName, a member of class TWebContext.

60.2 Interfaces

The following table lists interfaces in this documentation.

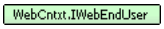
Interfaces

	Name	Description
	IWebEndUser (see page 5962)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 5970)	IWebPageInfo describes a Web page in a WebSnap application.

60.2.1 IWebEndUser Interface

IWebEndUser provides access information about the current end user in a WebSnap application.

Class Hierarchy



Pascal

```
IWebEndUser = interface;
```

C++

```
__interface IWebEndUser;
```

File

WebCntxt

Description

Use IWebEndUser to obtain information about the current end user, such as the user name, user ID, and whether the user is logged in. In addition to properties that provide information about the current end user, IWebEndUser has methods to log in an end user.

End user adapter components such as TEndUserAdapter and TEndUserSessionAdapter implement the IWebEndUser interface. If a WebSnap application includes an end user adapter, you can access its IWebEndUser interface by reading the EndUser property of the Web context.

Note: In C++, method declarations that use IWebEndUser use the _di_IWebEndUser type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebEndUser interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebEndUser > _di_IWebEndUser;

See Also

TAbstractWebContext (see page 5935)

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

60.2.1.1 IWebEndUser Members

The following tables list the members exposed by IWebEndUser.

Public Methods

	Name	Description
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 5967)	Logs in a specified user.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.

Public Properties

	Name	Description
R	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
R	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
R	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5969)	Indicates the action component for the login form.
R	LogoutAction (see page 5970)	Indicates the action component for logging a user out.

60.2.1.2 IWebEndUser Methods

The methods of the IWebEndUser class are listed here.

Public Methods

	Name	Description
	GetDisplayName (see page 5963)	Returns the value of the DisplayName (see page 5968) property.
	GetEndUserAdapter (see page 5964)	Returns the value of the EndUserAdapter (see page 5968) property.
	GetLoggedIn (see page 5964)	Returns the value of the LoggedIn (see page 5968) property.
	GetLoginFormAction (see page 5964)	Returns the value of the LoginFormAction (see page 5969) property.
	GetLogoutAction (see page 5965)	Returns the value of the LogoutAction (see page 5970) property.
	HasExecuteRights (see page 5965)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 5966)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 5966)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 5967)	Logs in a specified user.
	ProduceLoginPage (see page 5967)	Generates the Web page for logging in users.

60.2.1.2.1 IWebEndUser.GetDisplayName Method

Returns the value of the DisplayName (see page 5968) property.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the read implementation of the DisplayName (see page 5968) property.

See Also

DisplayName (see page 5968)

60.2.1.2.2 IWebEndUser.GetEndUserAdapter Method

Returns the value of the EndUserAdapter (see page 5968) property.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

GetEndUserAdapter is the read implementation of the EndUserAdapter (see page 5968) property.

See Also

EndUserAdapter (see page 5968)

60.2.1.2.3 IWebEndUser.GetLoggedIn Method

Returns the value of the LoggedIn (see page 5968) property.

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

GetLoggedIn is the read implementation of the LoggedIn (see page 5968) property.

See Also

LoggedIn (see page 5968)

60.2.1.2.4 IWebEndUser.GetLoginFormAction Method

Returns the value of the LoginFormAction (see page 5969) property.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

GetLoginFormAction is the read implementation of the LoginFormAction (see page 5969) property.

See Also

LoginFormAction (see page 5969)

60.2.1.2.5 IWebEndUser.GetLogoutAction Method

Returns the value of the LogoutAction (see page 5970) property.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

GetLogoutAction is the read implementation of the LogoutAction (see page 5970) property.

See Also

LogoutAction (see page 5970)

60.2.1.2.6 IWebEndUser.HasExecuteRights Method

Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(UnicodeString ARights);
```

Description

Call HasExecuteRights to determine whether the current end user is authorized to execute Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasExecuteRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasExecuteRights can determine the user's access rights.

See Also

LoggedIn (see page 5968)

HasViewRights (see page 5966)

HasModifyRights (see page 5966)

ViewAccess

ViewAccess (see page 5560)

TWebUserList (see page 6759)

AccessRightsStrings (see page 6754)

60.2.1.2.7 IWebEndUser.HasModifyRights Method

Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(UnicodeString ARights);
```

Description

Call HasModifyRights to determine whether the current end user is authorized to modify Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasModifyRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasModifyRights can determine the user's access rights.

See Also

LoggedIn (see page 5968)

HasViewRights (see page 5966)

HasExecuteRights (see page 5965)

ViewAccess

ViewAccess (see page 5560)

TWebUserList (see page 6759)

AccessRightsStrings (see page 6754)

60.2.1.2.8 IWebEndUser.HasViewRights Method

Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(UnicodeString ARights);
```

Description

Call HasViewRights to determine whether the current end user is authorized to view Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasViewRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasViewRights can determine the user's access rights.

See Also

LoggedIn (see page 5968)

HasExecuteRights (see page 5965)

HasModifyRights (see page 5966)

ViewAccess

ViewAccess (see page 5560)

TWebUserList (see page 6759)

AccessRightsStrings (see page 6754)

60.2.1.2.9 IWebEndUser.LogIn Method

Logs in a specified user.

Pascal

```
procedure LogIn(UserID: Variant);
```

C++

```
__fastcall LogIn(Variant UserID);
```

Description

Call LogIn to record a user as logged in. Typically, LogIn is called in response to a message from the login page that ProduceLoginPage (see page 5967) generates.

UserID is the identifier of the user that logged in. It is the value of the UserID property of the entry for the user in a WebUserList component, or, in applications that implement their own login scheme, any other Variant (see page 3430) that identifies an end user.

See Also

LoggedIn (see page 5968)

ProduceLoginPage (see page 5967)

TWebUserList (see page 6759)

UserID (see page 6754)

TLoginFormAdapter (see page 5810)

60.2.1.2.10 IWebEndUser.ProduceLoginPage Method

Generates the Web page for logging in users.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

Call ProduceLoginPage to generate a Web page for logging in users.

ProduceLoginPage returns true if it generates a login form page and updates the current response message so that it will be displayed to the user. ProduceLoginPage returns false if the end user adapter is unable to generate a login page.

See Also

LoggedIn (see page 5968)

LogIn (see page 5967)

LoginFormAction (see page 5969)

60.2.1.3 IWebEndUser Properties

The properties of the IWebEndUser class are listed here.

Public Properties

	Name	Description
	DisplayName (see page 5968)	Indicates the name of the end user as it should be used for display in Web pages.
	EndUserAdapter (see page 5968)	Provides access to the end user adapter that implements this interface.
	LoggedIn (see page 5968)	Indicates whether the current user is logged in.
	LoginFormAction (see page 5969)	Indicates the action component for the login form.
	LogoutAction (see page 5970)	Indicates the action component for logging a user out.

60.2.1.3.1 IWebEndUser.DisplayName Property

Indicates the name of the end user as it should be used for display in Web pages.

Pascal

```
property DisplayName: string;
```

C++

```
__property UnicodeString DisplayName;
```

Description

Read (see page 3727) DisplayName to obtain a string that can be added to Web pages to refer to the current user.

See Also

UserName (see page 6754)

60.2.1.3.2 IWebEndUser.EndUserAdapter Property

Provides access to the end user adapter that implements this interface.

Pascal

```
property EndUserAdapter: TComponent;
```

C++

```
__property TComponent * EndUserAdapter;
```

Description

Read (see page 3727) EndUserAdapter to obtain access to the end user adapter. This is the component that implements this interface. Typically, it is a TEndUserAdapter or TEndUserSessionAdapter instance.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

60.2.1.3.3 IWebEndUser.LoggedIn Property

Indicates whether the current user is logged in.

Pascal

```
property LoggedIn: Boolean;
```

C++

```
__property Boolean LoggedIn;
```

Description

Read (see page 3727) LoggedIn to determine whether the current user has logged in to the WebSnap application. By logging in, a caller can be identified as one of the users that a TWebUserList component lists.

Once the user has logged in, the HasViewRights (see page 5966), HasExecuteRights (see page 5965), and HasModifyRights (see page 5966) methods can be used to determine whether the user is authorized to view, execute actions, and modify data on restricted Web pages.

To log in a user that is not logged in, call the ProduceLoginPage (see page 5967) method to display a login Web page. That page should allow the user to provide some sort of UserID, which can then be used to log the user in by a call to the LogIn (see page 5967) method.

See Also

LogIn (see page 5967)

ProduceLoginPage (see page 5967)

TWebUserList (see page 6759)

HasExecuteRights (see page 5965)

HasModifyRights (see page 5966)

HasViewRights (see page 5966)

60.2.1.3.4 IWebEndUser.LoginFormAction Property

Indicates the action component for the login form.

Pascal

```
property LoginFormAction: TComponent;
```

C++

```
__property TComponent * LoginFormAction;
```

Description

Read (see page 3727) LoginFormAction to access the login form action component. The login form action component can be called from server-side scripts to display a login form Web page.

Note: To display the login form Web page using the IWebEndUser (see page 5962) interface (rather than from a server-side script), call the ProduceLoginPage (see page 5967) method.

See Also

ProduceLoginPage (see page 5967)

TEndUserAdapterLoginFormAction (see page 5762)

LogoutAction (see page 5970)

60.2.1.3.5 IWebEndUser.LogoutAction Property

Indicates the action component for logging a user out.

Pascal

```
property LogoutAction: TComponent;
```

C++

```
__property TComponent * LogoutAction;
```

Description

Read (see page 3727) LogoutAction to access the logout action component. The logout action component can be called from server-side scripts to perform any necessary tasks when the user logs out.

See Also

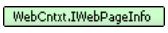
LoginFormAction (see page 5969)

TEndUserAdapterLogoutAction (see page 5770)

60.2.2 IWebPageInfo Interface

IWebPageInfo describes a Web page in a WebSnap application.

Class Hierarchy



Pascal

```
IWebPageInfo = interface;
```

C++

```
__interface IWebPageInfo;
```

File

WebCntxt

Description

Use IWebPageInfo to obtain information about a Web page, including its page name, page title, descriptive text, publish status, and access rights. This information is taken from a TWebPageInfo object that is stored with the Web page module factory.

Note: VCL Method declarations that use IWebPageInfo use the _di_IWebPageInfo type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebPageInfo interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebPageInfo >  _di_IWebPageInfo;
```

See Also

TWebPageInfo (see page 6271)

TAbstractWebContext (see page 5935)

60.2.2.1 IWebPageInfo Members

The following tables list the members exposed by IWebPageInfo.

Public Methods

	Name	Description
	GetHasViewAccess (see page 5971)	Returns the value of the HasViewAccess (see page 5974) property.
	GetLoginRequired (see page 5972)	Returns the value of the LoginRequired (see page 5974) property.
	GetPageDescription (see page 5972)	Returns the value of the PageDescription (see page 5974) property.
	GetPageHREF (see page 5972)	Returns the value of the PageHREF (see page 5975) property.
	GetPageIsPublished (see page 5973)	Returns the value of the PageIsPublished (see page 5975) property.
	GetPageName (see page 5973)	Returns the value of the PageName (see page 5976) property.
	GetPageTitle (see page 5973)	Returns the value of the PageTitle (see page 5976) property.

Public Properties

	Name	Description
R	HasViewAccess (see page 5974)	Indicates whether the current user has access rights to view the Web page.
R	LoginRequired (see page 5974)	Indicates whether callers must log in before accessing this Web page.
R	PageDescription (see page 5974)	Describes the Web page.
R	PageHREF (see page 5975)	Identifies the HREF to use for references to the associated Web page.
R	PageIsPublished (see page 5975)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 5976).
R	PageName (see page 5976)	Denotes the logical name for the Web page.
R	PageTitle (see page 5976)	Indicates the title that appears on the Web page.

60.2.2.2 IWebPageInfo Methods

The methods of the IWebPageInfo class are listed here.

Public Methods

	Name	Description
	GetHasViewAccess (see page 5971)	Returns the value of the HasViewAccess (see page 5974) property.
	GetLoginRequired (see page 5972)	Returns the value of the LoginRequired (see page 5974) property.
	GetPageDescription (see page 5972)	Returns the value of the PageDescription (see page 5974) property.
	GetPageHREF (see page 5972)	Returns the value of the PageHREF (see page 5975) property.
	GetPageIsPublished (see page 5973)	Returns the value of the PageIsPublished (see page 5975) property.
	GetPageName (see page 5973)	Returns the value of the PageName (see page 5976) property.
	GetPageTitle (see page 5973)	Returns the value of the PageTitle (see page 5976) property.

60.2.2.2.1 IWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess (see page 5974) property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess (see page 5974) property.

See Also

HasViewAccess (see page 5974)

60.2.2.2.2 IWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired (see page 5974) property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired (see page 5974) property.

See Also

LoginRequired (see page 5974)

60.2.2.2.3 IWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription (see page 5974) property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription (see page 5974) property.

See Also

PageDescription (see page 5974)

60.2.2.2.4 IWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF (see page 5975) property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF (see page 5975) property.

See Also

PageHREF (see page 5975)

60.2.2.2.5 IWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished (see page 5975) property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished (see page 5975) property.

See Also

PageIsPublished (see page 5975)

60.2.2.2.6 IWebPageInfo.GetPageName Method

Returns the value of the PageName (see page 5976) property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall UnicodeString GetPageName();
```

Description

GetPageName is the read implementation of the PageName (see page 5976) property.

See Also

PageName (see page 5976)

60.2.2.2.7 IWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle (see page 5976) property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle (see page 5976) property.

See Also

PageTitle (see page 5976)

60.2.2.3 IWebPageInfo Properties

The properties of the IWebPageInfo class are listed here.

Public Properties

	Name	Description
	HasViewAccess (see page 5974)	Indicates whether the current user has access rights to view the Web page.
	LoginRequired (see page 5974)	Indicates whether callers must log in before accessing this Web page.
	PageDescription (see page 5974)	Describes the Web page.
	PageHREF (see page 5975)	Identifies the HREF to use for references to the associated Web page.
	PageIsPublished (see page 5975)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 5976).
	PageName (see page 5976)	Denotes the logical name for the Web page.
	PageTitle (see page 5976)	Indicates the title that appears on the Web page.

60.2.2.3.1 IWebPageInfo.HasViewAccess Property

Indicates whether the current user has access rights to view the Web page.

Pascal

```
property HasViewAccess: Boolean;
```

C++

```
__property Boolean HasViewAccess;
```

Description

HasViewAccess is true if the current end user has view rights for the associated Web page, false otherwise.

See Also

HasViewRights (see page 5966)

60.2.2.3.2 IWebPageInfo.LoginRequired Property

Indicates whether callers must log in before accessing this Web page.

Pascal

```
property LoginRequired: Boolean;
```

C++

```
__property Boolean LoginRequired;
```

Description

Read (see page 3727) LoginRequired to determine whether users must log in before they can access this Web page. The application can overrule the LoginRequired property by supplying an OnIsLoginRequired event handler to the page dispatcher.

See Also

OnIsLoginRequired (see page 6173)

TPageDispatcher (see page 6210)

60.2.2.3.3 IWebPageInfo.PageDescription Property

Describes the Web page.

Pascal

```
property PageDescription: string;
```

C++

```
__property UnicodeString PageDescription;
```

Description

Read (see page 3727) PageDescription to obtain a brief description of the Web page.

See Also

PageName (see page 5976)

PageTitle (see page 5976)

60.2.2.3.4 IWebPageInfo.PageHREF Property

Identifies the HREF to use for references to the associated Web page.

Pascal

```
property PageHREF: string;
```

C++

```
__property UnicodeString PageHREF;
```

Description

Read (see page 3727) PageHREF to obtain the HTTP path string that should be inserted in HTML code to create a hypertext link to the associated Web page.

See Also

PageName (see page 5976)

60.2.2.3.5 IWebPageInfo.PageIsPublished Property

Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 5976).

Pascal

```
property PageIsPublished: Boolean;
```

C++

```
__property Boolean PageIsPublished;
```

Description

Read (see page 3727) PageIsPublished to determine whether the page dispatcher (TPageDispatcher) automatically calls on the associated Web page module when it receives HTTP request messages that address this Web page by name. HTTP request messages address a Web page when the pathinfo portion of their target URL matches the value of the PageName (see page 5976) property.

When IsPublished is false, the WebSnap application must explicitly retrieve the content for the Web page and add it to a response message.

See Also

PageName (see page 5976)

TPageDispatcher (see page 6210)

60.2.2.3.6 IWebPageInfo.PageName Property

Denotes the logical name for the Web page.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Read (see page 3727) PageName to determine the name of the Web page. This name is used to identify the page in server-side scripts, and as the pathinfo portion of the URL for the page when the PageIsPublished (see page 5975) property is true.

See Also

PageTitle (see page 5976)

PageHREF (see page 5975)

PageIsPublished (see page 5975)

60.2.2.3.7 IWebPageInfo.PageTitle Property

Indicates the title that appears on the Web page.

Pascal

```
property PageTitle: string;
```

C++

```
__property UnicodeString PageTitle;
```

Description

Read (see page 3727) PageTitle to determine the title that appears on the Web page when viewed from a user's browser.

See Also

PageName (see page 5976)

60.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebContext (see page 5977)	Returns the current Web context

60.3.1 WebCntxt.WebContext Function

Returns the current Web context

Pascal

```
function WebContext: TAbstractWebContext;
```

C++

```
TAbstractWebContext WebContext();
```

File

WebCntxt

Description

Use the WebContext function to obtain a reference to the global Web context. This object stores information about the components that are available in the Web application as well as about the current request that is being processed.

60.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFindPageOption (see page 5978)	This is record WebCntxt.TFindPageOption.

60.4.1 WebCntxt.TFindPageOption Enumeration

Pascal

```
TFindPageOption = (
  fpPublished,
  fpLoginRequired
);
```

C++

```
enum TFindPageOption {
  fpPublished,
  fpLoginRequired
};
```

File

WebCntxt

Description

This is record WebCntxt.TFindPageOption.

60.5 Types

The following table lists types in this documentation.

Types

Name	Description
TFindPageOptions (see page 5979)	This is type WebCntxt.TFindPageOptions.
TGetWebContextProc (see page 5979)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 5980)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 5980)	This is type WebCntxt.TSetWebContextProc.

60.5.1 WebCntxt.TFindPageOptions Type

Pascal

```
TFindPageOptions = set of TFindPageOption;
```

C++

```
TFindPageOption TFindPageOptions;
```

File

WebCntxt

Description

This is type WebCntxt.TFindPageOptions.

60.5.2 WebCntxt.TGetWebContextProc Type

Pascal

```
TGetWebContextProc = function : TAbstractWebContext;
```

C++

```
: TAbstractWebContext ( TGetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebContextProc.

60.5.3 WebCntxt.TGetWebEndUserProc Type

Pascal

```
TGetWebEndUserProc = function : IWebEndUser;
```

C++

```
: IWebEndUser ( TGetWebEndUserProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebEndUserProc.

60.5.4 WebCntxt.TSetWebContextProc Type

Pascal

```
TSetWebContextProc = procedure (AWebContext: TAbstractWebContext);
```

C++

```
(AWebContext: TAbstractWebContext) ( TSetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TSetWebContextProc.

60.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GetModuleFileNameProc (see page 5981)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 5981)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 5982)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 5982)	This is variable WebCntxt.SetWebContextProc.

60.6.1 WebCntxt.GetModuleFileNameProc Variable

Pascal

```
GetModuleFileNameProc: function: string = nil;
```

C++

```
function: string GetModuleFileNameProc = nil;
```

File

WebCntxt

Description

This is variable WebCntxt.GetModuleFileNameProc.

60.6.2 WebCntxt.GetWebContextProc Variable

Pascal

```
GetWebContextProc: TGetWebContextProc;
```

C++

```
TGetWebContextProc GetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebContextProc.

60.6.3 WebCntxt.GetWebEndUserProc Variable

Pascal

```
GetWebEndUserProc: TGetWebEndUserProc;
```

C++

```
TGetWebEndUserProc GetWebEndUserProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebEndUserProc.

60.6.4 WebCntxt.SetWebContextProc Variable

Pascal

```
SetWebContextProc: TSetWebContextProc;
```

C++

```
TSetWebContextProc SetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.SetWebContextProc.

61 WebComp Namespace

This is namespace WebComp.

Topics

Name	Description
Classes (see page 5985)	The following table lists classes in this documentation.
Interfaces (see page 6052)	The following table lists interfaces in this documentation.
Functions (see page 6079)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6082)	The following table lists structs, records, enums in this documentation.
Types (see page 6084)	The following table lists types in this documentation.
Constants (see page 6087)	The following table lists constants in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 5985)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 5988)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 5996)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 5999)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 6009)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 6012)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 6018)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 6026)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 6036)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 6041)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 6049)	TWebContentOptions controls what a Web item can include in its generated HTML.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 6087)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 6087)	This is constant WebComp.FindWebComponentEditorHelperProc.

RegisterWebComponentsProc (see page 6088)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 6088)	This is constant WebComp.UnRegisterWebComponentsProc.

Functions

	Name	Description
	DesigningComponent (see page 6079)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 6079)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 6080)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 6080)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 6080)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 6081)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 6081)	This is function WebComp.ValidateWebParent.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 6052)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 6054)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 6055)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 6056)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 6057)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 6059)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 6060)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 6062)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 6064)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 6065)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 6065)	IWebComponent is the interface supported by components that TWebComponentList (see page 5999) manages.
	IWebComponentCommands (see page 6069)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 6071)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 6073)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 6074)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 6075)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 6077)	This is class WebComp.IWebVariablesContainerName.

Structs, Records, Enums

	Name	Description
	TLabelPosition (see page 6082)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 6082)	This is record WebComp.TWebContentFlag.

Types

Name	Description
TEnumWebComponentsProc (see page 6084)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 6084)	This is type WebComp.TGetComponentListEvent.
TLayout (see page 6085)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 6085)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 6085)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 6086)	This is type WebComp.TWebContentFlags.

61.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 5985)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 5988)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 5996)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 5999)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 6009)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 6012)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 6018)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 6026)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 6036)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 6041)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 6049)	TWebContentOptions controls what a Web item can include in its generated HTML.

61.1.1 TLayoutAttributes Class

TLayoutAttributes describes how an HTML control should be laid out.

Class Hierarchy

Pascal

```
TLayoutAttributes = class(TObject);
```

C++

```
class TLayoutAttributes : public TObject;
```

File

WebComp

Description

TLayoutAttributes embodies a set of flags that describe how an HTML control should be laid out. These flags form the value of several public fields of the object, including

LabelAttributes (see page 5987): a string that describes style attributes that apply to the control's label (if any).

LabelPosition (see page 5988): an enumerated value describing the the position of the label (if any) relative to the control. Possible values are lposLeft (to the left of the control), lposRight (to the right of the control), lposAbove (above the control) and lposBelow (below the control).

ControlAttributes (see page 5987): a string that describes style attributes that apply to the control.

See Also

IWebContent (see page 6075)

61.1.1.1 TLayoutAttributes Members

The following tables list the members exposed by TLayoutAttributes.

Public Fields

	Name	Description
	ControlAttributes (see page 5987)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 5987)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 5988)	This is LabelPosition, a member of class TLayoutAttributes.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5987)	Creates a new TLayoutAttributes (see page 5985) object.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

61.1.1.2 TLayoutAttributes.Create Constructor

Creates a new TLayoutAttributes (see page 5985) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TLayoutAttributes();
```

Description

Typically, applications do not instantiate TLayoutAttributes (see page 5985). Instead, each Web item that generates HTML for an InternetExpress application instantiates an internal TLayoutAttributes (see page 5985) object and uses it to encode property values that affect the layout of its generated HTML.

61.1.1.3 TLayoutAttributes Fields

The fields of the TLayoutAttributes class are listed here.

Public Fields

	Name	Description
	ControlAttributes (see page 5987)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 5987)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 5988)	This is LabelPosition, a member of class TLayoutAttributes.

61.1.1.3.1 TLayoutAttributes.ControlAttributes Field

Pascal

```
ControlAttributes: string;
```

C++

```
UnicodeString ControlAttributes;
```

Description

This is ControlAttributes, a member of class TLayoutAttributes.

61.1.1.3.2 TLayoutAttributes.LabelAttributes Field

Pascal

```
LabelAttributes: string;
```

C++

```
UnicodeString LabelAttributes;
```

Description

This is LabelAttributes, a member of class TLayoutAttributes.

61.1.1.3.3 TLayoutAttributes.LabelPosition Field

Pascal

```
LabelPosition: TLabelPosition;
```

C++

```
TLabelPosition LabelPosition;
```

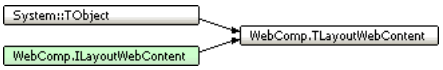
Description

This is LabelPosition, a member of class TLayoutAttributes.

61.1.2 TLayoutWebContent Class

TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
TLayoutWebContent = class(TObject, ILayoutWebContent);
```

C++

```
class TLayoutWebContent : public TObject, public ILayoutWebContent;
```

File

WebComp

Description

TLayoutWebContent is the base class for objects that implement the ILayoutWebContent (see page 6057) interface. Parent components pass a TLayoutWebContent descendant to child Web items as the ParentLayout (see page 5996) parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the TLayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

Do not create instances of TLayoutWebContent. This implementation of the ILayoutWebContent (see page 6057) methods consists of abstract or, in C++ terminology, pure virtual methods. Instead, use a TLayoutWebContent descendant such as TFormLayout or TAdapterFormLayout, which supplies an implementation for those methods.

See Also

TLayout (see page 6085)

IWebContent (see page 6075)

ILayoutWebContent (see page 6057)

TFormLayout

TAdapterFormLayout

61.1.2.1 TLayoutWebContent Members

The following tables list the members exposed by TLayoutWebContent.

Protected Methods

	Name	Description
	_AddRef (see page 5991)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 5991)	Responds when the caller releases a reference to this object's interface.
A	ImplLayoutButton (see page 5991)	Provides the underlying implementation for the LayoutButton (see page 5994) method.
A	ImplLayoutField (see page 5992)	Provides the underlying implementation for the LayoutField (see page 5994) method.
A	ImplLayoutLabelAndField (see page 5992)	Provides the underlying implementation for the LayoutLabelAndField (see page 5994) method.
A	ImplLayoutTable (see page 5993)	Provides the underlying implementation for the LayoutTable (see page 5995) method.
	LayoutButton (see page 5994)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 5994)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 5994)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 5995)	This is LayoutTable, a member of class TLayoutWebContent.
V	QueryInterface (see page 5995)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5990)	Creates a new TLayoutWebContent (see page 5988) object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.

	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ParentLayout (see page 5996)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

61.1.2.2 TLayoutWebContent.Create Constructor

Creates a new TLayoutWebContent (see page 5988) object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TLayoutWebContent(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent (see page 5988) descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout (see page 5996) property.

See Also

ParentLayout (see page 5996)

61.1.2.3 TLayoutWebContent Methods

The methods of the TLayoutWebContent class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5991)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 5991)	Responds when the caller releases a reference to this object's interface.
A	ImplLayoutButton (see page 5991)	Provides the underlying implementation for the LayoutButton (see page 5994) method.
A	ImplLayoutField (see page 5992)	Provides the underlying implementation for the LayoutField (see page 5994) method.
A	ImplLayoutLabelAndField (see page 5992)	Provides the underlying implementation for the LayoutLabelAndField (see page 5994) method.
A	ImplLayoutTable (see page 5993)	Provides the underlying implementation for the LayoutTable (see page 5995) method.
	LayoutButton (see page 5994)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 5994)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 5994)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 5995)	This is LayoutTable, a member of class TLayoutWebContent.
V	QueryInterface (see page 5995)	Returns a reference to a specified interface if the object supports that interface.

61.1.2.3.1 TLayoutWebContent._AddRef Method

Responds when the caller acquires a reference to this object's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

In Delphi, when an application uses an interface that is implemented by TLayoutWebContent (see page 5988) or one of its descendants, it automatically calls __AddRef. Typically, interfaced objects use the __AddRef method to increment a reference count, which is then decremented when the application releases the interface. In this typical scenario, when the reference count drops to zero, the object frees itself.

TLayoutWebContent (see page 5988) implements its own version of the interface methods __AddRef and _Release (see page 5991) that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 5988), __AddRef always returns –1.

See Also

_Release (see page 5991)

_AddRef (see page 3504)

61.1.2.3.2 TLayoutWebContent._Release Method

Responds when the caller releases a reference to this object's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

In Delphi, when an application releases an interface that is implemented by TLayoutWebContent (see page 5988) or one of its descendants, it automatically calls __Release. Typically, interfaced objects use the __Release method to decrement a reference count. When the reference count drops to zero, the object is automatically freed.

TLayoutWebContent (see page 5988) implements its own version of the interface methods _AddRef (see page 5991) and __Release that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 5988), __Release always returns –1.

See Also

_AddRef (see page 5991)

_Release (see page 3504)

61.1.2.3.3 TLayoutWebContent.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton (see page 5994) method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutButton(const UnicodeString HTMLButton,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutButton (see page 5994) method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton (see page 5994) returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent (see page 5988), ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton (see page 5994) method.

See Also

LayoutButton (see page 5994)

61.1.2.3.4 TLayoutWebContent.ImplLayoutField Method

Provides the underlying implementation for the LayoutField (see page 5994) method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutField(const UnicodeString HTMLField,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutField (see page 5994) method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField (see page 5994) returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent (see page 5988), ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField (see page 5994) method.

See Also

LayoutField (see page 5994)

61.1.2.3.5 TLayoutWebContent.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField (see page 5994) method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes) = 0;
```

Description

The LayoutLabelAndField (see page 5994) method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField (see page 5994) returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent (see page 5988), ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField (see page 5994) method.

See Also

LayoutLabelAndField (see page 5994)

61.1.2.3.6 TLayoutWebContent.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable (see page 5995) method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutTable(const UnicodeString HTMLTable,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutTable (see page 5995) method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable (see page 5995) returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent (see page 5988), ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable (see page 5995) method.

See Also

layouttable (see page 5995)

61.1.2.3.7 TLayoutWebContent.LayoutButton Method

Converts the HTML declaring a button into a form the associated Web item can use.

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutButton(const UnicodeString HTMLButton, TLayoutAttributes
Attributes);
```

Description

Call LayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

LayoutButton delegates the task of converting the HTML for the button to the ImplLayoutButton (see page 5991) method.

See Also

ImplLayoutButton (see page 5991)

61.1.2.3.8 TLayoutWebContent.LayoutField Method

Converts the HTML declaring a field into a form the associated Web item can use.

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutField(const UnicodeString HTMLField, TLayoutAttributes
Attributes);
```

Description

Call LayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

LayoutField delegates the task of converting the HTML for the field to the ImplLayoutField (see page 5992) method.

See Also

ImplLayoutField (see page 5992)

61.1.2.3.9 TLayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the associated Web item can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

LayoutLabelAndField delegates the task of converting the HTML for the label and field to the ImplLayoutLabelAndField (see page 5992) method.

See Also

ImplLayoutLabelAndField (see page 5992)

61.1.2.3.10 TLayoutWebContent.LayoutTable Method

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutTable(const UnicodeString HTMLTable, TLayoutAttributes
Attributes);
```

Description

This is LayoutTable, a member of class TLayoutWebContent.

61.1.2.3.11 TLayoutWebContent.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TLayoutWebContent (see page 5988) object supports the interface specified by IID. If so, QueryInterface sets the Obj parameter so that it points to an instance of the specified interface and returns 0 to indicate success.

If the layout object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

IID is the desired interface.

Obj returns a reference to the interface if the TLayoutWebContent (see page 5988) object supports it.

See Also

_AddRef (see page 5991)

_Release (see page 5991)

61.1.2.4 TLayoutWebContent Properties

The properties of the TLayoutWebContent class are listed here.

Public Properties

	Name	Description
	ParentLayout (see page 5996)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

61.1.2.4.1 TLayoutWebContent.ParentLayout Property

Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

Pascal

```
property ParentLayout: TLayout;
```

C++

```
__property TLayout ParentLayout;
```

Description

Web items form a hierarchy, where each child Web item generates its content on behalf of a parent object. This TLayoutWebContent (see page 5988) object is associated with a Web item, and supplies methods to help lay out the content of its child Web items. The associated Web item, in turn, may be the child Web item of another Web item. ParentLayout is the layout object of the associated Web item's parent item.

The TLayoutWebContent (see page 5988) object uses ParentLayout in the implementation of its methods so that they reflect the needs of the parent item as well as the needs of the associated Web item.

ParentLayout supports the ILayoutWebContent (see page 6057) interface. Typically, it is another TLayoutWebContent (see page 5988) descendant.

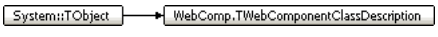
See Also

Create (see page 5990)

ILayoutWebContent (see page 6057)

61.1.3 TWebComponentClassDescription Class

Class Hierarchy



Pascal

```
TWebComponentClassDescription = class(TObject);
```

C++

```
class TWebComponentClassDescription : public TObject;
```

File

WebComp

Description

This is class WebComp.TWebComponentClassDescription.

61.1.3.1 TWebComponentClassDescription Members

The following tables list the members exposed by TWebComponentClassDescription.

Public Fields

	Name	Description
	ComponentClass (see page 5998)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 5998)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 5998)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 5999)	This is FrameClass, a member of class TWebComponentClassDescription.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.

	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

61.1.3.2 TWebComponentClassDescription Fields

The fields of the TWebComponentClassDescription class are listed here.

Public Fields

	Name	Description
	ComponentClass (see page 5998)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 5998)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 5998)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 5999)	This is FrameClass, a member of class TWebComponentClassDescription.

61.1.3.2.1 TWebComponentClassDescription.ComponentClass Field

Pascal

```
ComponentClass: TComponentClass;
```

C++

```
TComponentClass ComponentClass;
```

Description

This is ComponentClass, a member of class TWebComponentClassDescription.

61.1.3.2.2 TWebComponentClassDescription.ComponentClassName Field

Pascal

```
ComponentClassName: string;
```

C++

```
UnicodeString ComponentClassName;
```

Description

This is ComponentClassName, a member of class TWebComponentClassDescription.

61.1.3.2.3 TWebComponentClassDescription.Editor Field

Pascal

```
Editor: TObject;
```

C++

```
TObject * Editor;
```

Description

This is Editor, a member of class TWebComponentClassDescription.

61.1.3.2.4 TWebComponentClassDescription.FrameClass Field

Pascal

```
FrameClass: Boolean;
```

C++

```
Boolean FrameClass;
```

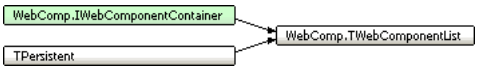
Description

This is FrameClass, a member of class TWebComponentClassDescription.

61.1.4 TWebComponentList Class

TWebComponentList maintains a list of Web item components.

Class Hierarchy



Pascal

```
TWebComponentList = class(TPersistent, IWebComponentContainer);
```

C++

```
class TWebComponentList : public TPersistent, public IWebComponentContainer;
```

File

WebComp

Description

TWebComponentList provides the structure for the hierarchy of Web items that generate HTML documents in a WebSnap or InternetExpress application. At the top of the hierarchy is a page producer (TINetXPageProducer or TAdapterPageProducer) or an adapter. The page producer assembles an HTML document, or the adapter provides information and performs tasks on behalf of an adapter page producer or server-side script. To perform their tasks, the root objects use a set of helper components. Each of these components, in turn, uses one or more helper components, and so on. Each component that uses helper components maintains a list of its helper components. That list is a TWebComponentList object.

Each component in the list must support the IWebComponent (see page 6065) interface. TWebComponentList uses this interface to coordinate the objects it manages. In addition, children of a page producer should support the IWebContent (see page 6075) interface. IWebContent (see page 6075) allows the parent component to extract HTML generated by its children.

Use the properties and methods of TWebComponentList to

Add (see page 6002) or delete the components in the list.

Locate and access components in the list.

Rearrange components in the list.

Access the parent component that uses the list.

See Also

IWebContent (see page 6075)

TBasePageItemsProducer

TWebControlGroup

TWebForm

TInetXPageProducer

IWebComponent (see page 6065)

TAdapterPageProducer (see page 1076)

61.1.4.1 TWebComponentList Members

The following tables list the members exposed by TWebComponentList.

Protected Methods

	Name	Description
	_AddRef (see page 6002)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 6002)	This is _Release, a member of class TWebComponentList.
	Changed (see page 6003)	This is Changed, a member of class TWebComponentList.
	GetComponent (see page 6003)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 6004)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 6004)	Returns the owner of an object.
V	NewComponent (see page 6005)	This is NewComponent, a member of class TWebComponentList.
V	QueryInterface (see page 6005)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 6007)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 6007)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Events

	Name	Description
	OnChange (see page 6009)	Occurs when a change occurs in the web component list.

Public Methods

	Name	Description
	Add (see page 6002)	Inserts a new component at the end of the list.
	Clear (see page 6003)	Deletes all components from the list.
	Create (see page 6001)	Creates and initializes new TWebComponentList (see page 5999) object.
V	Destroy (see page 6001)	Frees an instance of TWebComponentList (see page 5999).
	GetComponentCount (see page 6072)	This is GetComponentCount, a member of class IWebComponentContainer.
	IndexOf (see page 6004)	Returns the index of a specified component in the WebComponents (see page 6008) array.
	LockNotify (see page 6005)	This is LockNotify, a member of class TWebComponentList.
	NotifyLocked (see page 6005)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 6005)	Removes a component from the WebComponents (see page 6008) array.
	SetComponentIndex (see page 6006)	Moves a component to a new position in the WebComponents (see page 6008) array.
	UnlockNotify (see page 6006)	This is UnlockNotify, a member of class TWebComponentList.

Public Properties

	Name	Description
R	ComponentCount (see page 6072)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 6073)	This is Components, a member of class IWebComponentContainer.
R	Count (see page 6007)	Indicates the number of components in the list.

R	ParentComponent (see page 6008)	Identifies the component that uses this Web component list to maintain its Web Items.
R	WebComponents (see page 6008)	Lists the Web items in the list.

61.1.4.2 TWebComponentList.Create Constructor

Creates and initializes new TWebComponentList (see page 5999) object.

Pascal

```
constructor Create(AParentComponent: TComponent);
```

C++

```
__fastcall TWebComponentList(TComponent * AParentComponent);
```

Description

Call Create to instantiate TWebComponentList (see page 5999) at runtime. Most applications do not need to instantiate TWebComponentList (see page 5999). Instead, the Web items that use TWebComponentList (see page 5999) to manage their own Web items instantiate a TWebComponentList (see page 5999) from their constructors.

61.1.4.3 TWebComponentList.Destroy Destructor

Frees an instance of TWebComponentList (see page 5999).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebComponentList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TWebComponentList (see page 5999) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of WebComponents (see page 6008).

Note: Destroy does not free the components in the list.

See Also

System::TObject::Free (see page 3382)

61.1.4.4 TWebComponentList Methods

The methods of the TWebComponentList class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6002)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 6002)	This is _Release, a member of class TWebComponentList.
	Changed (see page 6003)	This is Changed, a member of class TWebComponentList.
	GetComponent (see page 6003)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 6004)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 6004)	Returns the owner of an object.

	NewComponent (see page 6005)	This is NewComponent, a member of class TWebComponentList.
	QueryInterface (see page 6005)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 6007)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 6007)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Methods

	Name	Description
	Add (see page 6002)	Inserts a new component at the end of the list.
	Clear (see page 6003)	Deletes all components from the list.
	IndexOf (see page 6004)	Returns the index of a specified component in the WebComponents (see page 6008) array.
	LockNotify (see page 6005)	This is LockNotify, a member of class TWebComponentList.
	NotifyLocked (see page 6005)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 6005)	Removes a component from the WebComponents (see page 6008) array.
	SetComponentIndex (see page 6006)	Moves a component to a new position in the WebComponents (see page 6008) array.
	UnlockNotify (see page 6006)	This is UnlockNotify, a member of class TWebComponentList.

61.1.4.4.1 TWebComponentList._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentList.

61.1.4.4.2 TWebComponentList._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentList.

61.1.4.4.3 TWebComponentList.Add Method

Inserts a new component at the end of the list.

Pascal

```
procedure Add(Component: TComponent);
```

C++

```
__fastcall Add(TComponent * Component);
```

Description

Call Add to insert a component at the end of the WebComponents (see page 6008) array.

Add increments the Count (see page 6007) property. After calling Add, the value of Count (see page 6007) is one greater than the index of the inserted component in the WebComponents (see page 6008) array. To move this component to another position, use the SetComponentIndex (see page 6006) method.

See Also

WebComponents (see page 6008)

Count (see page 6007)

SetComponentIndex (see page 6006)

Remove (see page 6005)

61.1.4.4.4 TWebComponentList.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TWebComponentList.

61.1.4.4.5 TWebComponentList.Clear Method

Deletes all components from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the WebComponents (see page 6008) array and set Count (see page 6007) to 0.

See Also

Count (see page 6007)

Remove (see page 6005)

WebComponents (see page 6008)

61.1.4.4.6 TWebComponentList.GetComponent Method

Pascal

```
function GetComponent(Index: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int Index);
```

Description

This is GetComponent, a member of class TWebComponentList.

61.1.4.4.7 TWebComponentList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebComponentList.

61.1.4.4.8 TWebComponentList.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

61.1.4.4.9 TWebComponentList.IndexOf Method

Returns the index of a specified component in the WebComponents (see page 6008) array.

Pascal

```
function IndexOf(Component: TComponent): Integer;
```

C++

```
__fastcall int IndexOf(TComponent * Component);
```

Description

Call IndexOf to get the index for a component in the WebComponents (see page 6008) array. Specify the desired component as the Component parameter.

The first component in the array has index 0, the second component has index 1, and so on. If a component is not in the list, IndexOf returns -1. If a component appears more than once in the array, IndexOf returns the index of the first appearance.

See Also

WebComponents (see page 6008)

61.1.4.4.10 TWebComponentList.LockNotify Method

Pascal

```
procedure LockNotify;
```

C++

```
__fastcall LockNotify();
```

Description

This is LockNotify, a member of class TWebComponentList.

61.1.4.4.11 TWebComponentList.NewComponent Method

Pascal

```
function NewComponent(AOwner: TComponent): TComponent; virtual;
```

C++

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

Description

This is NewComponent, a member of class TWebComponentList.

61.1.4.4.12 TWebComponentList.NotifyLocked Method

Pascal

```
function NotifyLocked: Boolean;
```

C++

```
__fastcall Boolean NotifyLocked();
```

Description

This is NotifyLocked, a member of class TWebComponentList.

61.1.4.4.13 TWebComponentList.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentList.

61.1.4.4.14 TWebComponentList.Remove Method

Removes a component from the WebComponents (see page 6008) array.

Pascal

```
procedure Remove(Component: TComponent);
```

C++

```
__fastcall Remove(TComponent * Component);
```

Description

Call Remove to remove a specific component from the WebComponents (see page 6008) array. After a component is removed, all the WebComponents (see page 6008) that follow it are moved up in index position and Count (see page 6007) is reduced by one.

See Also

Add (see page 6002)

Count (see page 6007)

Clear (see page 6003)

WebComponents (see page 6008)

61.1.4.4.15 TWebComponentList.SetComponentIndex Method

Moves a component to a new position in the WebComponents (see page 6008) array.

Pascal

```
procedure SetComponentIndex(Component: TComponent; Value: Integer);
```

C++

```
__fastcall SetComponentIndex(TComponent * Component, int Value);
```

Description

Call SetComponentIndex to move a component to a new position in the list. Component specifies the component to move. Value specifies the index it should have after it is moved. If Value is less than 0, the component is moved to the first position. If Value is greater than or equal to count, the component is moved to the last position.

See Also

WebComponents (see page 6008)

61.1.4.4.16 TWebComponentList.UnlockNotify Method

Pascal

```
procedure UnlockNotify;
```

C++

```
__fastcall UnlockNotify();
```

Description

This is UnlockNotify, a member of class TWebComponentList.

61.1.4.4.17 TWebComponentList.WebComponentContainer_GetComponent Method

Pascal

```
function WebComponentContainer_GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * WebComponentContainer_GetComponent(int I);
```

Description

This is WebComponentContainer_GetComponent, a member of class TWebComponentList.

61.1.4.4.18 TWebComponentList.WebComponentContainer_GetComponentCount Method

Pascal

```
function WebComponentContainer_GetComponentCount: Integer;
```

C++

```
__fastcall int WebComponentContainer_GetComponentCount();
```

Description

This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

61.1.4.5 TWebComponentList Properties

The properties of the TWebComponentList class are listed here.

Public Properties

	Name	Description
	Count (see page 6007)	Indicates the number of components in the list.
	ParentComponent (see page 6008)	Identifies the component that uses this Web component list to maintain its Web Items.
	WebComponents (see page 6008)	Lists the Web items in the list.

61.1.4.5.1 TWebComponentList.Count Property

Indicates the number of components in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to determine the number of components in the WebComponents (see page 6008) array. Count is a read-only property. To change the size of the WebComponents (see page 6008) array, add or remove components using the Add (see page 6002) and Remove (see page 6005) methods.

See Also

WebComponents (see page 6008)

Add (see page 6002)

Remove (see page 6005)

61.1.4.5.2 TWebComponentList.ParentComponent Property

Identifies the component that uses this Web component list to maintain its Web Items.

Pascal

```
property ParentComponent: TComponent;
```

C++

```
__property TComponent * ParentComponent;
```

Description

Read (see page 3727) ParentComponent to access the parent of the Web Items in the list. The value of ParentComponent is assigned in the constructor when the parent first instantiates the Web component list.

See Also

Create (see page 6001)

61.1.4.5.3 TWebComponentList.WebComponents Property

Lists the Web items in the list.

Pascal

```
property WebComponents [Index: Integer]: TComponent;
```

C++

```
__property TComponent * WebComponents[int Index];
```

Description

Use WebComponents to access the Web components listed by TWebComponentList (see page 5999). Use WebComponents with the Count (see page 6007) property to iterate through all of the components in the list. Use the Add (see page 6002) or Remove (see page 6005) method to add or remove components from the list.

WebComponents is the default property for TWebComponentList (see page 5999). This means that the property name can be omitted when accessing a component in the list. Thus, the line

```
WebItem := InetXPageProducer1.WebPageItems.WebComponents[0];
WebItem = InetXPageProducer1->WebPageItems->WebComponents[0];
```

can be written

```
WebItem := InetXPageProducer1.WebPageItems[0];
WebItem = (*(InetXPageProducer1->WebPageItems))[0];
```

See Also

Count (see page 6007)

Add (see page 6002)

Remove (see page 6005)

operator_sb

61.1.4.6 TWebComponentList Events

The events of the TWebComponentList class are listed here.

Public Events

	Name	Description
	OnChange (see page 6009)	Occurs when a change occurs in the web component list.

61.1.4.6.1 TWebComponentList.OnChange Event

Occurs when a change occurs in the web component list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

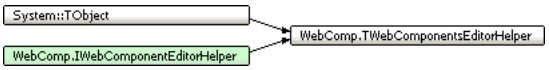
```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write an event handler to process an action when a change occurs in the web component list, such as when a new component is added.

61.1.5 TWebComponentsEditorHelper Class

Class Hierarchy



Pascal

```
TWebComponentsEditorHelper = class(TObject, IWebComponentEditorHelper);
```

C++

```
class TWebComponentsEditorHelper : public TObject, public IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.TWebComponentsEditorHelper.

61.1.5.1 TWebComponentsEditorHelper Members

The following tables list the members exposed by TWebComponentsEditorHelper.

Protected Methods

	Name	Description
	_AddRef (see page 6011)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 6011)	This is _Release, a member of class TWebComponentsEditorHelper.

	CanAddClassHelper (see page 6011)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
	ImplCanAddClassHelper (see page 6011)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
	QueryInterface (see page 6011)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

61.1.5.2 TWebComponentsEditorHelper Methods

The methods of the TWebComponentsEditorHelper class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6011)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 6011)	This is _Release, a member of class TWebComponentsEditorHelper.
	CanAddClassHelper (see page 6011)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
	ImplCanAddClassHelper (see page 6011)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
	QueryInterface (see page 6011)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

61.1.5.2.1 TWebComponentsEditorHelper._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentsEditorHelper.

61.1.5.2.2 TWebComponentsEditorHelper._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentsEditorHelper.

61.1.5.2.3 TWebComponentsEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

Description

This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.

61.1.5.2.4 TWebComponentsEditorHelper.ImplCanAddClassHelper Method

Pascal

```
function ImplCanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClassHelper(TComponent * AEditor, TComponent * AParent,
TClass AClass);
```

Description

This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.

61.1.5.2.5 TWebComponentsEditorHelper.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentsEditorHelper.

61.1.6 TWebContainedComponent Class

TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.

Class Hierarchy

Pascal

```
TWebContainedComponent = class(TComponent, IWebComponent);
```

C++

```
class TWebContainedComponent : public TComponent, public IWebComponent;
```

File

WebComp

Description

Use TWebContainedComponent as a base class for Web items. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 5999) object that lists a set of its Web items.

TWebContainedComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedComponent descendants act as leaf nodes in this hierarchy. That is, they do not have their own Web items that they use to generate HTML.

In order to allow TWebComponentList (see page 5999) to manage the Web item, TWebContainedComponent implements the IWebComponent (see page 6065) interface.

Do not create instances of TWebContainedComponent. TWebContainedComponent does not produce any HTML, but merely handles the relationship to a managing TWebComponentList (see page 5999) object.

See Also

TWebComponentList (see page 5999)

IWebComponent (see page 6065)

61.1.6.1 TWebContainedComponent Members

The following tables list the members exposed by TWebContainedComponent.

Protected Methods

	Name	Description
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.

	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

61.1.6.2 TWebContainedComponent.Create Constructor

Creates and initializes an instance of TWebContainedComponent (see page 6012).

Pascal

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

C++

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

Description

Call Create to instantiate a contained component at runtime. Web items added at design time are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedComponent (see page 6012) instance.

See Also

Owner

61.1.6.3 TWebContainedComponent.Destroy Destructor

Frees an instance of TWebContainedComponent (see page 6012).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedComponent (see page 6012).

61.1.6.4 TWebContainedComponent Methods

The methods of the TWebContainedComponent class are listed here.

Protected Methods

	Name	Description
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Public Methods

	Name	Description
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.

61.1.6.4.1 TWebContainedComponent.GetContainerList Method

Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 6015)

WebParent (see page 6018)

61.1.6.4.2 TWebContainedComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedComponent (see page 6012) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 6018) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 6015)

WebParent (see page 6018)

61.1.6.4.3 TWebContainedComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedComponent (see page 6012) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 6018) property is not nil (Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 6015)

WebParent (see page 6018)

61.1.6.4.4 TWebContainedComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 6012) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 5999) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 6014)

WebParent (see page 6018)

TWebComponentList (see page 5999)

61.1.6.4.5 TWebContainedComponent.ParentChanged Method

Responds when the WebParent (see page 6018) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 6018) property changes. As implemented in TWebContainedComponent (see page 6012), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 6018)

61.1.6.4.6 TWebContainedComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedComponent (see page 6012) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 6018) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 6015)

WebParent (see page 6018)

61.1.6.4.7 TWebContainedComponent.SetParentComponent Method

Sets the WebParent (see page 6018) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedComponent (see page 6012) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 6018) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 6015)

WebParent (see page 6018)

61.1.6.4.8 TWebContainedComponent.SetWebParent Method

Sets the value of the WebParent (see page 6018) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 6018) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 5999) object that implements a property on the new Web parent.

See Also

WebParent (see page 6018)

Add (see page 6002)

61.1.6.5 TWebContainedComponent Properties

The properties of the TWebContainedComponent class are listed here.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

61.1.6.5.1 TWebContainedComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedComponent (see page 6012) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 6065) interface. In addition to providing the implementation for the IWebComponent (see page 6065) interface, WebContainedComponentHelper provides the underlying implementation for most of the other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 6065)

61.1.6.5.2 TWebContainedComponent.WebParent Property

Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedComponent (see page 6012) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 5999) object that includes this TWebContainedComponent (see page 6012) object to represent its Web items.

See Also

TWebComponentList (see page 5999)

61.1.7 TWebContainedComponentHelper Class

TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.

Class Hierarchy

Pascal

```
TWebContainedComponentHelper = class(TObject);
```

C++

```
class TWebContainedComponentHelper : public TObject;
```

File

WebComp

Description

Components that generate HTML for another component use TWebContainedComponentHelper to provide the underlying implementation of many of their properties and methods. TWebContainedComponentHelper implements the IWebComponent (see page 6065) interface, which allows a TWebComponentList (see page 5999) object manage a list of child components. In addition, it implements a number of properties and methods that allow a child Web item to interact with its parent item.

See Also

IWebComponent (see page 6065)

TWebContainedComponent (see page 6012)

TWebContainedContainerComponent (see page 6026)

61.1.7.1 TWebContainedComponentHelper Members

The following tables list the members exposed by TWebContainedComponentHelper.

Protected Events

	Name	Description
	OnParentChanged (see page 6025)	Occurs when the WebParent (see page 6025) property changes.

Protected Methods

	Name	Description
	GetContainer (see page 6021)	Returns the value of the WebParent (see page 6025) property.
	GetIndex (see page 6022)	Returns the position of the associated Web item in the parent item's array of Web items.
	ParentChanged (see page 6023)	Generates an OnParentChanged (see page 6025) event.
	SetComponentList (see page 6023)	Sets the value of the ContainerList (see page 6025) property.
	SetContainer (see page 6023)	Sets the value of the WebParent (see page 6025) property.
	SetIndex (see page 6024)	Sets the position of the associated component in the list specified by ContainerList (see page 6025).
	SetWebParent (see page 6024)	Sets the value of the WebParent (see page 6025) property.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6020)	Creates an instance of TWebContainedComponentHelper (see page 6018).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6021)	Destroys an instance of TWebContainedComponentHelper (see page 6018).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	IsDefaultField (see page 6022)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ContainerList (see page 6025)	Indicates the list that contains the associated component.
R	WebParent (see page 6025)	Indicates the component that uses the list of Web items that includes the associated component.

61.1.7.2 TWebContainedComponentHelper.Create Constructor

Creates an instance of TWebContainedComponentHelper (see page 6018).

Pascal

```
constructor Create(AComponent: TComponent; AOnParentChanged: TNotifyEvent);
```

C++

```
__fastcall TWebContainedComponentHelper(TComponent * AComponent, TNotifyEvent
AOnParentChanged);
```

Description

Applications do not create instances of TWebContainedComponentHelper (see page 6018). The component that uses TWebContainedComponentHelper (see page 6018) so that it can appear as a child item in a TWebComponentList (see page 5999) object instantiates TWebContainedComponentHelper (see page 6018).

AComponent is the component that uses TWebContainedComponentHelper (see page 6018) to implement those properties and methods that allow it to appear in a TWebComponentList (see page 5999) object.

AOnParentChanged is an event handler that AComponent supplies so that it can update itself when it is moved to the list of a different parent component. This becomes the event handler for the OnParentChanged (see page 6025) event.

See Also

OnParentChanged (see page 6025)

61.1.7.3 TWebContainedComponentHelper.Destroy Destructor

Destroys an instance of TWebContainedComponentHelper (see page 6018).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainedComponentHelper (see page 6018) object is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

61.1.7.4 TWebContainedComponentHelper Methods

The methods of the TWebContainedComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetContainer (see page 6021)	Returns the value of the WebParent (see page 6025) property.
	GetIndex (see page 6022)	Returns the position of the associated Web item in the parent item's array of Web items.
V	ParentChanged (see page 6023)	Generates an OnParentChanged (see page 6025) event.
	SetComponentList (see page 6023)	Sets the value of the ContainerList (see page 6025) property.
	SetContainer (see page 6023)	Sets the value of the WebParent (see page 6025) property.
	SetIndex (see page 6024)	Sets the position of the associated component in the list specified by ContainerList (see page 6025).
	SetWebParent (see page 6024)	Sets the value of the WebParent (see page 6025) property.

Public Methods

	Name	Description
	IsDefaultField (see page 6022)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

61.1.7.4.1 TWebContainedComponentHelper.GetContainer Method

Returns the value of the WebParent (see page 6025) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

Use GetContainer to access the component that uses this helper object's associated component to assemble its HTML output. The component that this method returns uses the list specified by ContainerList (see page 6025) to manage its list of Web items.

See Also

WebParent (see page 6025)

ContainerList (see page 6025)

61.1.7.4.2 TWebContainedComponentHelper.GetIndex Method

Returns the position of the associated Web item in the parent item's array of Web items.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex returns the index of the associated component in the list that is the value of the ContainerList (see page 6025) property. The index of the first item is 0, the index of the second item is 1, and so on.

See Also

ContainerList (see page 6025)

61.1.7.4.3 TWebContainedComponentHelper.IsDefaultField Method

Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean;
```

C++

```
__fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponentHelper (see page 6018) uses IsDefaultField internally to locate the property on the Web parent that lists the associated item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 5999) that contains the associated item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if the associated component does not belong to such a default list.

See Also

ContainerList (see page 6025)

WebParent (see page 6025)

TWebComponentList (see page 5999)

61.1.7.4.4 TWebContainedComponentHelper.ParentChanged Method

Generates an OnParentChanged (see page 6025) event.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called automatically when the WebParent (see page 6025) property changes. It generates an OnParentChanged (see page 6025) event, which is handled by the event handler that the associated component passes to the TWebContainedComponentHelper (see page 6018) constructor.

See Also

OnParentChanged (see page 6025)

Create (see page 6020)

61.1.7.4.5 TWebContainedComponentHelper.SetComponentList Method

Sets the value of the ContainerList (see page 6025) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList assigns the object specified by List as the value of the ContainerList (see page 6025) property. Note that List must be an instance of TWebComponentList (see page 5999) (or nil (Delphi) or NULL (C++)).

See Also

ContainerList (see page 6025)

61.1.7.4.6 TWebContainedComponentHelper.SetContainer Method

Sets the value of the WebParent (see page 6025) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer assigns a new value to the WebParent (see page 6025) property.

Container is the new Web parent.

See Also

WebParent (see page 6025)

61.1.7.4.7 TWebContainedComponentHelper.SetIndex Method

Sets the position of the associated component in the list specified by ContainerList (see page 6025).

Pascal

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

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex moves the associated component to a specified position in the list of Web items that contains it.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

See Also

ContainerList (see page 6025)

61.1.7.4.8 TWebContainedComponentHelper.SetWebParent Method

Sets the value of the WebParent (see page 6025) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent moves the associated component to the component list of a new parent. To do so, it removes the associated component from the current parent's list of Web items, adds it to the new Web parent's list of Web items, and then generates an OnParentChanged (see page 6025) event.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

Value is the new Web parent.

See Also

WebParent (see page 6025)

OnParentChanged (see page 6025)

ContainerList (see page 6025)

61.1.7.5 TWebContainedComponentHelper Properties

The properties of the TWebContainedComponentHelper class are listed here.

Public Properties

	Name	Description
	ContainerList (see page 6025)	Indicates the list that contains the associated component.

	WebParent (see page 6025)	Indicates the component that uses the list of Web items that includes the associated component.

61.1.7.5.1 TWebContainedComponentHelper.ContainerList Property

Indicates the list that contains the associated component.

Pascal

```
property ContainerList: TWebComponentList;
```

C++

```
__property TWebComponentList ContainerList;
```

Description

Use ContainerList to access the list in which the associated component appears. The parent item uses the components in the list to generate HTML, which it then assembles into its own HTML output.

See Also

WebParent (see page 6025)

61.1.7.5.2 TWebContainedComponentHelper.WebParent Property

Indicates the component that uses the list of Web items that includes the associated component.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

WebParent specifies the component on whose behalf the associated component generates HTML. WebParent is the component that uses the list that is the value of the ContainerList (see page 6025) property.

See Also

ContainerList (see page 6025)

61.1.7.6 TWebContainedComponentHelper Events

The events of the TWebContainedComponentHelper class are listed here.

Protected Events

	Name	Description
	OnParentChanged (see page 6025)	Occurs when the WebParent (see page 6025) property changes.

61.1.7.6.1 TWebContainedComponentHelper.OnParentChanged Event

Occurs when the WebParent (see page 6025) property changes.

Pascal

```
property OnParentChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnParentChanged;
```

Description

Components that use TWebContainedComponentHelper (see page 6018) so that they can act as a child Web item in a hierarchy of Web items assign an event handler to OnParentChanged.

The associated component uses TWebContainedComponentHelper (see page 6018) to implement the IWebComponent (see page 6065) interface. This interface can be used to change the value of WebParent (see page 6025). By assigning an OnParentChanged event handler, the associated component can update itself when it moves from one parent item to another.

See Also

WebParent (see page 6025)

IWebComponent (see page 6065)

61.1.8 TWebContainedContainerComponent Class

TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainedContainerComponent = class(TComponent, IGetWebComponentList, IWebComponent);
```

C++

```
class TWebContainedContainerComponent : public TComponent, public IGetWebComponentList, public
IWebComponent;
```

File

WebComp

Description

Use TWebContainedContainerComponent as a base class for Web items that have their own subitems. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 5999) object that lists a set of its Web items. In addition, the TWebContainedContainerComponent descendant uses a TWebComponentList (see page 5999) object to store a set of Web items that it uses.

TWebContainedContainerComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedContainerComponent descendants act as internal nodes in this hierarchy. That is, they have both a Web parent and a set of child Web items.

In order to allow the parent's TWebComponentList (see page 5999) object to manage this Web item, TWebContainedContainerComponent implements the IWebComponent (see page 6065) interface.

Do not create instances of TWebContainedContainerComponent. TWebContainedContainerComponent does not produce any HTML, but merely handles the relationships to parent and child TWebComponentList (see page 5999) objects.

See Also

TWebComponentList (see page 5999)

IWebComponent (see page 6065)

61.1.8.1 TWebContainedContainerComponent Members

The following tables list the members exposed by TWebContainedContainerComponent.

Protected Methods

	Name	Description
V	GetChildren (see page 6029)	Executes a callback for each child Web item.
	GetContainerList (see page 6029)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 6030)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6031)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplGetDefaultComponentList (see page 6031)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	IsDefaultField (see page 6032)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 6032)	Responds when the WebParent (see page 6036) property changes.
V	ReadState (see page 6033)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetChildOrder (see page 6033)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 6033)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetParentComponent (see page 6034)	Sets the WebParent (see page 6036) property to be the same as this component's parent.
	SetWebParent (see page 6034)	Sets the value of the WebParent (see page 6036) property.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 6035)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebContainerComponentHelper (see page 6035)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 6036)	Indicates the component that uses this TWebContainedContainerComponent (see page 6026) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6028)	Creates and initializes an instance of TWebContainedContainerComponent (see page 6026).
V	Destroy (see page 6028)	Frees an instance of TWebContainedContainerComponent (see page 6026).
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
V	GetParentComponent (see page 6030)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6031)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

61.1.8.2 TWebContainedContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainedContainerComponent (see page 6026).

Pascal

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

C++

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

Description

Call Create to instantiate a contained component at runtime. Web items added at design time using the Web Page editor are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedContainerComponent (see page 6026) instance.

See Also

Owner

61.1.8.3 TWebContainedContainerComponent.Destroy Destructor

Frees an instance of TWebContainedContainerComponent (see page 6026).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedContainerComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedContainerComponent (see page 6026).

61.1.8.4 TWebContainedContainerComponent Methods

The methods of the TWebContainedContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 6029)	Executes a callback for each child Web item.
	GetContainerList (see page 6029)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.

	GetDefaultWebComponents (see page 6030)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6031)	Returns the list of Web items that this item uses to generate its HTML.
	ImplGetDefaultComponentList (see page 6031)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	IsDefaultField (see page 6032)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 6032)	Responds when the WebParent (see page 6036) property changes.
	ReadState (see page 6033)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetChildOrder (see page 6033)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6033)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetParentComponent (see page 6034)	Sets the WebParent (see page 6036) property to be the same as this component's parent.
	SetWebParent (see page 6034)	Sets the value of the WebParent (see page 6036) property.

Public Methods

	Name	Description
	GetParentComponent (see page 6030)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6031)	Indicates whether this component appears in another component's list of Web items.

61.1.8.4.1 TWebContainedContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainedContainerComponent (see page 6026) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that GetWebComponents (see page 6031) returns, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the TWebContainedContainerComponent (see page 6026) object.

See Also

GetWebComponents (see page 6031)

GetParentComponent

SetChildOrder (see page 6033)

61.1.8.4.2 TWebContainedContainerComponent.GetContainerList Method

Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 6032)

WebParent (see page 6036)

61.1.8.4.3 TWebContainedContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this Web item. As implemented in TWebContainedContainerComponent (see page 6026), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 6031)

IsDefaultField (see page 6032)

61.1.8.4.4 TWebContainedContainerComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedContainerComponent (see page 6026) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 6036) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 6031)

WebParent (see page 6036)

61.1.8.4.5 TWebContainedContainerComponent.GetWebComponents Method

Returns the list of Web items that this item uses to generate its HTML.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 5999) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 6030)

61.1.8.4.6 TWebContainedContainerComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedContainerComponent (see page 6026) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 6036) property is not nil (Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 6030)

WebParent (see page 6036)

61.1.8.4.7 TWebContainedContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the

virtual GetDefaultWebComponents (see page 6030) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 6030)

IsDefaultField (see page 6032)

61.1.8.4.8 TWebContainedContainerComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 6012) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 5999) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 6029)

WebParent (see page 6036)

TWebComponentList (see page 5999)

GetDefaultWebComponents (see page 6030)

GetWebComponents (see page 6031)

61.1.8.4.9 TWebContainedContainerComponent.ParentChanged Method

Responds when the WebParent (see page 6036) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 6036) property changes. As implemented in TWebContainedContainerComponent (see page 6026), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 6036)

61.1.8.4.10 TWebContainedContainerComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedContainerComponent (see page 6026) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 6036) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 6030)

WebParent (see page 6036)

61.1.8.4.11 TWebContainedContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 6031) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 6031)

SetChildOrder

61.1.8.4.12 TWebContainedContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 6030) method.

As implemented in TWebContainedContainerComponent (see page 6026), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 6030)

IsDefaultField (see page 6032)

61.1.8.4.13 TWebContainedContainerComponent.SetParentComponent Method

Sets the WebParent (see page 6036) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedContainerComponent (see page 6026) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 6036) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 6031)

WebParent (see page 6036)

61.1.8.4.14 TWebContainedContainerComponent.SetWebParent Method

Sets the value of the WebParent (see page 6036) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 6036) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 5999) object that implements a property on the new Web parent.

See Also

WebParent (see page 6036)

Add (see page 6002)

61.1.8.5 TWebContainedContainerComponent Properties

The properties of the TWebContainedContainerComponent class are listed here.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6035)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebContainerComponentHelper (see page 6035)	Provides access to the helper object that allows this object to manage sub Web items.
	WebParent (see page 6036)	Indicates the component that uses this TWebContainedContainerComponent (see page 6026) as one of its Web items.

61.1.8.5.1 TWebContainedContainerComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedContainerComponent (see page 6026) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 6065) interface. In addition to providing the implementation for the IWebComponent (see page 6065) interface, WebContainedComponentHelper provides the underlying implementation for other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 6065)

61.1.8.5.2 TWebContainedContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage sub Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainedContainerComponent (see page 6026) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign sub Web items. The GetDefaultWebComponents (see page 6030) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 6031) method returns this list.

See Also

GetDefaultWebComponents (see page 6030)

GetWebComponents (see page 6031)

61.1.8.5.3 TWebContainedContainerComponent.WebParent Property

Indicates the component that uses this TWebContainedContainerComponent (see page 6026) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedContainerComponent (see page 6026) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 5999) object that includes this TWebContainedContainerComponent (see page 6026) object to represent its child Web items.

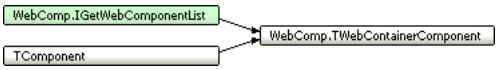
See Also

TWebComponentList (see page 5999)

61.1.9 TWebContainerComponent Class

TWebContainerComponent is the base class for components that use Web items to generate HTML.

Class Hierarchy



Pascal

```
TWebContainerComponent = class(TComponent, IGetWebComponentList);
```

C++

```
class TWebContainerComponent : public TComponent, public IGetWebComponentList;
```

File

WebComp

Description

Use TWebContainerComponent as a base class for objects that maintain a collection of Web items but that do not have a parent Web item. TWebContainerComponent uses a TWebComponentList (see page 5999) object to store the set of child items that it manages.

Do not create instances of TWebContainerComponent, it is intended only as a base class. It only provides support for storing and retrieving child Web items.

See Also

TWebComponentList (see page 5999)

IWebComponent (see page 6065)

TWebContainedContainerComponent (see page 6026)

61.1.9.1 TWebContainerComponent Members

The following tables list the members exposed by TWebContainerComponent.

Protected Methods

	Name	Description
	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetDefaultWebComponents (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6040)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Protected Properties

	Name	Description
	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	Create (see page 6037)	Creates and initializes an instance of TWebContainerComponent (see page 6036).
	Destroy (see page 6038)	Frees an instance of TWebContainerComponent (see page 6036).
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

61.1.9.2 TWebContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainerComponent (see page 6036).

Pascal

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

C++

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

Description

Call Create to instantiate a contained component at runtime.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainerComponent (see page 6036) instance.

See Also

Owner

61.1.9.3 TWebContainerComponent.Destroy Destructor

Frees an instance of TWebContainerComponent (see page 6036).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponent();
```

Description

Destroy frees the memory used by TWebContainerComponent (see page 6036).

61.1.9.4 TWebContainerComponent Methods

The methods of the TWebContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 6038)	Executes a callback for each child Web item.
	GetDefaultWebComponents (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6039)	Returns the list of Web items that this item manages.
	ImplGetDefaultComponentList (see page 6039)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetChildOrder (see page 6040)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6040)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

61.1.9.4.1 TWebContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainerComponent (see page 6036) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that GetWebComponents (see page 6039) returns, as long as that component has Root for an Owner. This allows the child Web

items to be loaded and saved with the TWebContainerComponent (see page 6036) object.

See Also

GetWebComponents (see page 6039)

GetParentComponent

SetChildOrder (see page 6040)

61.1.9.4.2 TWebContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TWebContainerComponent (see page 6036), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 6039)

IsDefaultField (see page 6015)

61.1.9.4.3 TWebContainerComponent.GetWebComponents Method

Returns the list of Web items that this item manages.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 5999) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 6039)

61.1.9.4.4 TWebContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the virtual GetDefaultWebComponents (see page 6039) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 6039)

IsDefaultField (see page 6015)

61.1.9.4.5 TWebContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 6039) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 6039)

SetChildOrder

61.1.9.4.6 TWebContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 6039) method.

As implemented in TWebContainerComponent (see page 6036), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 6039)

IsDefaultField (see page 6015)

61.1.9.5 TWebContainerComponent Properties

The properties of the TWebContainerComponent class are listed here.

Protected Properties

	Name	Description
	WebContainerComponentHelper (see page 6041)	Provides access to the helper object that allows this object to manage its Web items.

61.1.9.5.1 TWebContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage its Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainerComponent (see page 6036) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign child Web items. The GetDefaultWebComponents (see page 6039) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 6039) method returns this list.

See Also

GetDefaultWebComponents (see page 6039)

GetWebComponents (see page 6039)

61.1.10 TWebContainerComponentHelper Class

TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainerComponentHelper = class(TObject);
```

C++

```
class TWebContainerComponentHelper : public TObject;
```

File

WebComp

Description

Components that maintain a list of Web items to help them generate HTML use TWebContainerComponentHelper to provide the common implementation of properties and methods for managing the list of sub components. This includes

The logic for handling two lists of Web items: one that represents the Web items that the application assigns, and one that represents a set of default items that can be used when the application does not explicitly assign Web items.

The logic for interacting with the component streaming system, so that child Web items are correctly loaded and saved with the component that uses TWebContainerComponentHelper.

See Also

TWebContainedContainerComponent (see page 6026)

TWebContainerComponent (see page 6036)

TWebComponentList (see page 5999)

61.1.10.1 TWebContainerComponentHelper Members

The following tables list the members exposed by TWebContainerComponentHelper.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 6048)	Occurs when TWebContainerComponentHelper (see page 6041) needs to access the value of DefaultWebComponents (see page 6047).
	OnSetDefaultComponentList (see page 6049)	Occurs when TWebContainerComponentHelper (see page 6041) needs to assign the value of DefaultWebComponents (see page 6047).

Protected Methods

	Name	Description
	GetComponentList (see page 6045)	Returns the value of the WebComponents property.
	GetDefaultComponentList (see page 6045)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.
	GetDefaultWebComponents (see page 6046)	Generates an OnGetDefaultComponentList (see page 6048) event.
	SetDefaultWebComponents (see page 6046)	Generates an OnSetDefaultComponentList (see page 6049) event.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6043)	Creates an instance of TWebContainerComponentHelper (see page 6041).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 6044)	Destroys an instance of TWebContainerComponentHelper (see page 6041).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetChildren (see page 6044)	Executes a callback for each child Web item.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetChildOrder (see page 6046)	Positions a child component in the list of child Web items.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultWebComponents (see page 6047)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
	Webcomponents (see page 6048)	This is Webcomponents, a member of class TWebContainerComponentHelper.

61.1.10.2 TWebContainerComponentHelper.Create Constructor

Creates an instance of TWebContainerComponentHelper (see page 6041).

Pascal

```
constructor Create(AComponent: TComponent; AOnGetDefaultComponentList: TGetComponentListEvent;
AOnSetDefaultComponentList: TSetComponentListEvent);
```

C++

```
__fastcall TWebContainerComponentHelper(TComponent * AComponent, TGetComponentListEvent
AOnGetDefaultComponentList, TSetComponentListEvent AOnSetDefaultComponentList);
```

Description

Applications do not create instances of TWebContainerComponentHelper (see page 6041). The component that uses TWebContainerComponentHelper (see page 6041) so that it can manage a set of Web items instantiates TWebContainerComponentHelper (see page 6041).

AComponent is the component that uses TWebContainerComponentHelper (see page 6041) to manage its list of Web items and to interact with the component streaming system.

AOnGetDefaultComponentList is an event handler that AComponent supplies to provide the read implementation of the DefaultWebComponents (see page 6047) property. It is assigned as the event handler for OnGetDefaultComponentList (see page 6048).

AOnSetDefaultComponentList is an event handler that AComponent supplies to provide the write implementation of the DefaultWebComponents (see page 6047) property. It is assigned as the event handler for OnSetDefaultComponentList (see page 6049).

See Also

OnGetDefaultComponentList (see page 6048)

OnSetDefaultComponentList (see page 6049)

DefaultWebComponents (see page 6047)

61.1.10.3 TWebContainerComponentHelper.Destroy Destructor

Destroys an instance of TWebContainerComponentHelper (see page 6041).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainerComponentHelper (see page 6041) object is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3382)

61.1.10.4 TWebContainerComponentHelper Methods

The methods of the TWebContainerComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetComponentList (see page 6045)	Returns the value of the WebComponents property.
	GetDefaultComponentList (see page 6045)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.
	GetDefaultWebComponents (see page 6046)	Generates an OnGetDefaultComponentList (see page 6048) event.
	SetDefaultWebComponents (see page 6046)	Generates an OnSetDefaultComponentList (see page 6049) event.

Public Methods

	Name	Description
	GetChildren (see page 6044)	Executes a callback for each child Web item.
	SetChildOrder (see page 6046)	Positions a child component in the list of child Web items.

61.1.10.4.1 TWebContainerComponentHelper.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent);
```

C++

```
__fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren provides the underlying implementation of the associated component's GetChildren method. GetChildren is

introduced in TComponent for the streaming system that loads and saves components.

GetChildren executes the Proc procedure for each child that in the list specified by the WebComponents property, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the component that uses TWebContainerComponentHelper (see page 6041).

See Also

WebComponents (see page 6048)

SetChildOrder (see page 6046)

GetChildren

61.1.10.4.2 TWebContainerComponentHelper.GetComponentList Method

Returns the value of the WebComponents property.

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

GetComponentList returns the list of Web items that have been assigned to the associated component. The return value is an instance of TWebComponentList (see page 5999).

See Also

WebComponents (see page 6048)

GetDefaultComponentList (see page 6045)

TWebComponentList (see page 5999)

61.1.10.4.3 TWebContainerComponentHelper.GetDefaultComponentList Method

Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

GetComponentList (see page 6045) returns the value of the DefaultWebComponents (see page 6047) property. The return value is an instance of TWebComponentList (see page 5999).

See Also

DefaultWebComponents (see page 6047)

WebComponents (see page 6048)

TWebComponentList (see page 5999)

61.1.10.4.4 TWebContainerComponentHelper.GetDefaultWebComponents Method

Generates an OnGetDefaultComponentList (see page 6048) event.

Pascal

```
function GetDefaultWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents is the read implementation of the DefaultWebComponents (see page 6047) property. It generates an OnGetDefaultComponentList (see page 6048) event, so that the associated component can supply this list of default items. If there is no OnGetDefaultComponentList (see page 6048) event handler, GetDefaultWebComponents returns nil (Delphi) or NULL (C++).

See Also

OnGetDefaultComponentList (see page 6048)

DefaultWebComponents (see page 6047)

61.1.10.4.5 TWebContainerComponentHelper.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer);
```

C++

```
__fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder provides the underlying implementation of the associated component's SetChildOrder method. SetChildOrder is introduced in TComponent for the streaming system that loads and saves components. It positions a newly loaded child component in the list specified by WebComponents.

Child is the newly loaded child component.

Order indicates its position in WebComponents, where 0 is the first position, 1 is the second position, and so on.

See Also

WebComponents (see page 6048)

SetChildOrder

GetChildren (see page 6044)

61.1.10.4.6 TWebContainerComponentHelper.SetDefaultWebComponents Method

Generates an OnSetDefaultComponentList (see page 6049) event.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList);
```

C++

```
__fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 6046) is the write implementation of the DefaultWebComponents (see page 6047) property. It generates an OnSetDefaultComponentList (see page 6049) event, so that the associated component can assign the new list of default items.

See Also

Create (see page 6043)

OnSetDefaultComponentList (see page 6049)

DefaultWebComponents (see page 6047)

61.1.10.5 TWebContainerComponentHelper Properties

The properties of the TWebContainerComponentHelper class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 6047)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
R	Webcomponents (see page 6048)	This is Webcomponents, a member of class TWebContainerComponentHelper.

61.1.10.5.1 TWebContainerComponentHelper.DefaultWebComponents Property

Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Use DefaultWebComponents to access the list of default Web items that can be used when no items have been explicitly assigned.

TWebContainerComponentHelper (see page 6041) does not actually store this list. It leaves that task to the component that uses TWebContainerComponentHelper (see page 6041). Instead, TWebContainerComponentHelper (see page 6041) uses two events, OnGetDefaultComponentList (see page 6048) and OnSetDefaultComponentList (see page 6049) for reading and writing this list. The associated component that uses TWebContainerComponentHelper (see page 6041) assigns event handlers to these two events, thereby providing the implementation for this property.

See Also

WebComponents (see page 6048)

OnGetDefaultComponentList (see page 6048)

OnGetDefaultComponentList (see page 6048)

61.1.10.5.2 TWebContainerComponentHelper.Webcomponents Property

Pascal

```
property Webcomponents: TWebComponentList;
```

C++

```
__property TWebComponentList Webcomponents;
```

Description

This is Webcomponents, a member of class TWebContainerComponentHelper.

61.1.10.6 TWebContainerComponentHelper Events

The events of the TWebContainerComponentHelper class are listed here.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 6048)	Occurs when TWebContainerComponentHelper (see page 6041) needs to access the value of DefaultWebComponents (see page 6047).
	OnSetDefaultComponentList (see page 6049)	Occurs when TWebContainerComponentHelper (see page 6041) needs to assign the value of DefaultWebComponents (see page 6047).

61.1.10.6.1 TWebContainerComponentHelper.OnGetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 6041) needs to access the value of DefaultWebComponents (see page 6047).

Pascal

```
property OnGetDefaultComponentList: TGetComponentListEvent;
```

C++

```
__property TGetComponentListEvent OnGetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 6041) assigns an event handler for OnGetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 6041) instance. This event handler provides the underlying read implementation of the DefaultWebComponents (see page 6047) property.

If the component that uses TWebContainerComponentHelper (see page 6041) does not provide an OnGetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 6047) property is nil (Delphi) or NULL (C++).

See Also

OnSetDefaultComponentList (see page 6049)

DefaultWebComponents (see page 6047)

Create (see page 6043)

61.1.10.6.2 TWebContainerComponentHelper.OnSetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 6041) needs to assign the value of DefaultWebComponents (see page 6047).

Pascal

```
property OnSetDefaultComponentList: TSetComponentListEvent;
```

C++

```
__property TSetComponentListEvent OnSetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 6041) assigns an event handler for OnSetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 6041) instance. This event handler provides the underlying write implementation of the DefaultWebComponents (see page 6047) property.

If the component that uses TWebContainerComponentHelper (see page 6041) does not provide an OnSetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 6047) can't be changed.

See Also

OnGetDefaultComponentList (see page 6048)

DefaultWebComponents (see page 6047)

Create (see page 6043)

61.1.11 TWebContentOptions Class

TWebContentOptions controls what a Web item can include in its generated HTML.

Class Hierarchy

Pascal

```
TWebContentOptions = class(TObject);
```

C++

```
class TWebContentOptions : public TObject;
```

File

WebComp

Description

Use TWebContentOptions to influence the HTML generated by a component that supports the IWebContent (see page 6075) interface. Each TWebContentOptions represents a set of flags that are passed to the constructor.

See Also

IWebContent (see page 6075)

61.1.11.1 TWebContentOptions Members

The following tables list the members exposed by TWebContentOptions.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6050)	Creates a new TWebContentOptions (see page 6049) object.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Flags (see page 6051)	Lists the flags that indicate what type of generated HTML is permitted.

61.1.11.2 TWebContentOptions.Create Constructor

Creates a new TWebContentOptions (see page 6049) object.

Pascal

```
constructor Create(AFlags: TWebContentFlags);
```

C++

```
__fastcall TWebContentOptions(TWebContentFlags AFlags);
```

Description

Call Create to create a TWebContentOptions (see page 6049) instance at runtime. Pass the flags that the object should represent as the value of the AFlags parameter. This value initializes the Flags (see page 6051) property.

See Also

Flags (see page 6051)

61.1.11.3 TWebContentOptions Properties

The properties of the TWebContentOptions class are listed here.

Public Properties

	Name	Description
R	Flags (see page 6051)	Lists the flags that indicate what type of generated HTML is permitted.

61.1.11.3.1 TWebContentOptions.Flags Property

Lists the flags that indicate what type of generated HTML is permitted.

Pascal

```
property Flags: TWebContentFlags;
```

C++

```
__property TWebContentFlags Flags;
```

Description

Read (see page 3727) Flags to determine what types of HTML a Web item can generate.

See Also

Create (see page 6050)

61.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 6052)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 6054)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 6055)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 6056)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 6057)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 6059)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 6060)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 6062)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 6064)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 6065)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 6065)	IWebComponent is the interface supported by components that TWebComponentList (see page 5999) manages.
	IWebComponentCommands (see page 6069)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 6071)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 6073)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 6074)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 6075)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 6077)	This is class WebComp.IWebVariablesContainerName.

61.2.1 IAddFieldsEditor Interface

Class Hierarchy

WebComp.IAddFieldsEditor

Pascal

```
IAddFieldsEditor = interface;
```

C++

```
__interface IAddFieldsEditor;
```

File

WebComp

Description

This is class WebComp.IAddFieldsEditor.

61.2.1.1 IAddFieldsEditor Members

The following tables list the members exposed by IAddFieldsEditor.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

61.2.1.2 IAddFieldsEditor Methods

The methods of the IAddFieldsEditor class are listed here.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

61.2.1.2.1 IAddFieldsEditor.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.

61.2.1.2.2 IAddFieldsEditor.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.

61.2.1.2.3 IAddFieldsEditor.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class IAddFieldsEditor.

61.2.1.2.4 IAddFieldsEditor.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

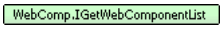
```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

61.2.2 IGetWebComponentList Interface

Class Hierarchy



Pascal

```
IGetWebComponentList = interface;
```

C++

```
__interface IGetWebComponentList;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentList.

61.2.2.1 IGetWebComponentList Members

The following tables list the members exposed by IGetWebComponentList.

Public Methods

	Name	Description
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

61.2.2.2 IGetWebComponentList Methods

The methods of the IGetWebComponentList class are listed here.

Public Methods

	Name	Description
	GetComponentList (see page 6055)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6055)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

61.2.2.2.1 IGetWebComponentList.GetComponentList Method

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

This is GetComponentList, a member of class IGetWebComponentList.

61.2.2.2.2 IGetWebComponentList.GetDefaultComponentList Method

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

This is GetDefaultComponentList, a member of class IGetWebComponentList.

61.2.3 IGetWebComponentNodeName Interface

Class Hierarchy

Pascal

```
IGetWebComponentNodeName = interface;
```

C++

```
__interface IGetWebComponentNodeName;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentNodeName.

61.2.3.1 IGetWebComponentNodeName Members

The following tables list the members exposed by IGetWebComponentNodeName.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

61.2.3.2 IGetWebComponentNodeName Methods

The methods of the IGetWebComponentNodeName class are listed here.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 6056)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

61.2.3.2.1 IGetWebComponentNodeName.GetWebComponentNodeName Method

Pascal

```
function GetWebComponentNodeName: string;
```

C++

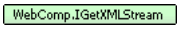
```
__fastcall UnicodeString GetWebComponentNodeName();
```

Description

This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

61.2.4 IGetXMLStream Interface

Class Hierarchy



Pascal

```
IGetXMLStream = interface;
```

C++

```
__interface IGetXMLStream;
```

File

WebComp

Description

This is class WebComp.IGetXMLStream.

61.2.4.1 IGetXMLStream Members

The following tables list the members exposed by IGetXMLStream.

Public Methods

	Name	Description
	GetXMLStream (see page 6057)	This is GetXMLStream, a member of class IGetXMLStream.

61.2.4.2 IGetXMLStream Methods

The methods of the IGetXMLStream class are listed here.

Public Methods

	Name	Description
	GetXMLStream (see page 6057)	This is GetXMLStream, a member of class IGetXMLStream.

61.2.4.2.1 IGetXMLStream.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

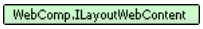
Description

This is GetXMLStream, a member of class IGetXMLStream.

61.2.5 ILayoutWebContent Interface

ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
ILayoutWebContent = interface;
```

C++

```
__interface ILayoutWebContent;
```

File

WebComp

Description

The Content method, implemented by all Web items to return the generated HTML, takes as a parameter an object of type TLayout (see page 6085). TLayout (see page 6085) is any object that implements the ILayoutWebContent interface. Typically, this is a descendant of TLayoutWebContent (see page 5988).

Parent components pass the TLayout (see page 6085) object to child Web items to provide helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the ILayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

See Also

TLayout (see page 6085)

IWebContent (see page 6075)

TLayoutWebContent (see page 5988)

61.2.5.1 ILayoutWebContent Members

The following tables list the members exposed by ILayoutWebContent.

Public Methods

	Name	Description
	LayoutButton (see page 6058)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 6058)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 6058)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6059)	This is LayoutTable, a member of class ILayoutWebContent.

61.2.5.2 ILayoutWebContent Methods

The methods of the ILayoutWebContent class are listed here.

Public Methods

	Name	Description
	LayoutButton (see page 6058)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 6058)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 6058)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6059)	This is LayoutTable, a member of class ILayoutWebContent.

61.2.5.2.1 ILayoutWebContent.LayoutButton Method

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutButton(const UnicodeString HTMLButton, TLayoutAttributes
Attributes);
```

Description

This is LayoutButton, a member of class ILayoutWebContent.

61.2.5.2.2 ILayoutWebContent.LayoutField Method

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutField(const UnicodeString HTMLField, TLayoutAttributes
Attributes);
```

Description

This is LayoutField, a member of class ILayoutWebContent.

61.2.5.2.3 ILayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the Web item's parent can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form required by the Web item's parent.

HTMLLabel is the HTML that declares the label. This is produced by the Web item.

HTMLField is the HTML that declares the field. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that can be used by the Web item's parent.

61.2.5.2.4 ILayoutWebContent.LayoutTable Method

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

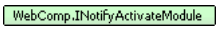
```
__fastcall UnicodeString LayoutTable(const UnicodeString HTMLTable, TLayoutAttributes
Attributes);
```

Description

This is LayoutTable, a member of class ILayoutWebContent.

61.2.6 INotifyActivateModule Interface

Class Hierarchy



Pascal

```
INotifyActivateModule = interface;
```

C++

```
__interface INotifyActivateModule;
```

File

WebComp

Description

This is class WebComp.INotifyActivateModule.

61.2.6.1 INotifyActivateModule Members

The following tables list the members exposed by INotifyActivateModule.

Public Methods

	Name	Description
	NotifyActivateModule (see page 6060)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 6060)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

61.2.6.2 INotifyActivateModule Methods

The methods of the INotifyActivateModule class are listed here.

Public Methods

	Name	Description
	NotifyActivateModule (see page 6060)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 6060)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

61.2.6.2.1 INotifyActivateModule.NotifyActivateModule Method

Pascal

```
procedure NotifyActivateModule(AWebModule: TComponent);
```

C++

```
__fastcall NotifyActivateModule(TComponent * AWebModule);
```

Description

This is NotifyActivateModule, a member of class INotifyActivateModule.

61.2.6.2.2 INotifyActivateModule.NotifyDeactivateModule Method

Pascal

```
procedure NotifyDeactivateModule(AWebModule: TComponent);
```

C++

```
__fastcall NotifyDeactivateModule(TComponent * AWebModule);
```

Description

This is NotifyDeactivateModule, a member of class INotifyActivateModule.

61.2.7 INotifyWebActivate Interface

Class Hierarchy

WebComp.INotifyWebActivate

Pascal

```
INotifyWebActivate = interface;
```

C++

```
__interface INotifyWebActivate;
```

File

WebComp

Description

This is class WebComp.INotifyWebActivate.

61.2.7.1 INotifyWebActivate Members

The following tables list the members exposed by INotifyWebActivate.

Public Methods

	Name	Description
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.

61.2.7.2 INotifyWebActivate Methods

The methods of the INotifyWebActivate class are listed here.

Public Methods

	Name	Description
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.

61.2.7.2.1 INotifyWebActivate.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class INotifyWebActivate.

61.2.7.2.2 INotifyWebActivate.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

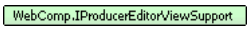
```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class INotifyWebActivate.

61.2.8 IProducerEditorViewSupport Interface

Class Hierarchy



Pascal

```
IProducerEditorViewSupport = interface;
```

C++

```
__interface IProducerEditorViewSupport;
```

File

WebComp

Description

This is class WebComp.IProducerEditorViewSupport.

61.2.8.1 IProducerEditorViewSupport Members

The following tables list the members exposed by IProducerEditorViewSupport.

Public Methods

	Name	Description
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

61.2.8.2 IProducerEditorViewSupport Methods

The methods of the IProducerEditorViewSupport class are listed here.

Public Methods

	Name	Description
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

61.2.8.2.1 IProducerEditorViewSupport.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class IProducerEditorViewSupport.

61.2.8.2.2 IProducerEditorViewSupport.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class IProducerEditorViewSupport.

61.2.8.2.3 IProducerEditorViewSupport.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class IProducerEditorViewSupport.

61.2.8.2.4 IProducerEditorViewSupport.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.

61.2.8.2.5 IProducerEditorViewSupport.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class IProducerEditorViewSupport.

61.2.8.2.6 IProducerEditorViewSupport.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.

61.2.8.2.7 IProducerEditorViewSupport.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

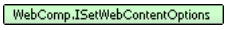
```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

61.2.9 ISetWebContentOptions Interface

Class Hierarchy



Pascal

```
ISetWebContentOptions = interface;
```

C++

```
__interface ISetWebContentOptions;
```

File

WebComp

Description

This is class WebComp.ISetWebContentOptions.

61.2.9.1 ISetWebContentOptions Members

The following tables list the members exposed by ISetWebContentOptions.

Public Methods

	Name	Description
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

61.2.9.2 ISetWebContentOptions Methods

The methods of the ISetWebContentOptions class are listed here.

Public Methods

	Name	Description
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

61.2.9.2.1 ISetWebContentOptions.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class ISetWebContentOptions.

61.2.10 ITopLevelWebComponent Interface

Class Hierarchy

WebComp.ITopLevelWebComponent

Pascal

```
ITopLevelWebComponent = interface;
```

C++

```
__interface ITopLevelWebComponent;
```

File

WebComp

Description

This is class WebComp.ITopLevelWebComponent.

61.2.11 IWebComponent Interface

IWebComponent is the interface supported by components that TWebComponentList (see page 5999) manages.

Class Hierarchy

WebComp.IWebComponent

Pascal

```
IWebComponent = interface;
```

C++

```
__interface IWebComponent;
```

File

WebComp

Description

TWebComponentList (see page 5999) uses IWebComponent to ensure that all components in its list have access to information about how they fit in the hierarchy of Web items.

Note: VCL methods that work with the IWebComponent interface wrap it using the DelphiInterface (see page 3263) class:

Note: typedef System::DelphiInterface (see page 3263)< IWebComponent > _di_IWebComponent;

See Also

TWebComponentList (see page 5999)

61.2.11.1 IWebComponent Members

The following tables list the members exposed by IWebComponent.

Public Methods

	Name	Description
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

61.2.11.2 IWebComponent Methods

The methods of the IWebComponent class are listed here.

Public Methods

	Name	Description
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

61.2.11.2.1 IWebComponent.GetContainer Method

Returns the value of the Container (see page 6069) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

GetContainer is the read method for the Container (see page 6069) property.

See Also

Container (see page 6069)

61.2.11.2.2 IWebComponent.GetIndex Method

Returns the value of the Index (see page 6069) property.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex is the read method for the Index (see page 6069) property.

See Also

Index (see page 6069)

61.2.11.2.3 IWebComponent.SetComponentList Method

Sets the value of the ComponentList (see page 6068) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList is the write method for the ComponentList (see page 6068) property. The List parameter specifies the value to assign to ComponentList (see page 6068).

See Also

ComponentList (see page 6068)

61.2.11.2.4 IWebComponent.SetContainer Method

Sets the value of the Container (see page 6069) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer is the write method for the Container (see page 6069) property. The Container (see page 6069) parameter specifies the value to assign to Container (see page 6069).

See Also

Container (see page 6069)

61.2.11.2.5 IWebComponent.SetIndex Method

Sets the value of the Index (see page 6069) property.

Pascal

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

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex is the write method for the Index (see page 6069) property. The Value parameter specifies the value to assign to Index (see page 6069).

See Also

Index (see page 6069)

61.2.11.3 IWebComponent Properties

The properties of the IWebComponent class are listed here.

Public Properties

	Name	Description
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

61.2.11.3.1 IWebComponent.ComponentList Property

Indicates the component list that contains the Web item.

Pascal

```
property ComponentList: TObject;
```

C++

```
__property TObject * ComponentList;
```

Description

The Web component list assigns a value to ComponentList when the component is added to it. The component can use this

value to access the list and obtain information about the other components in the list.

Note that the IWebComponent (see page 6065) interface does not include any methods for reading the value of ComponentList.

See Also

Container (see page 6069)

Index (see page 6069)

TWebComponentList (see page 5999)

61.2.11.3.2 IWebComponent.Container Property

Specifies the component that uses content generated by this Web item.

Pascal

```
property Container: TWebComponentContainer;
```

C++

```
__property TWebComponentContainer Container;
```

Description

Web items are arranged in a hierarchy, where each item generates its content using content produced by its children. Container specifies the parent that uses the content of this Web item. Container maintains a list of all its children using a TWebComponentList (see page 5999) object.

See Also

ComponentList (see page 6068)

ParentComponent (see page 6008)

61.2.11.3.3 IWebComponent.Index Property

Indicates where the Web item appears in the Web component list's array of Web items.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

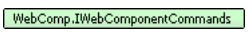
The Web component list sets Index to indicate where the component sits in its WebComponents property array.

See Also

WebComponents (see page 6008)

61.2.12 IWebComponentCommands Interface

Class Hierarchy



Pascal

```
IWebComponentCommands = interface;
```

C++

```
__interface IWebComponentCommands;
```

File

WebComp

Description

This is class WebComp.IWebComponentCommands.

61.2.12.1 IWebComponentCommands Members

The following tables list the members exposed by IWebComponentCommands.

Public Methods

	Name	Description
	ExecuteVerb (see page 6070)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 6070)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 6071)	This is GetVerbCount, a member of class IWebComponentCommands.

61.2.12.2 IWebComponentCommands Methods

The methods of the IWebComponentCommands class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 6070)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 6070)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 6071)	This is GetVerbCount, a member of class IWebComponentCommands.

61.2.12.2.1 IWebComponentCommands.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(AComponent: TComponent; Index: Integer);
```

C++

```
__fastcall ExecuteVerb(TComponent * AComponent, int Index);
```

Description

This is ExecuteVerb, a member of class IWebComponentCommands.

61.2.12.2.2 IWebComponentCommands.GetVerb Method

Pascal

```
function GetVerb(AComponent: TComponent; Index: Integer): string;
```

C++

```
__fastcall UnicodeString GetVerb(TComponent * AComponent, int Index);
```

Description

This is GetVerb, a member of class IWebComponentCommands.

61.2.12.2.3 IWebComponentCommands.GetVerbCount Method

Pascal

```
function GetVerbCount(AComponent: TComponent): Integer;
```

C++

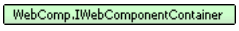
```
__fastcall int GetVerbCount(TComponent * AComponent);
```

Description

This is GetVerbCount, a member of class IWebComponentCommands.

61.2.13 IWebComponentContainer Interface

Class Hierarchy



Pascal

```
IWebComponentContainer = interface;
```

C++

```
__interface IWebComponentContainer;
```

File

WebComp

Description

This is class WebComp.IWebComponentContainer.

61.2.13.1 IWebComponentContainer Members

The following tables list the members exposed by IWebComponentContainer.

Public Methods

	Name	Description
	GetComponent (see page 6072)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 6072)	This is GetComponentCount, a member of class IWebComponentContainer.

Public Properties

	Name	Description
R	ComponentCount (see page 6072)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 6073)	This is Components, a member of class IWebComponentContainer.

61.2.13.2 IWebComponentContainer Methods

The methods of the IWebComponentContainer class are listed here.

Public Methods

	Name	Description
	GetComponent (see page 6072)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 6072)	This is GetComponentCount, a member of class IWebComponentContainer.

61.2.13.2.1 IWebComponentContainer.GetComponent Method

Pascal

```
function GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int I);
```

Description

This is GetComponent, a member of class IWebComponentContainer.

61.2.13.2.2 IWebComponentContainer.GetComponentCount Method

Pascal

```
function GetComponentCount: Integer;
```

C++

```
__fastcall int GetComponentCount();
```

Description

This is GetComponentCount, a member of class IWebComponentContainer.

61.2.13.3 IWebComponentContainer Properties

The properties of the IWebComponentContainer class are listed here.

Public Properties

	Name	Description
R	ComponentCount (see page 6072)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 6073)	This is Components, a member of class IWebComponentContainer.

61.2.13.3.1 IWebComponentContainer.ComponentCount Property

Pascal

```
property ComponentCount: Integer;
```

C++

```
__property int ComponentCount;
```

Description

This is ComponentCount, a member of class IWebComponentContainer.

61.2.13.3.2 IWebComponentContainer.Components Property

Pascal

```
property Components [I: Integer]: TComponent;
```

C++

```
__property TComponent * Components[int I];
```

Description

This is Components, a member of class IWebComponentContainer.

61.2.14 IWebComponentEditor Interface

IWebComponentEditor enables a Web item to be assigned children in the IDE.

Class Hierarchy

Pascal

```
IWebComponentEditor = interface;
```

C++

```
__interface IWebComponentEditor;
```

File

WebComp

Description

When the Web Page editor builds the list of new items that can be added as the child of a Web item, it calls the IWebComponentEditor interface. Components that contain child items for generating portions of their HTML content must implement this interface so that the Web Page editor can display the appropriate list of components to add.

IWebComponentEditor defines a single method, CanAddClass (see page 6074), which indicates whether a specified class can be used for a child Web item by the Web item that implements the interface.

See Also

IRestoreDefaults

TAdapterPageProducer (see page 1076)

TInetXPageProducer

TWebForm

;TWebControlGroup

TAdapterSubComponent (see page 5394)

61.2.14.1 IWebComponentEditor Members

The following tables list the members exposed by IWebComponentEditor.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.

61.2.14.2 IWebComponentEditor Methods

The methods of the IWebComponentEditor class are listed here.

Public Methods

	Name	Description
	CanAddClass (see page 6074)	Indicates whether a specified class can be used as a child Web item.

61.2.14.2.1 IWebComponentEditor.CanAddClass Method

Indicates whether a specified class can be used as a child Web item.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the Web item that implements this interface.

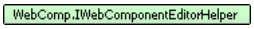
AParent is the Web item that will act as a parent to the added Web item.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web item can use content generated by a component of type AClass, false otherwise.

61.2.15 IWebComponentEditorHelper Interface

Class Hierarchy



Pascal

```
IWebComponentEditorHelper = interface;
```

C++

```
__interface IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.IWebComponentEditorHelper.

61.2.15.1 IWebComponentEditorHelper Members

The following tables list the members exposed by IWebComponentEditorHelper.

Public Methods

	Name	Description
	CanAddClassHelper (see page 6075)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

61.2.15.2 IWebComponentEditorHelper Methods

The methods of the IWebComponentEditorHelper class are listed here.

Public Methods

	Name	Description
	CanAddClassHelper (see page 6075)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

61.2.15.2.1 IWebComponentEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

Description

This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

61.2.16 IWebContent Interface

IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.

Class Hierarchy

Pascal

```
IWebContent = interface;
```

C++

```
__interface IWebContent;
```

File

WebComp

Description

TInetXPageProducer, TAdapterPageProducer, and TComponentsPageProducer create HTML documents that include content generated by other components, called Web items. Similarly, the Web items that generate content can make use of other Web items when producing their content.

For a component to act as a Web item, it must support the IWebContent interface. This interface defines one new method, Content (see page 6076), which returns the HTML generated by the Web item.

Note: The only other required interface for Web items is IWebComponent (see page 6065).

See Also

TAdapterPageProducer (see page 1076)

TInetXPageProducer

TBasePageItemsProducer

TWebComponentList (see page 5999)

IWebComponent (see page 6065)

Customizing the InternetExpress Page Producer Template

61.2.16.1 IWebContent Members

The following tables list the members exposed by IWebContent.

Public Methods

	Name	Description
	Content (see page 6076)	Returns the HTML that the Web item generates.

61.2.16.2 IWebContent Methods

The methods of the IWebContent class are listed here.

Public Methods

	Name	Description
	Content (see page 6076)	Returns the HTML that the Web item generates.

61.2.16.2.1 IWebContent.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent (see page 6057) interface, which the Web item uses to fit its own content into the parent's HTML structures.

Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 6057)

61.2.17 IWebVariablesContainerName Interface

Class Hierarchy

Pascal

```
IWebVariablesContainerName = interface;
```

C++

```
__interface IWebVariablesContainerName;
```

File

WebComp

Description

This is class WebComp.IWebVariablesContainerName.

61.2.17.1 IWebVariablesContainerName Members

The following tables list the members exposed by IWebVariablesContainerName.

Public Methods

	Name	Description
	GetContainerName (see page 6077)	This is GetContainerName, a member of class IWebVariablesContainerName.

Public Properties

	Name	Description
R	ContainerName (see page 6078)	This is ContainerName, a member of class IWebVariablesContainerName.

61.2.17.2 IWebVariablesContainerName Methods

The methods of the IWebVariablesContainerName class are listed here.

Public Methods

	Name	Description
	GetContainerName (see page 6077)	This is GetContainerName, a member of class IWebVariablesContainerName.

61.2.17.2.1 IWebVariablesContainerName.GetContainerName Method

Pascal

```
function GetContainerName: string;
```

C++

```
__fastcall UnicodeString GetContainerName();
```

Description

This is GetContainerName, a member of class IWebVariablesContainerName.

61.2.17.3 IWebVariablesContainerName Properties

The properties of the IWebVariablesContainerName class are listed here.

Public Properties

	Name	Description
	ContainerName (see page 6078)	This is ContainerName, a member of class IWebVariablesContainerName.

61.2.17.3.1 IWebVariablesContainerName.ContainerName Property

Pascal

```
property ContainerName: string;
```

C++

```
__property UnicodeString ContainerName;
```

Description

This is ContainerName, a member of class IWebVariablesContainerName.

61.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DesigningComponent (see page 6079)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 6079)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 6080)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 6080)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 6080)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 6081)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 6081)	This is function WebComp.ValidateWebParent.

61.3.1 WebComp.DesigningComponent Function

Pascal

```
function DesigningComponent(AComponent: TComponent): Boolean;
```

C++

```
Boolean DesigningComponent(TComponent * AComponent);
```

File

WebComp

Description

This is function WebComp.DesigningComponent.

61.3.2 WebComp.EnumRegisteredWebComponents Function

Pascal

```
procedure EnumRegisteredWebComponents(Proc: TEnumWebComponentsProc; Info: Pointer);
```

C++

```
EnumRegisteredWebComponents(TEnumWebComponentsProc Proc, void * Info);
```

File

WebComp

Description

This is function WebComp.EnumRegisteredWebComponents.

61.3.3 WebComp.FindWebComponentEditorHelper Function

Pascal

```
function FindWebComponentEditorHelper(AClass: TClass): TObject;
```

C++

```
TObject * FindWebComponentEditorHelper(TClass AClass);
```

File

WebComp

Description

This is function WebComp.FindWebComponentEditorHelper.

61.3.4 WebComp.QualifyFileName Function

Pascal

```
function QualifyFileName(const AFileName: string): string;
```

C++

```
UnicodeString QualifyFileName(const UnicodeString AFileName);
```

File

WebComp

Description

This is function WebComp.QualifyFileName.

61.3.5 WebComp.RegisterWebComponents Function

Pascal

```
procedure RegisterWebComponents(const AClasses: array of TComponentClass; Editor: TObject =
nil);
```

C++

```
RegisterWebComponents(const array of TComponentClass AClasses, TObject * Editor = nil);
```

File

WebComp

Description

This is function WebComp.RegisterWebComponents.

61.3.6 WebComp.UnRegisterWebComponents Function

Pascal

```
procedure UnRegisterWebComponents(const AClasses: array of TComponentClass);
```

C++

```
UnRegisterWebComponents(const array of TComponentClass AClasses);
```

File

WebComp

Description

This is function WebComp.UnRegisterWebComponents.

61.3.7 WebComp.ValidateWebParent Function

Pascal

```
procedure ValidateWebParent(AComponent: TComponent; AParent: TComponent);
```

C++

```
ValidateWebParent(TComponent * AComponent, TComponent * AParent);
```

File

WebComp

Description

This is function WebComp.ValidateWebParent.

61.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TLabelPosition (see page 6082)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 6082)	This is record WebComp.TWebContentFlag.

61.4.1 WebComp.TLabelPosition Enumeration

Pascal

```
TLabelPosition = (
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
);
```

C++

```
enum TLabelPosition {
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
};
```

File

WebComp

Description

This is record WebComp.TLabelPosition.

61.4.2 WebComp.TWebContentFlag Enumeration

Pascal

```
TWebContentFlag = (
  coNoScript,
  coLocalPaths,
  coNoExecuteScript
);
```

C++

```
enum TWebContentFlag {
  coNoScript,
  coLocalPaths,
  coNoExecuteScript
};
```

File

WebComp

Description

This is record WebComp.TWebContentFlag.

61.5 Types

The following table lists types in this documentation.

Types

Name	Description
TEnumWebComponentsProc (see page 6084)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 6084)	This is type WebComp.TGetComponentListEvent.
TLayout (see page 6085)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 6085)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 6085)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 6086)	This is type WebComp.TWebContentFlags.

61.5.1 WebComp.TEnumWebComponentsProc Type

Pascal

```
TEnumWebComponentsProc = procedure (WebComponentClass: TWebComponentClassDescription; Info:
Pointer) of object;
```

C++

```
(WebComponentClass: TWebComponentClassDescription; Info: Pointer) ( TEnumWebComponentsProc)();
```

File

WebComp

Description

This is type WebComp.TEnumWebComponentsProc.

61.5.2 WebComp.TGetComponentListEvent Type

Pascal

```
TGetComponentListEvent = procedure (Sender: TObject; var AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; var AList: TWebComponentList) ( TGetComponentListEvent)();
```

File

WebComp

Description

This is type WebComp.TGetComponentListEvent.

61.5.3 WebComp.TLayout Type

TLayout is the type for helper objects that generate layout code for a Web item.

Pascal

```
TLayout = TObject;
```

C++

```
TObject * TLayout;
```

File

WebComp

Description

TLayout is the type for objects that Web items use to transmit common functions that control the layout of HTML controls. When a Web item generates content that includes HTML produced by child components, it passes a TLayout object to those children. The TLayout object implements an interface (ILayoutWebContent (see page 6057)) that generates HTML to lay out the controls which the child components generate.

Note: Typically, TLayout objects are descendants of TLayoutWebContent (see page 5988).

See Also

IWebContent (see page 6075)

ILayoutWebContent (see page 6057)

TLayoutWebContent (see page 5988)

61.5.4 WebComp.TSetComponentListEvent Type

Pascal

```
TSetComponentListEvent = procedure (Sender: TObject; AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; AList: TWebComponentList) ( TSetComponentListEvent)();
```

File

WebComp

Description

This is type WebComp.TSetComponentListEvent.

61.5.5 WebComp.TWebComponentContainer Type

TWebComponentContainer is the type for components that maintain a list of Web items.

Pascal

```
TWebComponentContainer = TObject;
```

C++

```
TObject * TWebComponentContainer;
```

File

WebComp

Description

TWebComponentContainer is the type for objects that make use of Web items (components that generate HTML and support the IWebContent (see page 6075) interface). Each TWebComponentContainer has a property of type TWebComponentList (see page 5999) that lists its Web items. The TWebComponentContainer uses these Web items to generate constituent parts of the HTML that it generates.

See Also

TWebComponentList (see page 5999)

IWebContent (see page 6075)

61.5.6 WebComp.TWebContentFlags Type

Pascal

```
TWebContentFlags = set of TWebContentFlag;
```

C++

```
TWebContentFlag TWebContentFlags;
```

File

WebComp

Description

This is type WebComp.TWebContentFlags.

61.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 6087)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 6087)	This is constant WebComp.FindWebComponentEditorHelperProc.
RegisterWebComponentsProc (see page 6088)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 6088)	This is constant WebComp.UnRegisterWebComponentsProc.

61.6.1 WebComp.EnumRegisteredWebComponentsProc Constant

Pascal

```
EnumRegisteredWebComponentsProc: procedure (Proc: TEnumWebComponentsProc; Info: Pointer) = nil;
```

C++

```
procedure (Proc: TEnumWebComponentsProc; Info: Pointer) EnumRegisteredWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.EnumRegisteredWebComponentsProc.

61.6.2 WebComp.FindWebComponentEditorHelperProc Constant

Pascal

```
FindWebComponentEditorHelperProc: function (AClass: TClass): TObject = nil;
```

C++

```
function (AClass: TClass): TObject FindWebComponentEditorHelperProc = nil;
```

File

WebComp

Description

This is constant WebComp.FindWebComponentEditorHelperProc.

61.6.3 WebComp.RegisterWebComponentsProc Constant

Pascal

```
RegisterWebComponentsProc: procedure ( const AClasses: array of TComponentClass; Editor:
TObject) = nil;
```

C++

```
procedure ( const AClasses: array of TComponentClass; Editor: TObject)
RegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.RegisterWebComponentsProc.

61.6.4 WebComp.UnRegisterWebComponentsProc Constant

Pascal

```
UnRegisterWebComponentsProc: procedure (const AClasses: array of TComponentClass) = nil;
```

C++

```
procedure (const AClasses: array of TComponentClass) UnRegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.UnRegisterWebComponentsProc.

62 WebContnrs Namespace

This is namespace WebContnrs.

Topics

Name	Description
Classes (see page 6090)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6113)	The following table lists structs, records, enums in this documentation.
Types (see page 6114)	The following table lists types in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 6090)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6106) object.
	TAbstractNamedVariants (see page 6090)	TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3430) values.
	TNamedVariantsList (see page 6106)	TNamedVariantsList is a persistent list of name/Variant (see page 3430) pairs.

Structs, Records, Enums

	Name	Description
	TNamedVariantItem (see page 6113)	This is record WebContnrs.TNamedVariantItem.

Types

Name	Description
PNamedVariantItem (see page 6114)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 6114)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 6114)	This is type WebContnrs.TNamedVariantList.

62.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 6090)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6106) object.
	TAbstractNamedVariants (see page 6090)	TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3430) values.
	TNamedVariantsList (see page 6106)	TNamedVariantsList is a persistent list of name/Variant (see page 3430) pairs.

62.1.1 ENamedVariantsError Class

ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6106) object.

Class Hierarchy

Pascal

```
ENamedVariantsError = class(Exception);
```

C++

```
class ENamedVariantsError : public Exception;
```

File

WebContnrs

Description

ENamedVariantsError is raised when an error occurs in TNamedVariantsList (see page 6106). For example, when an application supplies an invalid index to one of the TNamedVariantsList (see page 6106) methods, it raises an ENamedVariantsError exception.

See Also

TNamedVariantsList (see page 6106)

62.1.2 TAbstractNamedVariants Class

TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3430) values.

Class Hierarchy

Pascal

```
TAbstractNamedVariants = class(TPersistent);
```

C++

```
class TAbstractNamedVariants : public TPersistent;
```

File

WebContnrs

Description

Use TAbstractNamedVariants as a base class for objects that represents a set of name/value pairs where the values are all Variants (see page 6105). TAbstractNamedVariants is a descendant of TPersistent, which means that if it is the type of a published property on a component, the named variants list is saved with the component.

Do not create instances of TAbstractNamedVariants: This class contains abstract or, in C++ terminology, pure virtual methods. Instead, use a descendant of TAbstractNamedVariants such as TNamedVariantsList (see page 6106). TNamedVariantsList (see page 6106) provides an implementation for the abstract or pure virtual methods introduced in TAbstractNamedVariants.

See Also

TStrings

TCollection

TList

TNamedVariantsList (see page 6106)

62.1.2.1 TAbstractNamedVariants Members

The following tables list the members exposed by TAbstractNamedVariants.

Protected Methods

	Name	Description
	CompareStrings (see page 6095)	Compares two strings.
	DefineProperties (see page 6096)	Allows the list of names and their corresponding Variant (see page 3430) values to be saved with the named Variants (see page 6105) list.
	Error (see page 6098)	Raises an ENamedVariantsError (see page 6090) exception.
	Get (see page 6098)	Retrieves the name and Variant (see page 3430) stored at a specified position in the list.
	GetCapacity (see page 6099)	Returns the value of the Capacity (see page 6103) property.
	GetCount (see page 6099)	Returns the value of the Count (see page 6103) property.
	Put (see page 6101)	Stores a name/Variant (see page 3430) pair at a specified position in the list.
	SetCapacity (see page 6102)	Sets the value of the Capacity (see page 6103) property.
	SetUpdateState (see page 6102)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
	UpdateCount (see page 6104)	Indicates the number of calls to BeginUpdate (see page 6094) that have not been matched by a corresponding call to EndUpdate (see page 6097).

Public Methods

	Name	Description
	Add (see page 6093)	Adds a named Variant (see page 3430) to the end of the list.
	Append (see page 6094)	Adds a named Variant (see page 3430) to the end of the list.
	Assign (see page 6094)	Copies the list of named Variants (see page 6105) from another object.
	BeginUpdate (see page 6094)	Signals the start of an update operation.
	Clear (see page 6095)	Empties the list of all its name/Variant (see page 3430) pairs.
	Delete (see page 6096)	Deletes an entry from the list.

	Destroy (see page 6092)	Disposes of an object instance.
	EndUpdate (see page 6097)	Signals the end of an update operation.
	Equals (see page 6097)	Compares this named Variants (see page 6105) list with another one.
	Exchange (see page 6098)	Swaps the positions of two items in the list.
	IndexOfName (see page 6100)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6100)	Inserts a name/Variant (see page 3430) pair at a specified position.
	Move (see page 6101)	Moves a name/Variant (see page 3430) pair to a new position in the list.

Public Properties

	Name	Description
	Capacity (see page 6103)	Specifies the allocated size of the named variants list.
	Count (see page 6103)	Indicates the number of name/value pairs in the list.
	Names (see page 6104)	Lists the names of all the Variants (see page 6105) in the list.
	Values (see page 6105)	Lists the Variants (see page 6105) in the list by name.
	Variants (see page 6105)	Lists the Variants in the list by position.

62.1.2.2 TAbstractNamedVariants.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractNamedVariants();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release (see page 3732) method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

62.1.2.3 TAbstractNamedVariants Methods

The methods of the TAbstractNamedVariants class are listed here.

Protected Methods

	Name	Description
	CompareStrings (see page 6095)	Compares two strings.

	DefineProperties (see page 6096)	Allows the list of names and their corresponding Variant (see page 3430) values to be saved with the named Variants (see page 6105) list.
	Error (see page 6098)	Raises an ENamedVariantsError (see page 6090) exception.
	Get (see page 6098)	Retrieves the name and Variant (see page 3430) stored at a specified position in the list.
	GetCapacity (see page 6099)	Returns the value of the Capacity (see page 6103) property.
	GetCount (see page 6099)	Returns the value of the Count (see page 6103) property.
	Put (see page 6101)	Stores a name/Variant (see page 3430) pair at a specified position in the list.
	SetCapacity (see page 6102)	Sets the value of the Capacity (see page 6103) property.
	SetUpdateState (see page 6102)	Responds when an update operation begins or ends.

Public Methods

	Name	Description
	Add (see page 6093)	Adds a named Variant (see page 3430) to the end of the list.
	Append (see page 6094)	Adds a named Variant (see page 3430) to the end of the list.
	Assign (see page 6094)	Copies the list of named Variants (see page 6105) from another object.
	BeginUpdate (see page 6094)	Signals the start of an update operation.
	Clear (see page 6095)	Empties the list of all its name/Variant (see page 3430) pairs.
	Delete (see page 6096)	Deletes an entry from the list.
	EndUpdate (see page 6097)	Signals the end of an update operation.
	Equals (see page 6097)	Compares this named Variants (see page 6105) list with another one.
	Exchange (see page 6098)	Swaps the positions of two items in the list.
	IndexOfName (see page 6100)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6100)	Inserts a name/Variant (see page 3430) pair at a specified position.
	Move (see page 6101)	Moves a name/Variant (see page 3430) pair to a new position in the list.

62.1.2.3.1 TAbstractNamedVariants.Add Method

Adds a named Variant (see page 3430) to the end of the list.

Pascal

```
function Add(const S: string; const AValue: Variant): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const UnicodeString S, const Variant AValue);
```

Description

Call Add to add a name/Variant (see page 3430) pair to the end of the list. Add adds a new name/Variant (see page 3430) pair to the list with a position index equal to the value of the Count (see page 6103) property.

S is the name of the Variant (see page 3430) value. This can be used to look up the value using the Values (see page 6105) property.

AValue is the Variant (see page 3430) value to associate with S.

Note: Unlike setting the Values (see page 6105) property, calling Add adds a duplicate name/Variant (see page 3430) pair to the list if S matches the name of a Variant (see page 3430) that is already in the list.

See Also

Values (see page 6105)

Append (see page 6094)

Insert (see page 6100)

Delete (see page 6096)

Count (see page 6103)

62.1.2.3.2 TAbstractNamedVariants.Append Method

Adds a named Variant (see page 3430) to the end of the list.

Pascal

```
procedure Append(const S: string; const AValue: Variant);
```

C++

```
__fastcall Append(const UnicodeString S, const Variant AValue);
```

Description

Call Append to add a name/Variant (see page 3430) pair to the end of the list. Append simply calls the Add (see page 6093) method to add the specified name and its associated Variant (see page 3430) value.

S is the name of the Variant (see page 3430) value. This can be used to look up the value using the Values (see page 6105) property.

AValue is the Variant (see page 3430) value to associate with S.

See Also

Values (see page 6105)

Add (see page 6093)

Insert (see page 6100)

Delete (see page 6096)

62.1.2.3.3 TAbstractNamedVariants.Assign Method

Copies the list of named Variants (see page 6105) from another object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy another list to this named Variants (see page 6105) list. If the source object is another named Variants (see page 6105) list, Assign discards any name/Variant (see page 3430) pairs currently in the list and replaces them with the name/Variant (see page 3430) pairs in the source list. If the source is any other type of object, Assign calls the inherited method, which copies the list from any object that implements the AssignTo method so that it copies its items to a named Variants (see page 6105) list.

Source is the object from which to copy the list.

See Also

Values (see page 6105)

AssignTo

62.1.2.3.4 TAbstractNamedVariants.BeginUpdate Method

Signals the start of an update operation.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before starting an operation that makes multiple changes to the named Variants (see page 6105) list. After completing all of the changes, call the EndUpdate (see page 6097) method to signal the end of the operation.

Every call to BeginUpdate must be matched by a corresponding call to the EndUpdate (see page 6097) method.

Note: TAbstractNamedVariants (see page 6090) calls BeginUpdate and EndUpdate (see page 6097) internally when performing operations that affect multiple entries in the list.

See Also

EndUpdate (see page 6097)

62.1.2.3.5 TAbstractNamedVariants.Clear Method

Empties the list of all its name/Variant (see page 3430) pairs.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Call Clear to empty the named Variants (see page 6105) list. After a call to Clear, the value of the Count (see page 6103) property is zero.

TAbstractNamedVariants (see page 6090) declares Clear as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Count (see page 6103)

Delete (see page 6096)

62.1.2.3.6 TAbstractNamedVariants.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: string; const S2: string): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const UnicodeString S1, const UnicodeString S2);
```

Description

TAbstractNamedVariants (see page 6090) calls CompareStrings internally to locate names in the list (for example, in the implementation of the Values (see page 6105) property). Descendant classes can use this method to sort the list entries by name.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 = S2, and a value greater than 0 if S1 > S2.

In TAbstractNamedVariants (see page 6090), CompareStrings calls the AnsiCompareText (see page 4115) function. This performs a locale-based case-insensitive comparison. Override this method in a descendant class to change the way the named Variants (see page 6105) list compares names. For example, by changing CompareStrings to a case sensitive function, the Values (see page 6105) property would distinguish two names that differed only in case.

See Also

AnsiCompareText (see page 4115)

Values (see page 6105)

Equals (see page 6097)

62.1.2.3.7 TAbstractNamedVariants.DefineProperties Method

Allows the list of names and their corresponding Variant (see page 3430) values to be saved with the named Variants (see page 6105) list.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

The component streaming system calls DefineProperties when it loads or saves the named Variants (see page 6105) list. TAbstractNamedVariants (see page 6090) overrides the inherited method to specify how to load and save the list of names and Variant (see page 3430) values.

Filer is the current TReader or TWriter object that is loading or saving the named Variants (see page 6105) list.

See Also

Values (see page 6105)

DefineProperties

TReader

TWriter

62.1.2.3.8 TAbstractNamedVariants.Delete Method

Deletes an entry from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Call Delete to remove a specified name/Variant (see page 3430) pair from the list. After a call to Delete, any name/Variant (see page 3430) pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

TAbstractNamedVariants (see page 6090) declares Delete as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6104)

Variants (see page 6105)

Clear (see page 6095)

62.1.2.3.9 TAbstractNamedVariants.EndUpdate Method

Signals the end of an update operation.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after completing an operation that was preceded by a call to the BeginUpdate (see page 6094) method. Every call to BeginUpdate (see page 6094) must be matched by a corresponding call to the EndUpdate method. Calls to BeginUpdate (see page 6094) and EndUpdate can be nested.

Note: TAbstractNamedVariants (see page 6090) calls BeginUpdate (see page 6094) and EndUpdate internally when performing operations that affect multiple entries in the list.

See Also

BeginUpdate (see page 6094)

62.1.2.3.10 TAbstractNamedVariants.Equals Method

Compares this named Variants (see page 6105) list with another one.

Pascal

```
function Equals(ANamedVariants: TAbstractNamedVariants): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TAbstractNamedVariants ANamedVariants);
```

Description

Call Equals to determine whether another named Variants (see page 6105) list contains the same name/Variant (see page 3430) pairs as this one.

ANamedVariants is the named Variants (see page 6105) list to compare to this one.

Equals returns true if every entry in ANamedVariants has the same name and Variant (see page 3430) value as the corresponding entry in this list. Equals returns false if ANamedVariants has a different number of entries, if its entries appear in a different order, or if it contains different entries.

Equals considers two name to be the same if they differ only by case.

62.1.2.3.11 TAbstractNamedVariants.Error Method

Raises an ENamedVariantsError (see page 6090) exception.

Pascal

```
procedure Error(const Msg: string; Data: Integer); overload;
```

C++

```
__fastcall Error(const UnicodeString Msg, int Data);
```

Description

TAbstractNamedVariants (see page 6090) calls Error internally when it encounters an invalid index. For example, if an application tries to delete or change a name/Variant (see page 3430) pair and supplies an index that is greater than Count (see page 6103), TAbstractNamedVariants (see page 6090) calls Error to generate the exception about the out-of-bounds index.

Msg is the error message for the exception. This message contains a format specifier for the integer that represents the index that caused the problem.

Data is the index that caused the problem.

See Also

ENamedVariantsError (see page 6090)

Count (see page 6103)

62.1.2.3.12 TAbstractNamedVariants.Exchange Method

Swaps the positions of two items in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to interchange the positions of two name/Variant (see page 3430) pairs.

Index1 and Index2 are the indexes of the two entries to swap. Exchange copies the name and Variant (see page 3430) stored at Index1 to the position specified by Index2, and the name and Variant (see page 3430) from Index2 to the position specified by Index1.

See Also

Move (see page 6101)

62.1.2.3.13 TAbstractNamedVariants.Get Method

Retrieves the name and Variant (see page 3430) stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; virtual;
abstract;
```

C++

```
virtual __fastcall Boolean Get(int Index, UnicodeString AName, Variant AValue) = 0;
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants (see page 6105) list based on item position. For example, the property read methods for the Names (see page 6104) and Variants (see page 6105) properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant (see page 3430) to read.

AName returns the name stored at the specified position.

AValue returns the Variant (see page 3430) value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant (see page 3430) pair at the specified position.

TAbstractNamedVariants (see page 6090) declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6104)

Variants (see page 6105)

Put (see page 6101)

62.1.2.3.14 TAbstractNamedVariants.GetCapacity Method

Returns the value of the Capacity (see page 6103) property.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 6103) property.

In TAbstractNamedVariants (see page 6090), GetCapacity always returns the value of the Count (see page 6103) property, because TAbstractNamedVariants (see page 6090) does not implement support for preallocating memory to hold name/Variant (see page 3430) pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 6103)

Count (see page 6103)

62.1.2.3.15 TAbstractNamedVariants.GetCount Method

Returns the value of the Count (see page 6103) property.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCapacity (see page 6099) is the protected read implementation of the Count (see page 6103) property.

TAbstractNamedVariants (see page 6090) declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 6103) property.

See Also

Count (see page 6103)

62.1.2.3.16 TAbstractNamedVariants.IndexOfName Method

Returns the index of the first item in the list with a specified name.

Pascal

```
function IndexOfName(const Name: string): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const UnicodeString Name);
```

Description

Call IndexOfName to determine where a name appears in the list, if it appears at all.

Name is the name to locate in the list.

If Name appears as the name part of a name/Variant (see page 3430) pair in the list, IndexOfName returns the index of the first item with that name. If there are no items with the specified name, IndexOfName returns –1.

Note: In TAbstractNamedVariants (see page 6090), IndexOfName uses a case-insensitive comparison to locate the name. In descendant classes that treat strings case sensitively, IndexOfName makes a case sensitive comparison.

See Also

Names (see page 6104)

62.1.2.3.17 TAbstractNamedVariants.Insert Method

Inserts a name/Variant (see page 3430) pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S, const Variant AValue) = 0;
```

Description

Call Insert to add a name/Variant (see page 3430) pair to the list at a specified position. After a call to Insert, any name/Variant (see page 3430) pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant (see page 3430) should be added.

S is the name of the new entry.

AValue is the Variant (see page 3430) value of the new entry.

TAbstractNamedVariants (see page 6090) declares Insert as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Values (see page 6105)

Append (see page 6094)

Add (see page 6093)

Delete (see page 6096)

Count (see page 6103)

62.1.2.3.18 TAbstractNamedVariants.Move Method

Moves a name/Variant (see page 3430) pair to a new position in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Call move to reposition an item in the list. Move removes the item from its current position and inserts it in the new position, readjusting the positions (indexes) of other items as necessary.

CurIndex is the current position of the item to move.

NewIndex is the position to which the item is moved.

See Also

Exchange (see page 6098)

62.1.2.3.19 TAbstractNamedVariants.Put Method

Stores a name/Variant (see page 3430) pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString AName, const Variant AValue) = 0;
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants (see page 6105) list based on item position. For example, the property write methods for the Names (see page 6104) and Variants (see page 6105) properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant (see page 3430) to write.

AName is the name to store at the specified position.

AValue is the Variant (see page 3430) value to store at the specified position.

TAbstractNamedVariants (see page 6090) declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6104)

Variants (see page 6105)

Get (see page 6098)

62.1.2.3.20 TAbstractNamedVariants.SetCapacity Method

Sets the value of the Capacity (see page 6103) property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity (see page 6103) property.

In TAbstractNamedVariants (see page 6090), SetCapacity does nothing, because TAbstractNamedVariants (see page 6090) does not implement support for preallocating memory to hold name/Variant (see page 3430) pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 6103)

GetCapacity (see page 6099)

62.1.2.3.21 TAbstractNamedVariants.SetUpdateState Method

Responds when an update operation begins or ends.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

The BeginUpdate (see page 6094) and EndUpdate (see page 6097) methods call SetUpdate state when an update operation first starts or finally stops. If the calls to BeginUpdate (see page 6094) and EndUpdate (see page 6097) are nested, SetUpdate is only called by the first BeginUpdate (see page 6094) call and the last EndUpdate (see page 6097) call.

In TAbstractNamedVariants (see page 6090), SetUpdateState does nothing. Descendant classes can override this method to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 6094)

EndUpdate (see page 6097)

62.1.2.4 TAbstractNamedVariants Properties

The properties of the TAbstractNamedVariants class are listed here.

Protected Properties

	Name	Description
	UpdateCount (see page 6104)	Indicates the number of calls to BeginUpdate (see page 6094) that have not been matched by a corresponding call to EndUpdate (see page 6097).

Public Properties

	Name	Description
	Capacity (see page 6103)	Specifies the allocated size of the named variants list.
	Count (see page 6103)	Indicates the number of name/value pairs in the list.
	Names (see page 6104)	Lists the names of all the Variants (see page 6105) in the list.
	Values (see page 6105)	Lists the Variants (see page 6105) in the list by name.
	Variants (see page 6105)	Lists the Variants in the list by position.

62.1.2.4.1 TAbstractNamedVariants.Capacity Property

Specifies the allocated size of the named variants list.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to the number of named Variants (see page 6105) the list will need to contain. When a value is added to a list that is already filled to capacity, the Capacity property is automatically increased. Setting Capacity before adding values can reduce the number of memory reallocations and thereby improve performance. In addition, in descendants that allocate memory in chunks, setting Capacity can ensure that you do not allocate more memory for the list than your application needs.

Read (see page 3727) Capacity to learn number of values the list can hold without reallocating memory. Do not confuse Capacity with the Count (see page 6103) property, which is the number of entries in the list that are in use. The value of Capacity is always greater than or equal to the value of Count (see page 6103). When Capacity is greater than Count (see page 6103), the unused memory can be reclaimed by setting Capacity to Count (see page 6103).

See Also

Count (see page 6103)

Insert (see page 6100)

62.1.2.4.2 TAbstractNamedVariants.Count Property

Indicates the number of name/value pairs in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to determine the number of values the list contains. Count acts as an upper bound to the index that you can use for the Names (see page 6104) and Variants (see page 6105) properties.

See Also

Capacity (see page 6103)

Names (see page 6104)

Variants (see page 6105)

62.1.2.4.3 TAbstractNamedVariants.Names Property

Lists the names of all the Variants (see page 6105) in the list.

Pascal

```
property Names [Index: Integer]: string;
```

C++

```
__property UnicodeString Names[int Index];
```

Description

Read (see page 3727) Names to obtain the name of a Variant (see page 3430) in the list.

Index is the position of the name (and its corresponding Variant (see page 3430)) in the named Variants (see page 6105) list. The value of Index falls in the range from 0 to Count (see page 6103) – 1.

See Also

Count (see page 6103)

Values (see page 6105)

Variants (see page 6105)

62.1.2.4.4 TAbstractNamedVariants.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 6094) that have not been matched by a corresponding call to EndUpdate (see page 6097).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether the named Variants (see page 6105) list is in the middle of an update operation (an operation that makes several changes to the list). Every time the application calls BeginUpdate (see page 6094), the value of UpdateCount is incremented. Every call to EndUpdate (see page 6097) decrements the value of UpdateCount.

When UpdateCount changes from 0 to 1, and when it changes from 1 to 0, TAbstractNamedVariants (see page 6090) calls the SetUpdateState (see page 6102) method. This allows TAbstractNamedVariants (see page 6090) descendants to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 6094)

EndUpdate (see page 6097)

SetUpdateState (see page 6102)

62.1.2.4.5 TAbstractNamedVariants.Values Property

Lists the Variants (see page 6105) in the list by name.

Pascal

```
property Values [const Name: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const Name];
```

Description

Read (see page 3727) Values to look up the Variant (see page 3430) that is stored under a particular name. Set Values to change the Variant (see page 3430) associated with a name, or to add a new name and Variant (see page 3430) to the list.

Name is the name of the Variant (see page 3430). If there is no Variant (see page 3430) in the list with the specified name, reading Values gives an unassigned Variant (see page 3430), and setting Values adds a new name and Variant (see page 3430) to the list.

See Also

Count (see page 6103)

Values

Variants (see page 6105)

62.1.2.4.6 TAbstractNamedVariants.Variants Property

Lists the Variants in the list by position.

Pascal

```
property Variants [Index: Integer]: Variant;
```

C++

```
__property Variant Variants[int Index];
```

Description

Read (see page 3727) Variants to look up the Variant (see page 3430) that is stored at a particular position. Set Variants to change the Variant (see page 3430) stored in a particular position.

Index is the position of the Variant (see page 3430) in the named Variants list. The value of Index falls in the range from 0 to Count (see page 6103) – 1. If you specify a value of Index that is out of range, TAbstractNamedVariants (see page 6090) raises an ENamedVariantsError (see page 6090) exception.

See Also

Count (see page 6103)

Values (see page 6105)

Names (see page 6104)

ENamedVariantsError (see page 6090)

62.1.3 TNamedVariantsList Class

TNamedVariantsList is a persistent list of name/Variant (see page 3430) pairs.

Class Hierarchy

Pascal

```
TNamedVariantsList = class(TAbstractNamedVariants);
```

C++

```
class TNamedVariantsList : public TAbstractNamedVariants;
```

File

WebContnrs

Description

Use TNamedVariantsList to represent a set of name/value pairs where the values are all Variants (see page 4997). Because TNamedVariantsList is a descendant of TPersistent, it can be used as the type of a published property on a component.

TNamedVariantsList has properties and methods that let you:

Look up Variant (see page 3430) values by name or by index.

Locate a name in the list.

Add (see page 6093) or delete name/Variant (see page 3430) pairs.

Rearrange the items in the list.

Compare this list to another named variants list.

Note: TNamedVariantsList treats the names it stores in a case-insensitive manner. When locating a name in the list or assigning a Variant (see page 3430) value to an existing name, TNamedVariantsList treats names that differ only by case as the same.

See Also

TStringList

THashedStringList

TCollection

TList

TInterfaceList

62.1.3.1 TNamedVariantsList Members

The following tables list the members exposed by TNamedVariantsList.

Protected Methods

	Name	Description
	CompareStrings (see page 6095)	Compares two strings.
	DefineProperties (see page 6096)	Allows the list of names and their corresponding Variant (see page 3430) values to be saved with the named Variants (see page 6105) list.
	Error (see page 6098)	Raises an ENamedVariantsError (see page 6090) exception.
	Get (see page 6109)	Retrieves the name and Variant stored at a specified position in the list.
	GetCapacity (see page 6109)	Returns the value of the Capacity property.
	GetCount (see page 6110)	Returns the value of the Count property.
	Put (see page 6111)	Stores a name/Variant pair at a specified position in the list.
	SetCapacity (see page 6111)	Sets the value of the Capacity property.
	SetUpdateState (see page 6102)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
	UpdateCount (see page 6104)	Indicates the number of calls to BeginUpdate (see page 6094) that have not been matched by a corresponding call to EndUpdate (see page 6097).

Public Methods

	Name	Description
	Add (see page 6093)	Adds a named Variant (see page 3430) to the end of the list.
	Append (see page 6094)	Adds a named Variant (see page 3430) to the end of the list.
	Assign (see page 6094)	Copies the list of named Variants (see page 6105) from another object.
	BeginUpdate (see page 6094)	Signals the start of an update operation.
	Clear (see page 6108)	Deletes all the name/Variant (see page 3430) pairs from the list.
	Delete (see page 6108)	Removes a specified Variant (see page 3430) specified by the Index parameter.
	Destroy (see page 6107)	Frees an instance of TNamedVariantsList (see page 6106).
	EndUpdate (see page 6097)	Signals the end of an update operation.
	Equals (see page 6097)	Compares this named Variants (see page 6105) list with another one.
	Exchange (see page 6098)	Swaps the positions of two items in the list.
	IndexOfName (see page 6100)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6110)	Inserts a name/Variant (see page 3430) pair at a specified position.
	Move (see page 6101)	Moves a name/Variant (see page 3430) pair to a new position in the list.

Public Properties

	Name	Description
	Capacity (see page 6103)	Specifies the allocated size of the named variants list.
	Count (see page 6103)	Indicates the number of name/value pairs in the list.
	Names (see page 6104)	Lists the names of all the Variants (see page 6105) in the list.
	Values (see page 6105)	Lists the Variants (see page 6105) in the list by name.
	Variants (see page 6105)	Lists the Variants in the list by position.

62.1.3.2 TNamedVariantsList.Destroy Destructor

Frees an instance of TNamedVariantsList (see page 6106).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNamedVariantsList();
```

Description

Do not call the TNamedVariantsList (see page 6106) destructor directly. Instead, use the Free method, which first checks whether the reference to the object is nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

62.1.3.3 TNamedVariantsList Methods

The methods of the TNamedVariantsList class are listed here.

Protected Methods

	Name	Description
	Get (see page 6109)	Retrieves the name and Variant stored at a specified position in the list.
	GetCapacity (see page 6109)	Returns the value of the Capacity property.
	GetCount (see page 6110)	Returns the value of the Count property.
	Put (see page 6111)	Stores a name/Variant pair at a specified position in the list.
	SetCapacity (see page 6111)	Sets the value of the Capacity property.

Public Methods

	Name	Description
	Clear (see page 6108)	Deletes all the name/Variant (see page 3430) pairs from the list.
	Delete (see page 6108)	Removes a specified Variant (see page 3430) specified by the Index parameter.
	Insert (see page 6110)	Inserts a name/Variant (see page 3430) pair at a specified position.

62.1.3.3.1 TNamedVariantsList.Clear Method

Deletes all the name/Variant (see page 3430) pairs from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the list. Clear frees the memory that the Variants (see page 4997) in the list use to store their values as well as the memory used to store the list itself.

After a call to Clear, the value of the Count (see page 6103) property is zero.

See Also

Delete (see page 6108)

Variants (see page 6105)

Count

62.1.3.3.2 TNamedVariantsList.Delete Method

Removes a specified Variant (see page 3430) specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a specified name/Variant (see page 3430) pair from the list. Delete frees the Variant (see page 3430) when it removes the item from the list.

After a call to Delete, any name/Variant (see page 3430) pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

See Also

Clear (see page 6108)

IndexOfName (see page 6100)

62.1.3.3.3 TNamedVariantsList.Get Method

Retrieves the name and Variant stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean Get(int Index, UnicodeString AName, Variant AValue);
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants list based on item position. For example, the property read methods for the Names and Variants properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant to read.

AName returns the name stored at the specified position.

AValue returns the Variant value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant pair at the specified position.

TAbstractNamedVariants declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6104)

Variants (see page 6105)

Put (see page 6101)

62.1.3.3.4 TNamedVariantsList.GetCapacity Method

Returns the value of the Capacity property.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property.

In TAbstractNamedVariants, GetCapacity always returns the value of the Count property, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 6103)

Count (see page 6103)

62.1.3.3.5 TNamedVariantsList.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCapacity is the protected read implementation of the Count property.

TAbstractNamedVariants declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count property.

See Also

Count (see page 6103)

62.1.3.3.6 TNamedVariantsList.Insert Method

Inserts a name/Variant (see page 3430) pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S, const Variant AValue);
```

Description

Call Insert to add a name/Variant (see page 3430) pair to the list at a specified position. After a call to Insert, any name/Variant (see page 3430) pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant (see page 3430) should be added.

S is the name of the new entry.

AValue is the Variant (see page 3430) value of the new entry.

See Also

Values (see page 6105)

Append (see page 6094)

Add (see page 6093)

Delete (see page 6108)

Count

62.1.3.3.7 TNamedVariantsList.Put Method

Stores a name/Variant pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString AName, const Variant AValue);
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants list based on item position. For example, the property write methods for the Names and Variants properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant to write.

AName is the name to store at the specified position.

AValue is the Variant value to store at the specified position.

TAbstractNamedVariants declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6104)

Variants (see page 6105)

Get (see page 6098)

62.1.3.3.8 TNamedVariantsList.SetCapacity Method

Sets the value of the Capacity property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity property.

In TAbstractNamedVariants, SetCapacity does nothing, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support

for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 6103)

GetCapacity (see page 6099)

62.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TNamedVariantItem (see page 6113)	This is record WebContnrs.TNamedVariantItem.

62.2.1 WebContnrs.TNamedVariantItem Record

Pascal

```
TNamedVariantItem = record
  FString: string;
  FVariant: Variant;
end;
```

C++

```
struct TNamedVariantItem {
  UnicodeString FString;
  Variant FVariant;
};
```

File

WebContnrs

Description

This is record WebContnrs.TNamedVariantItem.

62.3 Types

The following table lists types in this documentation.

Types

Name	Description
PNamedVariantItem (see page 6114)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 6114)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 6114)	This is type WebContnrs.TNamedVariantList.

62.3.1 WebContnrs.PNamedVariantItem Type

Pascal

```
PNamedVariantItem = ^TNamedVariantItem;
```

C++

```
^TNamedVariantItem PNamedVariantItem;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantItem.

62.3.2 WebContnrs.PNamedVariantList Type

Pascal

```
PNamedVariantList = ^TNamedVariantList;
```

C++

```
^TNamedVariantList PNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantList.

62.3.3 WebContnrs.TNamedVariantList Type

Pascal

```
TNamedVariantList = array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem;
```

C++

```
array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem TNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.TNamedVariantList.

63 WebDisp Namespace

This is namespace WebDisp.

Topics

Name	Description
Classes (see page 6120)	The following table lists classes in this documentation.
Interfaces (see page 6231)	The following table lists interfaces in this documentation.
Functions (see page 6233)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6239)	The following table lists structs, records, enums in this documentation.
Types (see page 6240)	The following table lists types in this documentation.
Constants (see page 6247)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 6120)	eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 6121)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 6122)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 6122)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 6128)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 6134)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 6140)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 6142)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 6155)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 6162)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 6175)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 6190)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 6194)	TLocateFileService controls the access of templates and server-side script include files at runtime.

	TMultiModuleWebAppServices (see page 6197)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 6210)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 6217)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 6225)	TWebSession stores temporary information associated with the user in a WebSnap application.

Constants

Name	Description
sFailurePageParamName (see page 6247)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 6247)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 6248)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 6248)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 6248)	This is constant WebDisp.sSuccessPageParamName.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 6233)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 6233)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 6234)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 6234)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 6234)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 6235)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 6235)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 6235)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 6236)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 6236)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 6236)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 6237)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 6237)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 6237)	Redirects a browser to another specified page.

Interfaces

	Name	Description
	IWebSessionAccess (see page 6231)	This is class WebDisp.IWebSessionAccess.

Structs, Records, Enums

	Name	Description
	TPageAccessDenied (see page 6239)	This is record WebDisp.TPageAccessDenied.

Types

Name	Description
TActivateWebModuleEvent (see page 6240)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 6240)	This is type WebDisp.TCanViewPageEvent.
TDispatchActionEvent (see page 6241)	This is type WebDisp.TDispatchActionEvent.
TDispatchActionHandledEvent (see page 6241)	This is type WebDisp.TDispatchActionHandledEvent.
TDispatchImageEvent (see page 6241)	This is type WebDisp.TDispatchImageEvent.
TDispatchImageHandledEvent (see page 6242)	This is type WebDisp.TDispatchImageHandledEvent.
TDispatchPageEvent (see page 6242)	This is type WebDisp.TDispatchPageEvent.
TDispatchPageHandledEvent (see page 6242)	This is type WebDisp.TDispatchPageHandledEvent.
TDispatchPageParamsHandledEvent (see page 6243)	This is type WebDisp.TDispatchPageParamsHandledEvent.
TGetAdapterRequestParamsEvent (see page 6243)	This is type WebDisp.TGetAdapterRequestParamsEvent.
TLocateFileServiceFindFileEvent (see page 6243)	This is type WebDisp.TLocateFileServiceFindFileEvent.
TLocateFileServiceFindStreamEvent (see page 6244)	This is type WebDisp.TLocateFileServiceFindStreamEvent.

TLoginPagedHandledEvent (see page 6244)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 6244)	This is type WebDisp.TLoginRequiredEvent.
TPageAccessDeniedEvent (see page 6245)	This is type WebDisp.TPageAccessDeniedEvent.
TWebRequest (see page 6245)	This is type WebDisp.TWebRequest.
TWebResponse (see page 6245)	This is type WebDisp.TWebResponse.

63.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 6120)	eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 6121)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 6122)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 6122)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 6128)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 6134)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 6140)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 6142)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 6155)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 6162)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 6175)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 6190)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 6194)	TLocateFileService controls the access of templates and server-side script include files at runtime.
	TMultiModuleWebAppServices (see page 6197)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 6210)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 6217)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 6225)	TWebSession stores temporary information associated with the user in a WebSnap application.

63.1.1 EAdapterDispatcherException Class

eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.

Class Hierarchy

Pascal

```
EAdapterDispatcherException = class(EWebBrokerException);
```

C++

```
class EAdapterDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

When the TAdapterDispatcher (see page 6122) component can't successfully dispatch an action or obtain a dynamic image in response to a request, it generates an OnActionRequestNotHandled or an OnImageRequestNotHandled event. If there is no event handler for that event, or if the event handler does not handle the situation, TAdapterDispatcher (see page 6122) raises an eadapterdispatcherexception exception.

See Also

TCustomAdapterDispatcher (see page 6142)

TCustomAdapterDispatcher (see page 6142)

63.1.2 EPageDispatcherException Class

EPageDispatcherException raises an exception when a page dispatcher encounters an error.

Class Hierarchy

Pascal

```
EPageDispatcherException = class(EWebBrokerException);
```

C++

```
class EPageDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

EPageDispatcherException represents exceptions that occur when a page dispatcher tries to obtain the content of a Web page in response to an incoming HTTP request. These exceptions include

The incoming request does not request a specific Web page and the application does not provide a default Web page to handle this situation.

The current user does not have access rights for the requested page.

The page dispatcher could not locate a Web page module to provide the requested page.

The Web module with the same name as the requested page can't produce a Web page.

The application is attempting to send a response that redirects the user to a page that the dispatcher can't locate.

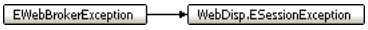
See Also

TPageDispatcher (see page 6210)

63.1.3 ESessionException Class

ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.

Class Hierarchy



Pascal

```
ESessionException = class(EWebBrokerException);
```

C++

```
class ESessionException : public EWebBrokerException;
```

File

WebDisp

Description

The TInvalidWebSession (see page 6190) class raises ESessionException when the application tries to use the current session and that session could not be created. This includes attempts to change the properties of the current session (such as setting the time-out period), attempts to get or set values, or attempts to terminate the current session.

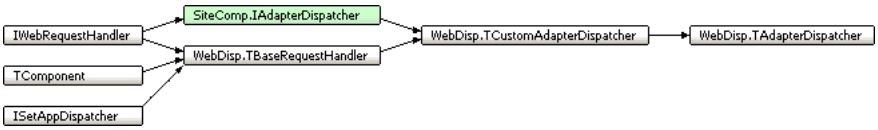
See Also

TInvalidWebSession (see page 6190)

63.1.4 TAdapterDispatcher Class

TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy



Pascal

```
TAdapterDispatcher = class(TCustomAdapterDispatcher);
```

C++

```
class TAdapterDispatcher : public TCustomAdapterDispatcher;
```

File

WebDisp

Description

TAdapterDispatcher is one of the default application components that is added when creating a WebSnap application. Events are triggered before and after dispatching an action or image, when adapter parameters are requested, and in cases when the action or image request was not handled.

Note: Your web application will be unable to execute adapter actions or obtain images from adapter fields unless it includes a TAdapterDispatcher component.

See Also

TAdapter (see page 5179)

TCustomWebAppComponents (see page 6175)

63.1.4.1 TAdapterDispatcher Members

The following tables list the members exposed by TAdapterDispatcher.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6144)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6145)	Generates an OnActionRequestNotHandled (see page 6151) and an OnImageRequestNotHandled (see page 6154) event.
	DoAfterDispatchAdapterRequest (see page 6146)	Generates an OnAfterDispatchAction (see page 6152) and an OnAfterDispatchImage (see page 6152) event.
	DoBeforeDispatchAdapterRequest (see page 6146)	Generates an OnBeforeDispatchAction (see page 6153) and an OnBeforeDispatchImage (see page 6153) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6147)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6147)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6148)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6148)	Parses a request to determine the adapter to which it is directed.
	HandleRequest (see page 6141)	Handles an HTTP request.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6147) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6147) method.
	ImplGetAdapterItemRequestParams (see page 6150)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6148) method.
	ImplHandleRequest (see page 6141)	Handles application-specific processing of an HTTP request.
	ImplHandleRequest (see page 6150)	Provides the underlying implementation for the HandleRequest (see page 6141) method.
	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.

Protected Properties

	Name	Description
	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 928)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 928)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 929)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 6124)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6125)	Occurs after the target adapter has finished executing an action.

	OnAfterDispatchImage (see page 6125)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6125)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6126)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6126)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6127)	Occurs when the WebSnap application could not provide a requested image.

63.1.4.2 TAdapterDispatcher Events

The events of the TAdapterDispatcher class are listed here.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 6124)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6125)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6125)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6125)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6126)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6126)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6127)	Occurs when the WebSnap application could not provide a requested image.

63.1.4.2.1 TAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates an OnImageRequestNotHandled. If the OnImageRequestNotHandled event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchAction (see page 6152)

OnBeforeDispatchAction (see page 6153)

EAdapterDispatcherException (see page 6120)

WebContext (see page 5977)

Response (see page 5949)

OnImageRequestNotHandled (see page 6154)

63.1.4.2.2 TAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 6151)

OnBeforeDispatchAction (see page 6153)

63.1.4.2.3 TAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 6154)

OnBeforeDispatchImage (see page 6153)

63.1.4.2.4 TAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated

adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnActionRequestNotHandled (see page 6151)

OnAfterDispatchAction (see page 6152)

WebContext (see page 5977)

Response (see page 5949)

63.1.4.2.5 TAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnImageRequestNotHandled (see page 6154)

OnAfterDispatchImage (see page 6152)

WebContext (see page 5977)

Response (see page 5949)

63.1.4.2.6 TAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 6148)

EncodeAdapterItemRequestAsFieldValue (see page 6147)

EncodeAdapterItemRequestAsHREF (see page 6147)

AddSpecialParams (see page 6144)

63.1.4.2.7 TAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchImage (see page 6152)

OnBeforeDispatchImage (see page 6153)

EAdapterDispatcherException (see page 6120)

WebContext (see page 5977)

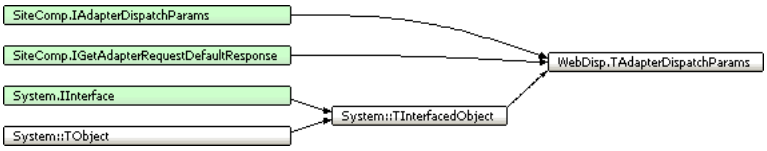
Response (see page 5949)

OnActionRequestNotHandled (see page 6151)

63.1.5 TAdapterDispatchParams Class

TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.

Class Hierarchy



Pascal

```
TAdapterDispatchParams = class(TInterfacedObject, IAdapterDispatchParams,
IGetAdapterRequestDefaultResponse);
```

C++

```
class TAdapterDispatchParams : public TInterfacedObject, public IAdapterDispatchParams, public
IGetAdapterRequestDefaultResponse;
```

File

WebDisp

Description

TAdapterDispatcher (see page 6122) uses TAdapterDispatchParams as a helper object for locating the adapter that can handle a request message. The TAdapterDispatchParams constructor takes a request message as an argument, and parses the request to determine the adapter, if any, that should handle it and the parameters that the adapter needs to handle the request.

The TAdapterDispatchParms properties identify the adapter that can process the request, the name of the action or field that the adapter uses to process the request, and the type of request (an action to execute or a field value to retrieve).

See Also

TAdapterDispatcher (see page 6122)

IAdapterDispatchParams (see page 929)

63.1.5.1 TAdapterDispatchParams Members

The following tables list the members exposed by TAdapterDispatchParams.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 6131)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 6131)	Returns the value of the Handler property.

	GetObjectIdentifier (see page 6132)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 6132)	Returns the value of the Params property.
	GetRequestIdentifier (see page 6132)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 6132)	This is GetSuccessPage, a member of class TAdapterDispatchParams.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6130)	Creates and initializes a TAdapterDispatchParams (see page 6128) object based on the information in an HTTP request.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6130)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	FailurePage (see page 968)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.

R	Handler (see page 6133)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 6133)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 933)	Indicates the parameter values to use when executing the action or looking up the image.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
R	RequestIdentifier (see page 6134)	Identifies the type of the request identified by the ObjectIdentifier (see page 6133) property.
R	SuccessPage (see page 968)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

63.1.5.2 TAdapterDispatchParams.Create Constructor

Creates and initializes a TAdapterDispatchParams (see page 6128) object based on the information in an HTTP request.

Pascal

```
constructor Create(ARequest: TWebRequest);
```

C++

```
__fastcall TAdapterDispatchParams(TWebRequest ARequest);
```

Description

Call Create to create a new TAdapterDispatchParams (see page 6128) object and use it to obtain information encoded in an HTTP request message.

The TAdapterDispatchParams (see page 6128) constructor parses the request message that is supplied as ARequest, locating the values that identify the adapter that must be used to handle the request, and the name and type of the information that that adapter supplies. Create uses this information to initialize the Handler (see page 6133), ObjectIdentifier (see page 6133), and RequestIdentifier (see page 6134) properties.

See Also

Handler (see page 6133)

ObjectIdentifier (see page 6133)

RequestIdentifier (see page 6134)

63.1.5.3 TAdapterDispatchParams.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDispatchParams();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

63.1.5.4 TAdapterDispatchParams Methods

The methods of the TAdapterDispatchParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 6131)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 6131)	Returns the value of the Handler property.
	GetObjectIdentifier (see page 6132)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 6132)	Returns the value of the Params property.
	GetRequestIdentifier (see page 6132)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 6132)	This is GetSuccessPage, a member of class TAdapterDispatchParams.

63.1.5.4.1 TAdapterDispatchParams.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

This is GetFailurePage, a member of class TAdapterDispatchParams.

63.1.5.4.2 TAdapterDispatchParams.GetHandler Method

Returns the value of the Handler property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler property.

See Also

Handler (see page 932)

63.1.5.4.3 TAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall UnicodeString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier property.

See Also

ObjectIdentifier (see page 932)

63.1.5.4.4 TAdapterDispatchParams.GetParams Method

Returns the value of the Params property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params property.

See Also

Params (see page 933)

63.1.5.4.5 TAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall UnicodeString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier property.

See Also

RequestIdentifier (see page 933)

63.1.5.4.6 TAdapterDispatchParams.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class TAdapterDispatchParams.

63.1.5.5 TAdapterDispatchParams Properties

The properties of the TAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
	Handler (see page 6133)	Indicates the adapter object that can handle the current request.
	ObjectIdentifier (see page 6133)	Specifies the name of the action to execute or the field to retrieve.
	RequestIdentifier (see page 6134)	Identifies the type of the request identified by the ObjectIdentifier (see page 6133) property.

63.1.5.5.1 TAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read (see page 3727) Handler to access the adapter that can execute the action or supply the image that is requested. The TAdapterDispatchParams (see page 6128) constructor initializes this property based on the information it parses from the current request message.

See Also

Create (see page 6130)

ObjectIdentifier (see page 6133)

RequestIdentifier (see page 6134)

63.1.5.5.2 TAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property UnicodeString ObjectIdentifier;
```

Description

Read (see page 3727) ObjectIdentifier to determine the name of the requested action or field. The TAdapterDispatchParams (see page 6128) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 6133) parameter.

See Also

Create (see page 6130)

Handler (see page 6133)

RequestIdentifier (see page 6134)

63.1.5.5.3 TAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 6133) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property UnicodeString RequestIdentifier;
```

Description

Read (see page 3727) RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action. The TAdapterDispatchParams (see page 6128) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 6133) parameter.

See Also

Create (see page 6130)

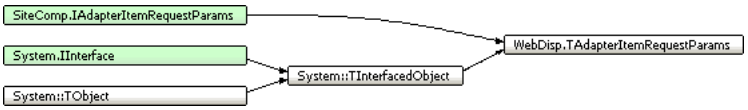
Handler (see page 6133)

ObjectIdentifier (see page 6133)

63.1.6 TAdapterItemRequestParams Class

TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.

Class Hierarchy



Pascal

```
TAdapterItemRequestParams = class(TInterfacedObject, IAdapterItemRequestParams);
```

C++

```
class TAdapterItemRequestParams : public TInterfacedObject, public IAdapterItemRequestParams;
```

File

WebDisp

Description

TAdapterItemRequestParams is the default implementation of the IAdapterItemRequestParams (see page 947) interface. It holds the name/value pairs that should be included as request parameters to an adapter or one of its subcomponents.

The parameters, which can be accessed through the ParamValues (see page 6139) property, are a set of 'Name=Value' strings. In addition to holding these parameters, TAdapterItemRequestParams has two properties, SuccessPage (see page 6140) and FailurePage (see page 6139), which identify the Web pages that should be returned to the caller when the request succeeds or fails.

See Also

TAdapterDispatcher (see page 6122)

IAdapterItemRequestParams (see page 947)

GetAdapterItemRequestParams

63.1.6.1 TAdapterItemRequestParams Members

The following tables list the members exposed by TAdapterItemRequestParams.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 6137)	Returns the value of the FailurePage property.
	GetParamValues (see page 6137)	Returns the value of the ParamValues property.
	GetSuccessPage (see page 6138)	Returns the value of the SuccessPage property.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	SetFailurePage (see page 6138)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 6138)	Sets the value of the SuccessPage property.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 6136)	Creates and initializes a TAdapterItemRequestParams (see page 6134) object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6137)	Frees an instance of TAdapterItemRequestParams (see page 6134).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	FailurePage (see page 6139)	Indicates the Web page that the application should return if the current request fails.
R	ParamValues (see page 6139)	Lists the parameter values that are included in requests to the adapter action or field.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	SuccessPage (see page 6140)	Indicates the Web page that the application should return if the request succeeds.

63.1.6.2 TAdapterItemRequestParams.Create Constructor

Creates and initializes a TAdapterItemRequestParams (see page 6134) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterItemRequestParams();
```

Description

Typically, applications have no need to create TAdapterItemRequestParams (see page 6134) instances. The adapter dispatcher creates these objects to hold the parameters for a request.

See Also

TAdapterDispatcher (see page 6122)

63.1.6.3 TAdapterItemRequestParams.Destroy Destructor

Frees an instance of TAdapterItemRequestParams (see page 6134).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterItemRequestParams();
```

Description

Most applications have no need to create or free TAdapterItemRequestParams (see page 6134) instances. These objects are managed by the adapter dispatcher.

See Also

TAdapterDispatcher (see page 6122)

63.1.6.4 TAdapterItemRequestParams Methods

The methods of the TAdapterItemRequestParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 6137)	Returns the value of the FailurePage property.
	GetParamValues (see page 6137)	Returns the value of the ParamValues property.
	GetSuccessPage (see page 6138)	Returns the value of the SuccessPage property.
	SetFailurePage (see page 6138)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 6138)	Sets the value of the SuccessPage property.

63.1.6.4.1 TAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage property.

See Also

FailurePage (see page 950)

63.1.6.4.2 TAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues property.

See Also

ParamValues (see page 951)

63.1.6.4.3 TAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage property.

See Also

SuccessPage (see page 951)

63.1.6.4.4 TAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const UnicodeString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 950)

63.1.6.4.5 TAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const UnicodeString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 951)

63.1.6.5 TAdapterItemRequestParams Properties

The properties of the TAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 6139)	Indicates the Web page that the application should return if the current request fails.
R	ParamValues (see page 6139)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 6140)	Indicates the Web page that the application should return if the request succeeds.

63.1.6.5.1 TAdapterItemRequestParams.FailurePage Property

Indicates the Web page that the application should return if the current request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 6140)

63.1.6.5.2 TAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names

Values

63.1.6.5.3 TAdapterItemRequestParams.SuccessPage Property

Indicates the Web page that the application should return if the request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

```
__property UnicodeString SuccessPage;
```

Description

SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

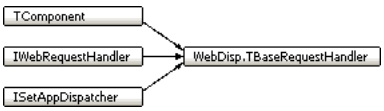
See Also

SuccessPage

63.1.7 TBaseRequestHandler Class

The TBaseRequestHandler component is a base class for WebSnap dispatcher components.

Class Hierarchy



Pascal

```
TBaseRequestHandler = class(TComponent, IWebRequestHandler, ISetAppDispatcher);
```

C++

```
class TBaseRequestHandler : public TComponent, public IWebRequestHandler, public
ISetAppDispatcher;
```

File

WebDisp

Description

The TCustomAdapterDispatcher (see page 6142) and TCustomPageDispatcher (see page 6162) components are derived from TBaseRequestHandler.

63.1.7.1 TBaseRequestHandler Members

The following tables list the members exposed by TBaseRequestHandler.

Protected Methods

	Name	Description
	HandleRequest (see page 6141)	Handles an HTTP request.
	ImplHandleRequest (see page 6141)	Handles application-specific processing of an HTTP request.

	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.

Protected Properties

	Name	Description
R	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

63.1.7.2 TBaseRequestHandler Methods

The methods of the TBaseRequestHandler class are listed here.

Protected Methods

	Name	Description
	HandleRequest (see page 6141)	Handles an HTTP request.
V	ImplHandleRequest (see page 6141)	Handles application-specific processing of an HTTP request.
	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.

63.1.7.2.1 TBaseRequestHandler.HandleRequest Method

Handles an HTTP request.

Pascal

```
function HandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

HandleRequest returns true if it handles the request specified by ARequest. This function calls the ImplHandleRequest (see page 6141) to process the request. A customized overridden function should be written for ImplHandleRequest (see page 6141) to do the necessary processing specific to the application.

See Also

ImplHandleRequest (see page 6141)

63.1.7.2.2 TBaseRequestHandler.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

The ImplHandleRequest function is called from HandleRequest (see page 6141) to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 6141)

63.1.7.2.3 TBaseRequestHandler.SetAppDispatcher Method

Sets the AppDispatcher (see page 6142) property.

Pascal

```
procedure SetAppDispatcher(const AWebAppDispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * AWebAppDispatcher);
```

Description

SetAppDispatcher provides a way to set the AppDispatcher (see page 6142) property.

See Also

AppDispatcher (see page 6142)

63.1.7.3 TBaseRequestHandler Properties

The properties of the TBaseRequestHandler class are listed here.

Protected Properties

	Name	Description
	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

63.1.7.3.1 TBaseRequestHandler.AppDispatcher Property

Provides read-only access to the application dispatcher component.

Pascal

```
property AppDispatcher: TComponent;
```

C++

```
__property TComponent * AppDispatcher;
```

Description

The application dispatcher component is typically TWebAppComponents (see page 6217). The application dispatcher component is used to access application objects such as TWebRequest (see page 6245) and application components such as TEndUserAdapter.

See Also

TCustomAdapterDispatcher (see page 6142)

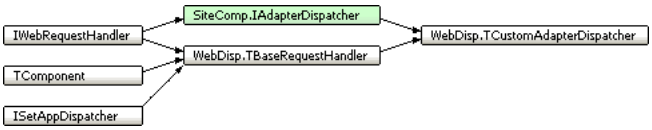
TCustomPageDispatcher (see page 6162)

TWebAppComponents (see page 6217)

63.1.8 TCustomAdapterDispatcher Class

TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy



Pascal

```
TCustomAdapterDispatcher = class(TBaseRequestHandler, IAdapterDispatcher);
```

C++

```
class TCustomAdapterDispatcher : public TBaseRequestHandler, public IAdapterDispatcher;
```

File

WebDisp

Description

TCustomAdapterDispatcher is the base class for the adapter dispatcher. The adapter dispatcher is responsible for dispatching adapter actions and references to image fields. It also handles the encoding of parameters that must be included in HTML form submissions that invoke those adapter actions or field references. If a WebSnap application does not include an AdapterDispatcher component, then it can't execute adapter actions or obtain images from adapter fields.

See Also

TAdapter (see page 5179)

TAdapterDispatcher (see page 6122)

TCustomWebAppComponents (see page 6175)

63.1.8.1 TCustomAdapterDispatcher Members

The following tables list the members exposed by TCustomAdapterDispatcher.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6144)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6145)	Generates an OnActionRequestNotHandled (see page 6151) and an OnImageRequestNotHandled (see page 6154) event.
	DoAfterDispatchAdapterRequest (see page 6146)	Generates an OnAfterDispatchAction (see page 6152) and an OnAfterDispatchImage (see page 6152) event.
	DoBeforeDispatchAdapterRequest (see page 6146)	Generates an OnBeforeDispatchAction (see page 6153) and an OnBeforeDispatchImage (see page 6153) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6147)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6147)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6148)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6148)	Parses a request to determine the adapter to which it is directed.
	HandleRequest (see page 6141)	Handles an HTTP request.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6147) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6147) method.
	ImplGetAdapterItemRequestParams (see page 6150)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6148) method.

	ImplHandleRequest (see page 6150)	Provides the underlying implementation for the HandleRequest (see page 6141) method.
	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.

Protected Properties

	Name	Description
	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 6151)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6152)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6152)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6153)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6153)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6154)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6154)	Occurs when the WebSnap application could not provide a requested image.

63.1.8.2 TCustomAdapterDispatcher Methods

The methods of the TCustomAdapterDispatcher class are listed here.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6144)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6145)	Generates an OnActionRequestNotHandled (see page 6151) and an OnImageRequestNotHandled (see page 6154) event.
	DoAfterDispatchAdapterRequest (see page 6146)	Generates an OnAfterDispatchAction (see page 6152) and an OnAfterDispatchImage (see page 6152) event.
	DoBeforeDispatchAdapterRequest (see page 6146)	Generates an OnBeforeDispatchAction (see page 6153) and an OnBeforeDispatchImage (see page 6153) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6147)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6147)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6148)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6148)	Parses a request to determine the adapter to which it is directed.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6147) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6149)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6147) method.
	ImplGetAdapterItemRequestParams (see page 6150)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6148) method.
	ImplHandleRequest (see page 6150)	Provides the underlying implementation for the HandleRequest (see page 6141) method.

63.1.8.2.1 TCustomAdapterDispatcher.AddSpecialParams Method

Adds parameters for the success and failure Web pages to the list of adapter parameters.

Pascal

```
procedure AddSpecialParams(AHandler: IUnknown; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall AddSpecialParams(IUnknown AHandler, IAdapterItemRequestParams Params);
```

Description

The adapter dispatcher calls AddSpecialParams internally after retrieving a set of parameters from an adapter (handler) and before generating an OnGetAdapterRequestParams (see page 6154) event. This method adds parameters to the list that are not required by the adapter for handling the request.

AddSpecialParams retrieves the Web pages that should be displayed when the adapter succeeds or fails in processing the request with which the handler is associated. If the handler has any special pages to display, AddSpecialParams adds the names of these pages to the parameters, using the parameter names of '__sp' and '__fp' (for success page and failure page), or '__p' (if the same page should be used for both success and failure).

AHandler is the interface for the adapter associated with the action or image.

Params is an interface that provides access to the current list of parameters.

See Also

GetAdapterItemRequestParams (see page 6148)

OnGetAdapterRequestParams (see page 6154)

FailurePage (see page 950)

SuccessPage (see page 951)

63.1.8.2.2 TCustomAdapterDispatcher.DoAdapterRequestNotHandled Method

Generates an OnActionRequestNotHandled (see page 6151) and an OnImageRequestNotHandled (see page 6154) event.

Pascal

```
procedure DoAdapterRequestNotHandled(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoAdapterRequestNotHandled(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 6142) calls DoAdapterRequestNotHandled internally when an adapter request remains unhandled. The request may be unhandled because the adapter could not execute the requested action or obtain the requested image, or it may be unhandled because the dispatcher could not locate an adapter to handle the request.

DoAdapterRequestNotHandled assembles the parameters for an action request and generates an OnActionRequestNotHandled (see page 6151) event. It then assembles the parameters for an image request and generates an OnImageRequestNotHandled (see page 6154) event.

AObject is the adapter that failed to handle the request (or nil (Delphi) or NULL (C++) if the dispatcher could not locate an adapter for the request).

AHandled returns true if an event handler handled the request.

See Also

OnActionRequestNotHandled (see page 6151)

OnImageRequestNotHandled (see page 6154)

63.1.8.2.3 TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest Method

Generates an OnAfterDispatchAction (see page 6152) and an OnAfterDispatchImage (see page 6152) event.

Pascal

```
procedure DoAfterDispatchAdapterRequest(AObject: TObject);
```

C++

```
__fastcall DoAfterDispatchAdapterRequest(TObject * AObject);
```

Description

TCustomAdapterDispatcher (see page 6142) calls DoAfterDispatchAdapterRequest internally after calling on an adapter to handle a request. It calls this method both when the adapter indicates success and when it indicates failure in handling the request.

DoAfterDispatchAdapterRequest assembles the parameters for an action request and generates an OnAfterDispatchAction (see page 6152) event. It then assembles the parameters for an image request and generates an OnAfterDispatchImage (see page 6152) event.

AObject is the adapter that just handled the current request.

See Also

OnAfterDispatchAction (see page 6152)

OnAfterDispatchImage (see page 6152)

63.1.8.2.4 TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest Method

Generates an OnBeforeDispatchAction (see page 6153) and an OnBeforeDispatchImage (see page 6153) event.

Pascal

```
procedure DoBeforeDispatchAdapterRequest(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoBeforeDispatchAdapterRequest(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 6142) calls DoBeforeDispatchAdapterRequest internally before calling on an adapter to handle a request. It calls this method only after it has identified the appropriate handler for responding to the current request.

DoBeforeDispatchAdapterRequest assembles the parameters for an action request and generates an OnBeforeDispatchAction (see page 6153) event. It then assembles the parameters for an image request and generates an OnBeforeDispatchImage (see page 6153) event.

AObject is the adapter for handling the current request.

AHandled returns true if an event handler handled the request.

See Also

OnBeforeDispatchAction (see page 6153)

OnBeforeDispatchImage (see page 6153)

63.1.8.2.5 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsFieldValue to encode the action or field parameters as they should appear as field values in form submissions. The resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 6148)

EncodeAdapterItemRequestAsHREF (see page 6147)

63.1.8.2.6 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsHREF to encode the action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 6148)

EncodeAdapterItemRequestAsFieldValue (see page 6147)

63.1.8.2.7 TCustomAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams retrieves the parameters for a call to execute an adapter action or to fetch the value of an image field.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

GetAdapterItemRequestParams simply calls the ImplGetAdapterItemRequestParams (see page 6150) method, which fetches the parameter values and writes them to AParams.

See Also

EncodeAdapterItemRequestAsHREF (see page 6147)

EncodeAdapterItemRequestAsFieldValue (see page 6147)

ImplGetAdapterItemRequestParams (see page 6150)

63.1.8.2.8 TCustomAdapterDispatcher.GetDispatchParams Method

Parses a request to determine the adapter to which it is directed.

Pascal

```
function GetDispatchParams(Request: TWebRequest): TAdapterDispatchParams; virtual;
```

C++

```
virtual __fastcall TAdapterDispatchParams GetDispatchParams(TWebRequest Request);
```

Description

TCustomAdapterDispatcher (see page 6142) calls GetDispatchParams internally to obtain information about the adapter to which a request is directed. GetDispatchParams creates a new TAdapterDispatchParams (see page 6128) object, which parses the current request to determine the adapter to which it is directed.

Request is the current http request object.

GetDispatchParams returns the new TAdapterDispatchParams (see page 6128) object, whose properties have been filled out to reflect the information in the current request. It is the caller's responsibility to free this object.

63.1.8.2.9
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6147) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplEncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsFieldValue (see page 6147) method calls ImplEncodeAdapterItemRequestAsFieldValue to retrieve a set of parameters formatted as a set of field values for an HTTP request. ImplEncodeAdapterItemRequestAsFieldValue calls GetAdapterItemRequestParams (see page 6148) to fetch the parameter values, and then formats them as a set of field values.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

EncodeAdapterItemRequestAsFieldValue (see page 6147)

GetAdapterItemRequestParams (see page 6148)

63.1.8.2.10
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6147) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplEncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsHREF (see page 6147) method calls ImplEncodeAdapterItemRequestAsFieldValue (see page 6149) to retrieve a set of parameters formatted as the parameters to an anchor tag in an HTML form. ImplEncodeAdapterItemRequestAsHREF calls GetAdapterItemRequestParams (see page 6148) to fetch the parameter values, and then formats them.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF (see page 6147) returns the parameter string.

See Also

EncodeAdapterItemRequestAsHREF (see page 6147)

GetAdapterItemRequestParams (see page 6148)

63.1.8.2.11 TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams Method

Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6148) method.

Pascal

```
procedure ImplGetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams (see page 6148) calls ImplGetAdapterItemRequestParams to retrieve the parameters for a call to execute an adapter action or to fetch the value of an image field. This method fetches the requested parameters from the adapter interface, calls the AddSpecialParams (see page 6144) method to add additional parameters for the return Web page, and then generates an OnGetAdapterRequestParams (see page 6154) event so that you can make your own final adjustments to the parameter list.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. ImplGetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

GetAdapterItemRequestParams (see page 6148)

AddSpecialParams (see page 6144)

OnGetAdapterRequestParams (see page 6154)

63.1.8.2.12 TCustomAdapterDispatcher.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6141) method.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The HandleRequest (see page 6141) method calls ImplHandleRequest to process an http request message and obtain the response that should be sent back to the caller. (The HandleRequest (see page 6141) method is called using the IWebRequestHandler interface.)

ImplHandleRequest

1. Calls GetDispatchParams (see page 6148) to parse the request and identify the adapter that should handle it.
2. If it finds an adapter, calls DoBeforeDispatchAdapterRequest (see page 6146) to generate the OnBeforeDispatchAction (see page 6153) and OnBeforeDispatchImage (see page 6153) events.
3. If the event handlers do not handle the event, forwards the request to the adapter for processing.
4. Calls DoAfterDispatchAdapterRequest (see page 6146) to generate the OnAfterDispatchAction (see page 6152) and OnAfterDispatchImage (see page 6152) events.
5. Calls DoAdapterRequestNotHandled (see page 6145) to generate the OnActionRequestNotHandled (see page 6151) and OnImageRequestNotHandled (see page 6154) events if the request has not yet been handled.

See Also

IWebRequestHandler

HandleRequest (see page 6141)

GetDispatchParams (see page 6148)

DoBeforeDispatchAdapterRequest (see page 6146)

DoAfterDispatchAdapterRequest (see page 6146)

DoAdapterRequestNotHandled (see page 6145)

63.1.8.3 TCustomAdapterDispatcher Events

The events of the TCustomAdapterDispatcher class are listed here.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 6151)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6152)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6152)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6153)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6153)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6154)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6154)	Occurs when the WebSnap application could not provide a requested image.

63.1.8.3.1 TCustomAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write (see page 3789) an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates

an OnImageRequestNotHandled (see page 6154). If the OnImageRequestNotHandled (see page 6154) event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException (see page 6120) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 5977) function.

See Also

OnAfterDispatchAction (see page 6152)

OnBeforeDispatchAction (see page 6153)

EAdapterDispatcherException (see page 6120)

WebContext (see page 5977)

Response (see page 5949)

OnImageRequestNotHandled (see page 6154)

63.1.8.3.2 TCustomAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write (see page 3789) an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 6151)

OnBeforeDispatchAction (see page 6153)

63.1.8.3.3 TCustomAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write (see page 3789) an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 6154)

OnBeforeDispatchImage (see page 6153)

63.1.8.3.4 TCustomAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write (see page 3789) an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction (see page 6152) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 5977) function.

See Also

OnActionRequestNotHandled (see page 6151)

OnAfterDispatchAction (see page 6152)

WebContext (see page 5977)

Response (see page 5949)

63.1.8.3.5 TCustomAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write (see page 3789) an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage (see page 6152) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 5977) function.

See Also

OnImageRequestNotHandled (see page 6154)

OnAfterDispatchImage (see page 6152)

WebContext (see page 5977)

Response (see page 5949)

63.1.8.3.6 TCustomAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write (see page 3789) an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 6148)

EncodeAdapterItemRequestAsFieldValue (see page 6147)

EncodeAdapterItemRequestAsHREF (see page 6147)

AddSpecialParams (see page 6144)

63.1.8.3.7 TCustomAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write (see page 3789) an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException (see page 6120) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 5977) function.

See Also

OnAfterDispatchImage (see page 6152)

OnBeforeDispatchImage (see page 6153)

EAdapterDispatcherException (see page 6120)

WebContext (see page 5977)

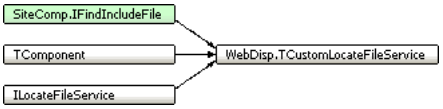
Response (see page 5949)

OnActionRequestNotHandled (see page 6151)

63.1.9 TCustomLocateFileService Class

TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.

Class Hierarchy



Pascal

```
TCustomLocateFileService = class(TComponent, ILocateFileService, IFindIncludeFile);
```

C++

```
class TCustomLocateFileService : public TComponent, public ILocateFileService, public
IFindIncludeFile;
```

File

WebDisp

Description

TCustomLocateFileService is the base class for components that control access to files in a WebSnap application. TCustomLocateFileService defines events that indicate where to locate a requested file or that supply a stream object from which to read the contents of a file. By using a TCustomLocateFileService descendant, a WebSnap application can adjust to differences in the way files are deployed or even to use different files, depending on the current Web context.

Do not create instances of TCustomLocateFileService. All of the events it introduces are protected, and so unavailable to your application. Instead, use a TCustomLocateFileService descendant, such as TLocateFileService (see page 6194), which publishes those events.

See Also

TLocateFileService (see page 6194)

TStream

63.1.9.1 TCustomLocateFileService Members

The following tables list the members exposed by TCustomLocateFileService.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 6160)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6161)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6161)	Occurs when the WebSnap application tries to locate the template file for a content producer.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6156)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6157)	Generates an OnFindStream (see page 6161) event.
	FindTemplateFile (see page 6157)	Generates an OnFindTemplateFile (see page 6161) event.
V	GetFileStream (see page 6158)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6158)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6159)	Provides the underlying implementation for the FindIncludeFile (see page 6156) method.
V	ImplGetTemplateStream (see page 6159)	Provides the underlying implementation for the GetTemplateStream (see page 6158) method.

63.1.9.2 TCustomLocateFileService Methods

The methods of the TCustomLocateFileService class are listed here.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6156)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6157)	Generates an OnFindStream (see page 6161) event.
	FindTemplateFile (see page 6157)	Generates an OnFindTemplateFile (see page 6161) event.
V	GetFileStream (see page 6158)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6158)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6159)	Provides the underlying implementation for the FindIncludeFile (see page 6156) method.
V	ImplGetTemplateStream (see page 6159)	Provides the underlying implementation for the GetTemplateStream (see page 6158) method.

63.1.9.2.1 TCustomLocateFileService.FindIncludeFile Method

Returns the fully-qualified path name for an include file for a server-side script.

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const UnicodeString AFileName,
UnicodeString AResult);
```

Description

WebSnap applications call FindIncludeFile when they encounter a reference to an include file in server-side script. This method returns the fully-qualified path name for the include file.

FindIncludeFile calls the virtual ImplFindIncludeFile (see page 6159) method to obtain the path and name of the include file. In TCustomLocateFileService (see page 6155), ImplFindIncludeFile (see page 6159) fetches the include file name by generating an OnFindIncludeFile (see page 6160) event. Component writers that which to locate include files in some other

way should override the ImplFindIncludeFile (see page 6159) method.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile returns true if it locates the requested file and returns its fully-qualified name as AResult.

See Also

OnFindIncludeFile (see page 6160)

ImplFindIncludeFile (see page 6159)

WebModuleContext

63.1.9.2.2 TCustomLocateFileService.FindStream Method

Generates an OnFindStream (see page 6161) event.

Pascal

```
function FindStream(AComponent: TComponent; const AFileName: string; out AResult: TStream; out AOwned: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindStream(TComponent * AComponent, const UnicodeString AFileName, TStream AResult, Boolean AOwned);
```

Description

TCustomLocateFileService (see page 6155) calls FindStream internally to generate an OnFindStream (see page 6161) event.

AComponent is the object that uses or is associated with the requested file. Typically, this is a content producer.

AFileName is the name of the file to locate. This can be a "logical" file that does not exist on disk.

AResult returns a stream from which the caller can read the contents of the requested file.

AOwned indicates whether the caller should free AResult after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

FindStream returns true if it supplies a stream as the value of the AResult parameter. It returns false if there is no OnFindStream (see page 6161) event handler or if the event handler does not return a value for AResult.

See Also

OnFindStream (see page 6161)

63.1.9.2.3 TCustomLocateFileService.FindTemplateFile Method

Generates an OnFindTemplateFile (see page 6161) event.

Pascal

```
function FindTemplateFile(AComponent: TComponent; const AFileName: string; var AResult: string): Boolean;
```

C++

```
__fastcall Boolean FindTemplateFile(TComponent * AComponent, const UnicodeString AFileName, UnicodeString AResult);
```

Description

TCustomLocateFileService (see page 6155) calls FindTemplateFile internally to generate an OnFindTemplateFile (see page 6161) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the name of the file to locate.

AResult returns the fully-qualified path name for the requested file.

FindTemplateFile returns true if it supplies a file name as the value of the AResult parameter. It returns false if there is no OnFindTemplateFile (see page 6161) event handler or if the event handler does not return a value for AResult.

See Also

OnFindTemplateFile (see page 6161)

63.1.9.2.4 TCustomLocateFileService.GetFileStream Method

Returns a stream object from which to read a specified file.

Pascal

```
function GetFileStream(const AFileName: string; out AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetFileStream(const UnicodeString AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 6158) method calls GetFileStream internally to obtain the stream it returns when the requested file maps to a fully-qualified path name.

AFileName is the fully-qualified name of a file whose contents can be read from the returned stream.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetFileStream returns a stream object from which to read the contents of AFileName.

See Also

GetTemplateStream (see page 6158)

63.1.9.2.5 TCustomLocateFileService.GetTemplateStream Method

Returns a stream from which to read a template file.

Pascal

```
function GetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned: Boolean):
TStream;
```

C++

```
__fastcall TStream GetTemplateStream(TComponent * AComponent, UnicodeString AFileName, Boolean
AOwned);
```

Description

Producer components call GetTemplateStream to read the contents of the template files they use to generate their content. GetTemplateStream locates the specified template file (which can be a logical rather than a physical file name), and returns a stream from which the caller can read the file contents.

GetTemplateStream calls the ImplGetTemplateStream (see page 6159) method, which generates first an OnFindStream (see page 6161) event, and, if OnFindStream (see page 6161) is unsuccessful, an OnFindTemplateFile (see page 6161) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL (C++).

See Also

ImplGetTemplateStream (see page 6159)

OnFindStream (see page 6161)

OnFindTemplateFile (see page 6161)

63.1.9.2.6 TCustomLocateFileService.ImplFindIncludeFile Method

Provides the underlying implementation for the FindIncludeFile (see page 6156) method.

Pascal

```
function ImplFindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplFindIncludeFile(TComponent * AComponent, const UnicodeString
AFileName, UnicodeString AResult);
```

Description

The FindIncludeFile (see page 6156) method calls ImplFindIncludeFile to locate the fully-qualified path name for an include file to a server-side script. ImplFindIncludeFile generates an OnFindIncludeFile (see page 6160) event to obtain the fully-qualified path name.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile (see page 6156) returns true if it locates the requested file and returns its fully-qualified name as AResult. It returns false if there is no OnFindIncludeFile (see page 6160) event handler or if the event handler does not return a value for AResult.

See Also

OnFindIncludeFile (see page 6160)

FindIncludeFile (see page 6156)

WebModuleContext

63.1.9.2.7 TCustomLocateFileService.ImplGetTemplateStream Method

Provides the underlying implementation for the GetTemplateStream (see page 6158) method.

Pascal

```
function ImplGetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned:
Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplGetTemplateStream(TComponent * AComponent, UnicodeString
AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 6158) method calls ImplGetTemplateStream to obtain a stream from which to read a specified template. ImplGetTemplateStream generates an OnFindStream (see page 6161) event, and, if OnFindStream (see page 6161) is unsuccessful, an OnFindTemplateFile (see page 6161) event. If it gets a file name from the OnFindTemplateFile (see page 6161) event handler, ImplGetTemplateStream calls the GetFileStream (see page 6158) method to obtain a stream object from which to read the file.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream (see page 6158) returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL (C++).

See Also

GetTemplateStream (see page 6158)

OnFindStream (see page 6161)

OnFindTemplateFile (see page 6161)

GetFileStream (see page 6158)

63.1.9.3 TCustomLocateFileService Events

The events of the TCustomLocateFileService class are listed here.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 6160)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6161)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6161)	Occurs when the WebSnap application tries to locate the template file for a content producer.

63.1.9.3.1 TCustomLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write (see page 3789) an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream (see page 6161) event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 6161)

OnFindTemplateFile (see page 6161)

63.1.9.3.2 TCustomLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write (see page 3789) an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile (see page 6161). One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 6160)

OnFindTemplateFile (see page 6161)

63.1.9.3.3 TCustomLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write (see page 3789) an OnFindIncludeFile (see page 6160) event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an OnFindTemplateFile or an OnFindStream (see page 6161) event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

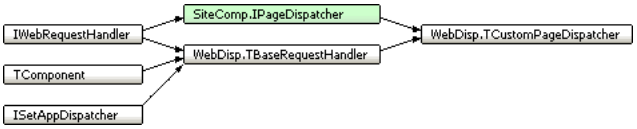
OnFindStream (see page 6161)

OnFindIncludeFile (see page 6160)

63.1.10 TCustomPageDispatcher Class

Dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy



Pascal

```
TCustomPageDispatcher = class(TBaseRequestHandler, IPageDispatcher);
```

C++

```
class TCustomPageDispatcher : public TBaseRequestHandler, public IPageDispatcher;
```

File

WebDisp

Description

TCustomPageDispatcher is the base class for components that locate the appropriate page module for handling an incoming HTTP message and retrieve the content that the identified page module creates for the corresponding response message. Typically, the WebSnap application wizard adds a TCustomPageDispatcher descendant to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to an URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 6169) property.

See Also

TPageDispatcher (see page 6210)

63.1.10.1 TCustomPageDispatcher Members

The following tables list the members exposed by TCustomPageDispatcher.

Protected Methods

	Name	Description
	DispatchPageName (see page 6164)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6164)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6165)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6165)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6165)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6166)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	HandleRequest (see page 6141)	Handles an HTTP request.
	ImplDispatchPageName (see page 6166)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 6166)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 6166)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

	ImplGetLoginRequired (see page 6167)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
	ImplPageNameOfRequest (see page 6167)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6167)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6168)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6168)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6168)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6168)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.
	ValidatePage (see page 6169)	This is ValidatePage, a member of class TCustomPageDispatcher.

Protected Properties

	Name	Description
	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6170)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 6171)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 6171)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 6172)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 6172)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6173)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6174)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6174)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Public Properties

	Name	Description
	DefaultPage (see page 6169)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
	DefaultPageName (see page 1005)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

63.1.10.2 TCustomPageDispatcher Methods

The methods of the TCustomPageDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchPageName (see page 6164)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6164)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6165)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6165)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6165)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6166)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	ImplDispatchPageName (see page 6166)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
	ImplGetCanView (see page 6166)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
	ImplGetDefaultPageName (see page 6166)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

	ImplGetLoginRequired (see page 6167)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
	ImplPageNameOfRequest (see page 6167)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6167)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6168)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6168)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6168)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6168)	Fills out a response message so that it redirects the caller to another page in the application.
	ValidatePage (see page 6169)	This is ValidatePage, a member of class TCustomPageDispatcher.

63.1.10.2.1 TCustomPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 1004)

HandleRequest

Content

WebContext (see page 5977)

63.1.10.2.2 TCustomPageDispatcher.EndUser Method

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

This is EndUser, a member of class TCustomPageDispatcher.

63.1.10.2.3 TCustomPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const UnicodeString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 1003)

HasViewRights (see page 5966)

OnCanViewPage (see page 6172)

63.1.10.2.4 TCustomPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall UnicodeString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName property.

See Also

DefaultPageName (see page 1005)

63.1.10.2.5 TCustomPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const UnicodeString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 1003)

IsLoginRequired

OnIsLoginRequired (see page 6173)

63.1.10.2.6 TCustomPageDispatcher.GetRequestedPage Method

Pascal

```
function GetRequestedPage(const APageName: string; AFlags: TDispatchPageFlags): TObject;
```

C++

```
__fastcall TObject * GetRequestedPage(const UnicodeString APageName, TDispatchPageFlags
AFlags);
```

Description

This is GetRequestedPage, a member of class TCustomPageDispatcher.

63.1.10.2.7 TCustomPageDispatcher.ImplDispatchPageName Method

Pascal

```
function ImplDispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplDispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

This is ImplDispatchPageName, a member of class TCustomPageDispatcher.

63.1.10.2.8 TCustomPageDispatcher.ImplGetCanView Method

Pascal

```
function ImplGetCanView(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetCanView(const UnicodeString APageName);
```

Description

This is ImplGetCanView, a member of class TCustomPageDispatcher.

63.1.10.2.9 TCustomPageDispatcher.ImplGetDefaultPageName Method

Pascal

```
function ImplGetDefaultPageName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDefaultPageName();
```

Description

This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

63.1.10.2.10 TCustomPageDispatcher.ImplGetLoginRequired Method

Pascal

```
function ImplGetLoginRequired(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoginRequired(const UnicodeString APageName);
```

Description

This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.

63.1.10.2.11 TCustomPageDispatcher.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The ImplHandleRequest function is called from HandleRequest to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 6141)

63.1.10.2.12 TCustomPageDispatcher.ImplPageNameOfRequest Method

Pascal

```
function ImplPageNameOfRequest(Request: TWebRequest): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplPageNameOfRequest(TWebRequest Request);
```

Description

This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.

63.1.10.2.13 TCustomPageDispatcher.ImplRedirectToPageName Method

Pascal

```
function ImplRedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplRedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.

63.1.10.2.14 TCustomPageDispatcher.IsLoggedIn Method

Pascal

```
function IsLoggedIn: Boolean;
```

C++

```
__fastcall Boolean IsLoggedIn();
```

Description

This is IsLoggedIn, a member of class TCustomPageDispatcher.

63.1.10.2.15 TCustomPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString PageNameOfRequest(TWebRequest Request);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 1005)

63.1.10.2.16 TCustomPageDispatcher.ProduceLoginPage Method

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

This is ProduceLoginPage, a member of class TCustomPageDispatcher.

63.1.10.2.17 TCustomPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 1002)

HandleRequest

Content

WebContext (see page 5977)

63.1.10.2.18 TCustomPageDispatcher.ValidatePage Method

Pascal

```
function ValidatePage(const APage: string; var AHandled: Boolean): Boolean;
```

C++

```
__fastcall Boolean ValidatePage(const UnicodeString APage, Boolean AHandled);
```

Description

This is ValidatePage, a member of class TCustomPageDispatcher.

63.1.10.3 TCustomPageDispatcher Properties

The properties of the TCustomPageDispatcher class are listed here.

Public Properties

	Name	Description
	DefaultPage (see page 6169)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

63.1.10.3.1 TCustomPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property UnicodeString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

63.1.10.4 TCustomPageDispatcher Events

The events of the TCustomPageDispatcher class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6170)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 6171)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 6171)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 6172)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 6172)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6173)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6174)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6174)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

63.1.10.4.1 TCustomPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write (see page 3789) an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied (see page 6174) event occurs instead. If the page could not be located, an OnPageNotFound (see page 6174) event occurs instead.

See Also

OnBeforeDispatchPage (see page 6171)

OnPageNotFound (see page 6174)

OnCanViewPage (see page 6172)

OnIsLoginRequired (see page 6173)

OnPageAccessDenied (see page 6174)

63.1.10.4.2 TCustomPageDispatcher.OnAfterRedirectToPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write (see page 3789) an OnAfterRedirectToPage event handler to take specific action after an HTTP request has been successfully redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost), and the new target Web page module succeeds in handling the request.

See Also

OnBeforeRedirectToPage (see page 6172)

OnAfterDispatchPage (see page 6170)

RedirectOptions (see page 5464)

63.1.10.4.3 TCustomPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write (see page 3789) an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext (see page 5977) function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 6170)

OnPageNotFound (see page 6174)

OnCanViewPage (see page 6172)

OnIsLoginRequired (see page 6173)

OnPageAccessDenied (see page 6174)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

63.1.10.4.4 TCustomPageDispatcher.OnBeforeRedirectToPage Event

Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write (see page 3789) an OnBeforeRedirectToPage event handler to take specific action before an HTTP request is redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the dispatcher assumes it has handled the request and performs no further processing. Otherwise, after the event handler exits, the page dispatcher calls on the Web Module to set up the response message and call SendRedirect.

Tip: To access the Web request and response objects from within an OnBeforeRedirectToPage event handler, call the global WebContext (see page 5977) function to access the Web context and read its Request and Response properties.

See Also

OnAfterRedirectToPage (see page 6171)

OnBeforeDispatchPage (see page 6171)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

SendRedirect

63.1.10.4.5 TCustomPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write (see page 3789) an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage (see page 6171) event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired (see page 6173) event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied (see page 6174) event, where the application can handle the login process.

b. If the OnPageAccessDenied (see page 6174) event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the dispatcher calls on a TEndUserAdapter component to check the user's access rights.

   Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext (see page 5977) function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 6173)

OnBeforeDispatchPage (see page 6171)

OnPageAccessDenied (see page 6174)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

ViewAccess

FindPageInfo (see page 5941)

63.1.10.4.6 TCustomPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write (see page 3789) an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage (see page 6171) event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied (see page 6174) event handler, or if the OnPageAccessDenied (see page 6174) event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 6172)

OnBeforeDispatchPage (see page 6171)

OnPageAccessDenied (see page 6174)

LoginPage (see page 5503)

63.1.10.4.7 TCustomPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write (see page 3789) an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext (see page 5977) function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 6173)

OnCanViewPage (see page 6172)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

LoginPage (see page 5503)

63.1.10.4.8 TCustomPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write (see page 3789) an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException (see page 6121) exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext (see page 5977) function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 6171)

OnAfterDispatchPage (see page 6170)

OnPageAccessDenied (see page 6174)

EPageDispatcherException (see page 6121)

WebContext (see page 5977)

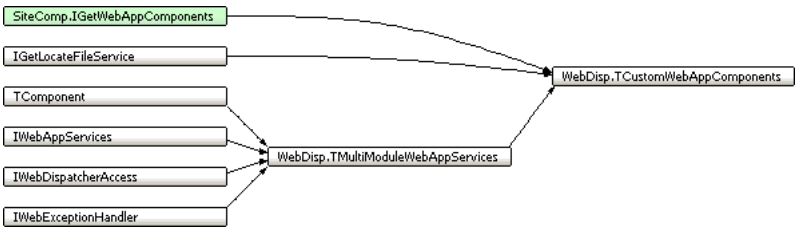
Request (see page 5948)

Response (see page 5949)

63.1.11 TCustomWebAppComponents Class

TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.

Class Hierarchy



Pascal

```
TCustomWebAppComponents = class(TMultiModuleWebAppServices, IGetWebAppComponents,
IGetLocateFileService);
```

C++

```
class TCustomWebAppComponents : public TMultiModuleWebAppServices, public
IGetWebAppComponents, public IGetLocateFileService;
```

File

WebDisp

Description

Use TCustomWebAppComponents as a base class for the component that centralizes access to the main WebSnap components. It keeps track of the current request and response objects, maintains a list of the available Web modules, and locates the appropriate components to handle incoming request messages. TCustomWebAppComponents introduces properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include:

PageDispatcher (see page 6188), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 6185), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 6188), which manages the sessions for each caller.

ApplicationAdapter (see page 6186), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 6187), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 6189), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 6187), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 6186), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fill in these properties for any components that you select.

See Also

TApplicationAdapter (see page 5427)

TEndUserAdapter (see page 5750)

TPageDispatcher (see page 6210)

TAdapterDispatcher (see page 6122)

TLocateFileService (see page 6194)

TSessionsService (see page 6722)

TWebUserList (see page 6759)

WebContext

63.1.11.1 TCustomWebAppComponents Members

The following tables list the members exposed by TCustomWebAppComponents.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6209)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6209)	Occurs before TMultiModuleWebAppServices (see page 6197) passes an HTTP request onto the other components in the application for processing.

Protected Methods

	Name	Description
	Access_Request (see page 6199)	Returns the value of the Access_Request property.
	Access_Response (see page 6199)	Returns the value of the Access_Response property.
V	CreateContext (see page 6178)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6200)	Generates an OnAfterDispatch (see page 6209) event.
	DoBeforeDispatch (see page 6200)	Generates an OnBeforeDispatch (see page 6209) event.
V	DoneModule (see page 6201)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6201)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 6179)	Returns the value of the AdapterDispatcher (see page 6185) property.
	GetExceptionHandler (see page 6202)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetLocateFileService (see page 6179)	Returns the value of the LocateFileService (see page 6187) property.
	GetPageDispatcher (see page 6179)	Returns the value of the PageDispatcher (see page 6188) property.
	GetWebApplicationInfo (see page 6180)	Returns the value of the ApplicationAdapter (see page 6186) property.
	GetWebEndUser (see page 6180)	Returns the value of the EndUserAdapter (see page 6187) property.
	GetWebUserList (see page 6180)	Returns the value of the UserListService (see page 6189) property.
	HandleException (see page 6202)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6203)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6181)	Provides the underlying implementation of the FinishContext (see page 6201) method.
V	ImplHandleException (see page 6181)	Provides the underlying implementation of the HandleException (see page 6202) method.
V	ImplHandleRequest (see page 6182)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
V	ImplInitContext (see page 6182)	Provides the underlying implementation for the InitContext (see page 6205) method.
	InitContext (see page 6205)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6183)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6184)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6184)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	ModuleAdded (see page 6207)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 6185)	Updates the TCustomWebAppComponents (see page 6175) properties when components are added to or removed from the application.

Protected Properties

	Name	Description
R	Request (see page 6208)	Provides access to the current request object.
R	Response (see page 6208)	Provides access to the current response object.
R	WebModules (see page 6208)	Lists the Web modules available in the WebSnap application.

Public Events

	Name	Description
	OnException (see page 6189)	Occurs when an exception escapes all try blocks in the WebSnap application.

Public Properties

	Name	Description
	AdapterDispatcher (see page 6185)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6186)	Provides access to the application adapter.
	DispatchActions (see page 6186)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6187)	Provides access to an end-user adapter.
	LocateFileService (see page 6187)	Provides access to the file location service.
	PageDispatcher (see page 6188)	Provides access to the current page dispatcher object.
	Sessions (see page 6188)	Provides access to the sessions service.

	UserListService (see page 6189)	Provides access to the file location service.
R	WebApplicationInfo (see page 981)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 981)	This is WebEndUser, a member of class IGetWebAppComponents.

63.1.11.2 TCustomWebAppComponents Methods

The methods of the TCustomWebAppComponents class are listed here.

Protected Methods

	Name	Description
V	CreateContext (see page 6178)	Creates the Web context for the current request message.
	GetAdapterDispatcher (see page 6179)	Returns the value of the AdapterDispatcher (see page 6185) property.
	GetLocateFileService (see page 6179)	Returns the value of the LocateFileService (see page 6187) property.
	GetPageDispatcher (see page 6179)	Returns the value of the PageDispatcher (see page 6188) property.
	GetWebApplicationInfo (see page 6180)	Returns the value of the ApplicationAdapter (see page 6186) property.
	GetWebEndUser (see page 6180)	Returns the value of the EndUserAdapter (see page 6187) property.
	GetWebUserList (see page 6180)	Returns the value of the UserListService (see page 6189) property.
V	ImplFinishContext (see page 6181)	Provides the underlying implementation of the FinishContext (see page 6201) method.
V	ImplHandleException (see page 6181)	Provides the underlying implementation of the HandleException (see page 6202) method.
V	ImplHandleRequest (see page 6182)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
V	ImplInitContext (see page 6182)	Provides the underlying implementation for the InitContext (see page 6205) method.
V	InitModule (see page 6183)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6184)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6184)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
V	Notification (see page 6185)	Updates the TCustomWebAppComponents (see page 6175) properties when components are added to or removed from the application.

63.1.11.2.1 TCustomWebAppComponents.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; override;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The ImplInitContext (see page 6182) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext (see page 5977) function.

In TCustomWebAppComponents (see page 6175), CreateContext creates a TWebContext (see page 5950) object, creating a new session object for the current request and passing the session object along with the values of the protected Request (see page 6208), Response (see page 6208), and WebModules (see page 6208) properties to the TWebContext (see page 5950) constructor.

See Also

WebContext (see page 5977)

ImplInitContext (see page 6182)

Create (see page 5952)

Request (see page 6208)

Response (see page 6208)

WebModules (see page 6208)

63.1.11.2.2 TCustomWebAppComponents.GetAdapterDispatcher Method

Returns the value of the AdapterDispatcher (see page 6185) property.

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

The protected GetAdapterDispatcher method is used internally to fetch the value of the AdapterDispatcher (see page 6185) property.

See Also

AdapterDispatcher (see page 6185)

63.1.11.2.3 TCustomWebAppComponents.GetLocateFileService Method

Returns the value of the LocateFileService (see page 6187) property.

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

The protected GetLocateFileService method is used internally to fetch the value of the LocateFileService (see page 6187) property.

See Also

LocateFileService (see page 6187)

63.1.11.2.4 TCustomWebAppComponents.GetPageDispatcher Method

Returns the value of the PageDispatcher (see page 6188) property.

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

The protected GetPageDispatcher method is used internally to fetch the value of the PageDispatcher (see page 6188) property.

See Also

PageDispatcher (see page 6188)

63.1.11.2.5 TCustomWebAppComponents.GetWebApplicationInfo Method

Returns the value of the ApplicationAdapter (see page 6186) property.

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

The protected GetWebApplicationInfo method is used internally to fetch the value of the ApplicationAdapter (see page 6186) property.

See Also

ApplicationAdapter (see page 6186)

63.1.11.2.6 TCustomWebAppComponents.GetWebEndUser Method

Returns the value of the EndUserAdapter (see page 6187) property.

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

The protected GetWebEndUser method is used internally to fetch the value of the EndUserAdapter (see page 6187) property.

See Also

EndUserAdapter (see page 6187)

63.1.11.2.7 TCustomWebAppComponents.GetWebUserList Method

Returns the value of the UserListService (see page 6189) property.

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

The protected GetWebUserList method is used internally to fetch the value of the UserListService (see page 6189) property.

See Also

UserListService (see page 6189)

63.1.11.2.8 TCustomWebAppComponents.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 6201) method.

Pascal

```
procedure ImplFinishContext; override;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

ImplFinishContext

1. Instructs the Web dispatcher that implements the DispatchActions (see page 6186) property to perform any final tasks.
2. Calls the DoneModule (see page 6201) method for every active Web module so that it can be freed or deactivated.
3. Frees the Web context for the current request.
4. Adds the session ID of the current session to the response message so that subsequent requests from the same caller can use the same session.

See Also

FinishContext (see page 6201)

DoneModule (see page 6201)

WebContext (see page 5977)

DispatchActions (see page 6186)

Sessions (see page 6188)

63.1.11.2.9 TCustomWebAppComponents.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 6202) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); override;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 6202) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code.

ImplHandleException generates an OnException (see page 6189) event, so that an event handler can take care of the exception. If the OnException (see page 6189) event handler does not handle the exception, ImplHandleException checks whether the response message has not yet been sent. If not, it adds the exception error message to the response.

E is the exception object for the exception that occurred.

Handled returns true if the OnException (see page 6189) event handler handled the exception or if ImplHandleException added the error message to the response object. If Handled returns false, the WebSnap application displays a dialog box that

indicates an error occurred.

See Also

HandleException (see page 6202)

OnException (see page 6189)

63.1.11.2.10 TCustomWebAppComponents.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6203) method.

Pascal

```
function ImplHandleRequest: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 6203) method calls ImplHandleRequest to handle the current HTTP request message.

ImpleHandleRequest

1.Generates an OnBeforeDispatch (see page 6209) event.

2.If the OnBeforeDispatch (see page 6209) event handler does not handle the request, checks the PageDispatcher (see page 6188), AdapterDispatcher (see page 6185), and DispatchActions (see page 6186) properties to find a dispatcher that can handle the request. If it finds one, the dispatcher forwards the request to the appropriate components for handling.

3.If the request was handled by the OnBeforeDispatch (see page 6209) event or by one of the dispatchers, and if the response was not already sent to the caller, generates an OnAfterDispatch (see page 6209) event.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 6203)

OnBeforeDispatch (see page 6209)

OnAfterDispatch (see page 6209)

PageDispatcher (see page 6188)

AdapterDispatcher (see page 6185)

DispatchActions (see page 6186)

63.1.11.2.11 TCustomWebAppComponents.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 6205) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 6205) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. If the DispatchActions (see page 6186) property is set, instructs the Web dispatcher to initialize itself.
2. Calls the CreateContext (see page 6178) method to create a Web context for the current request, supplying it with the values of the AWebModuleList, ARequest, and AResponse parameters. This Web context can then be accessed using the global WebContext (see page 5977) function.
3. Calls the InitModules (see page 6184) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message)
4. Hooks up the InitModule (see page 6183) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 6205)

CreateContext (see page 6178)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

WebModules (see page 5950)

InitModules (see page 6184)

DispatchActions (see page 6186)

InitModule (see page 6183)

63.1.11.2.12 TCustomWebAppComponents.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AModule: TComponent); override;
```

C++

```
virtual __fastcall InitModule(TComponent * AModule);
```

Description

InitModule is called whenever the WebSnap application instantiates a Web module to handle the current request. InitModule handles any initializations, including assigning itself as the Web dispatcher to any content producers in the Web module, generating an OnActivate event on the Web module, and allowing the Web dispatcher that is the value of the DispatchActions (see page 6186) property to perform its own initializations.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 6184)

Dispatcher

DispatchActions (see page 6186)

OnActivate (see page 6522)

63.1.11.2.13 TCustomWebAppComponents.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; override;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 6208) property lists, calling the InitModule (see page 6183) method to initialize each one. It then checks whether the session for the request is new (rather than a continuation of the session from a previous request), and if so, generates an OnStartSession event on the sessions service.

See Also

InitModule (see page 6183)

WebModules (see page 6208)

InitContext (see page 6205)

OnStartSession (see page 6721)

63.1.11.2.14 TCustomWebAppComponents.ListRequestHandlers Method

Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.

Pascal

```
procedure ListRequestHandlers(L: IInterfaceList); virtual;
```

C++

```
virtual __fastcall ListRequestHandlers(IInterfaceList L);
```

Description

The ImplHandleRequest (see page 6182) method calls ListRequestHandlers to obtain a list of all interfaces it can call to dispatch the request. ListRequestHandlers adds the interfaces that are the values of the PageDispatcher (see page 6188), AdapterDispatcher (see page 6185), and DispatchActions (see page 6186) properties to the list.

L is the interface list to which the dispatcher interfaces should be added.

See Also

ImplHandleRequest (see page 6182)

PageDispatcher (see page 6188)

AdapterDispatcher (see page 6185)

DispatchActions (see page 6186)

63.1.11.2.15 TCustomWebAppComponents.Notification Method

Updates the TCustomWebAppComponents (see page 6175) properties when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. It checks whether AComponent supports one of the interfaces that is the type of a TCustomWebAppComponents (see page 6175) property, and if so, sets that property to nil (Delphi) or NULL (C++) if the object is about to be removed, or sets the property to the AComponent interface if the object was just added.

See Also

Notification

AdapterDispatcher (see page 6185)

ApplicationAdapter (see page 6186)

DispatchActions (see page 6186)

EndUserAdapter (see page 6187)

LocateFileService (see page 6187)

Sessions (see page 6188)

PageDispatcher (see page 6188)

UserListService (see page 6189)

63.1.11.3 TCustomWebAppComponents Properties

The properties of the TCustomWebAppComponents class are listed here.

Public Properties

	Name	Description
	AdapterDispatcher (see page 6185)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6186)	Provides access to the application adapter.
	DispatchActions (see page 6186)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6187)	Provides access to an end-user adapter.
	LocateFileService (see page 6187)	Provides access to the file location service.
	PageDispatcher (see page 6188)	Provides access to the current page dispatcher object.
	Sessions (see page 6188)	Provides access to the sessions service.
	UserListService (see page 6189)	Provides access to the file location service.

63.1.11.3.1 TCustomWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher (see page 6122) object. If the WebSnap application does not include a TAdapterDispatcher (see page 6122) component, or some other component that implements the IAdapterDispatcher (see page 927) interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 6122)

DispatchActions (see page 6186)

63.1.11.3.2 TCustomWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo (see page 1019) interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 5427)

63.1.11.3.3 TCustomWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read (see page 3727) DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule

TWebDispatcher

63.1.11.3.4 TCustomWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser (see page 5962) interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

63.1.11.3.5 TCustomWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the

templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService (see page 6194) object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 6194)

63.1.11.3.6 TCustomWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher (see page 6210) object. If the WebSnap application does not include a TPageDispatcher (see page 6210) component, or some other component that implements the IPageDispatcher (see page 1001) interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 6210)

AdapterDispatcher (see page 6185)

63.1.11.3.7 TCustomWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that implements the ISessionsService (see page 1007) interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 6722)

63.1.11.3.8 TCustomWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList (see page 1057) interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 6759)

TPageDispatcher (see page 6210)

63.1.11.4 TCustomWebAppComponents Events

The events of the TCustomWebAppComponents class are listed here.

Public Events

	Name	Description
	OnException (see page 6189)	Occurs when an exception escapes all try blocks in the WebSnap application.

63.1.11.4.1 TCustomWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write (see page 3789) an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents (see page 6175) updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 6209)

63.1.12 TInvalidWebSession Class

Serves as a placeholder when an HTTP request could not be given a session.

Class Hierarchy

Pascal

```
TInvalidWebSession = class(TAbstractWebSession);
```

C++

```
class TInvalidWebSession : public TAbstractWebSession;
```

File

WebDisp

Description

TInvalidWebSession implements the Session property on the Web context when the WebSnap application fails to create an active session (TWebSession (see page 6225)) for the current request. The failure to create an active session can arise from various reasons, including the situation where the WebSnap application does not have a TSessionsService component or the situation where an incoming request message tries to reactivate a session that was terminated.

TInvalidWebSession acts as a placeholder, so that the WebSnap application does not need to raise an exception when the attempt to create an active session fails. Any attempt to use TInvalidWebSession, however, leads to an ESessionException (see page 6122) exception.

See Also

TSessionsService (see page 6722)

TAbstractWebContext (see page 5935)

TWebSession (see page 6225)

ESessionException (see page 6122)

63.1.12.1 TInvalidWebSession Members

The following tables list the members exposed by TInvalidWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 6191)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6192)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6192)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6193)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6193)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Create (see page 6191)	Creates and initializes an instance of TInvalidWebSession (see page 6190).
V	Terminate (see page 6193)	Raises an ESessionException (see page 6122) exception.
V	UpdateResponse (see page 6194)	Does nothing.

63.1.12.2 TInvalidWebSession.Create Constructor

Creates and initializes an instance of TInvalidWebSession (see page 6190).

Pascal

```
constructor Create(const AMsg: string);
```

C++

```
__fastcall TInvalidWebSession(const UnicodeString AMsg);
```

Description

Do not create instances of TInvalidWebSession (see page 6190). WebSnap applications automatically create TInvalidWebSession (see page 6190) objects when a call to the TWebSession (see page 6225) constructor fails.

AMsg is the error message from the exception that was raised within a call to the TWebSession (see page 6225) constructor. This error message is used for any ESessionException (see page 6122) exception that the TInvalidWebSession (see page 6190) object raises.

See Also

ESessionException (see page 6122)

Create (see page 6226)

63.1.12.3 TInvalidWebSession Methods

The methods of the TInvalidWebSession class are listed here.

Protected Methods

	Name	Description
V	GetSessionID (see page 6191)	Returns the value of the SessionID property.
V	GetTimeoutMinutes (see page 6192)	Returns the value of the TimeoutMinutes property.
V	GetValue (see page 6192)	Returns the value associated with a specified name.
V	SetTimeoutMinutes (see page 6193)	Sets the value of the TimoutMinutes property.
V	SetValue (see page 6193)	Assigns a value to a specified name.

Public Methods

	Name	Description
V	Terminate (see page 6193)	Raises an ESessionException (see page 6122) exception.
V	UpdateResponse (see page 6194)	Does nothing.

63.1.12.3.1 TInvalidWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID

63.1.12.3.2 TInvalidWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes

SetTimeoutMinutes

63.1.12.3.3 TInvalidWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const UnicodeString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values

SetValue

63.1.12.3.4 TInvalidWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes

TimeoutMinutes

63.1.12.3.5 TInvalidWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue

Values

63.1.12.3.6 TInvalidWebSession.Terminate Method

Raises an ESessionException (see page 6122) exception.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Do not call the Terminate method. TInvalidWebSession (see page 6190) does not represent an active Web session, so there is no Web session to terminate.

If the application calls the Terminate method, TInvalidWebSession (see page 6190) raises an ESessionException (see page 6122) object.

See Also

ESessionException (see page 6122)

63.1.12.3.7 TInvalidWebSession.UpdateResponse Method

Does nothing.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

UpdateResponse is called automatically to allow the WebSnap application to add the session ID to the response message before it is sent to the user. In TInvalidWebSession (see page 6190), UpdateResponse does nothing, because invalid Web sessions do not have session ID's.

See Also

SessionID

63.1.13 TLocateFileService Class

TLocateFileService controls the access of templates and server-side script include files at runtime.

Class Hierarchy

Pascal

```
TLocateFileService = class(TCustomLocateFileService);
```

C++

```
class TLocateFileService : public TCustomLocateFileService;
```

File

WebDisp

Description

Add a TLocateFileService component to your WebSnap application when you want greater control over how template and

server-side script include files are accessed at runtime. Define event handlers that will be called when the application requires one of these files. Event handlers can specify a different directory location for a file or return a TStream containing the file.

See Also

TStream

63.1.13.1 TLocateFileService Members

The following tables list the members exposed by TLocateFileService.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6156)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6157)	Generates an OnFindStream (see page 6161) event.
	FindTemplateFile (see page 6157)	Generates an OnFindTemplateFile (see page 6161) event.
V	GetFileStream (see page 6158)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6158)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6159)	Provides the underlying implementation for the FindIncludeFile (see page 6156) method.
V	ImplGetTemplateStream (see page 6159)	Provides the underlying implementation for the GetTemplateStream (see page 6158) method.

Public Methods

	Name	Description
	FindIncludeFile (see page 959)	This is FindIncludeFile, a member of class IFindIncludeFile.

Published Events

	Name	Description
	OnFindIncludeFile (see page 6195)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6196)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6196)	Occurs when the WebSnap application tries to locate the template file for a content producer.

63.1.13.2 TLocateFileService Events

The events of the TLocateFileService class are listed here.

Published Events

	Name	Description
	OnFindIncludeFile (see page 6195)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6196)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6196)	Occurs when the WebSnap application tries to locate the template file for a content producer.

63.1.13.2.1 TLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 6161)

OnFindTemplateFile (see page 6161)

63.1.13.2.2 TLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile. One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 6160)

OnFindTemplateFile (see page 6161)

63.1.13.2.3 TLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an OnFindTemplateFile or an OnFindStream event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

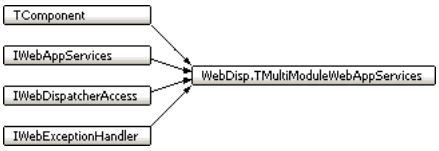
OnFindStream (see page 6161)

OnFindIncludeFile (see page 6160)

63.1.14 TMultiModuleWebAppServices Class

TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.

Class Hierarchy



Pascal

```
TMultiModuleWebAppServices = class(TComponent, IWebAppServices, IWebDispatcherAccess,
IWebExceptionHandler);
```

C++

```
class TMultiModuleWebAppServices : public TComponent, public IWebAppServices, public
IWebDispatcherAccess, public IWebExceptionHandler;
```

File

WebDisp

Description

TMultiModuleWebAppServices is the base class for components that provide access to the services available in a WebSnap application. It keeps track of the current request and response objects, maintains a list of the available Web modules, and introduces methods for locating the appropriate components to handle incoming request messages.

Do not use TMultiModuleWebAppServices directly. The methods that process request messages and any exceptions that arise during that processing are simple stubs that should be overridden by a descendant class. Rather, use a TMultiModuleWebAppServices descendant such as TWebAppComponents (see page 6217), to provide access to the central components of the WebSnap application.

See Also

TWebRequest

TWebResponse

TWebRequestHandler (see page 6600)

TWebAppComponents (see page 6217)

IWebAppServices

63.1.14.1 TMultiModuleWebAppServices Members

The following tables list the members exposed by TMultiModuleWebAppServices.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6209)	Occurs after the WebSnap application generates the content of a response message.

	OnBeforeDispatch (see page 6209)	Occurs before TMultiModuleWebAppServices (see page 6197) passes an HTTP request onto the other components in the application for processing.

Protected Methods

	Name	Description
	Access_Request (see page 6199)	Returns the value of the Access_Request property.
	Access_Response (see page 6199)	Returns the value of the Access_Response property.
V	CreateContext (see page 6200)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6200)	Generates an OnAfterDispatch (see page 6209) event.
	DoBeforeDispatch (see page 6200)	Generates an OnBeforeDispatch (see page 6209) event.
V	DoneModule (see page 6201)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6201)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 6202)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleException (see page 6202)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6203)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6203)	Provides the underlying implementation of the FinishContext (see page 6201) method.
V	ImplHandleException (see page 6204)	Provides the underlying implementation of the HandleException (see page 6202) method.
V	ImplHandleRequest (see page 6204)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
V	ImplInitContext (see page 6205)	Provides the underlying implementation for the InitContext (see page 6205) method.
	InitContext (see page 6205)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6206)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6207)	Initializes all the currently active Web modules.
	ModuleAdded (see page 6207)	Initializes a Web module when it is activated while processing the current request.

Protected Properties

	Name	Description
R	Request (see page 6208)	Provides access to the current request object.
R	Response (see page 6208)	Provides access to the current response object.
R	WebModules (see page 6208)	Lists the Web modules available in the WebSnap application.

63.1.14.2 TMultiModuleWebAppServices Methods

The methods of the TMultiModuleWebAppServices class are listed here.

Protected Methods

	Name	Description
	Access_Request (see page 6199)	Returns the value of the Access_Request property.
	Access_Response (see page 6199)	Returns the value of the Access_Response property.
V	CreateContext (see page 6200)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6200)	Generates an OnAfterDispatch (see page 6209) event.
	DoBeforeDispatch (see page 6200)	Generates an OnBeforeDispatch (see page 6209) event.
V	DoneModule (see page 6201)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6201)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 6202)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleException (see page 6202)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6203)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6203)	Provides the underlying implementation of the FinishContext (see page 6201) method.

	ImplHandleException (see page 6204)	Provides the underlying implementation of the HandleException (see page 6202) method.
	ImplHandleRequest (see page 6204)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
	ImplInitContext (see page 6205)	Provides the underlying implementation for the InitContext (see page 6205) method.
	InitContext (see page 6205)	Creates and initializes the Web context for the current Web request.
	InitModule (see page 6206)	Initializes a Web module before the WebSnap application uses it to handle the current request.
	InitModules (see page 6207)	Initializes all the currently active Web modules.
	ModuleAdded (see page 6207)	Initializes a Web module when it is activated while processing the current request.

63.1.14.2.1 TMultiModuleWebAppServices.Access_Request Method

Returns the value of the Access_Request property.

Pascal

```
function Access_Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Access_Request();
```

Description

Access_Access_Request provides the TMultiModuleWebAppServices (see page 6197) object's implementation of the Access_Request method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current request message.

See Also

Access_Request

Request

Dispatcher

63.1.14.2.2 TMultiModuleWebAppServices.Access_Response Method

Returns the value of the Access_Response property.

Pascal

```
function Access_Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Access_Response();
```

Description

Access_Access_Response provides the TMultiModuleWebAppServices (see page 6197) object's implementation of the Access_Response method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current Access_Response message.

See Also

Access_Response

Response

Dispatcher

63.1.14.2.3 TMultiModuleWebAppServices.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; virtual;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The InitContext (see page 6205) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext (see page 5977) function.

In TMultiModuleWebAppServices (see page 6197), CreateContext creates a TWebContext (see page 5950) object, passing the values of the Request (see page 6208), Response (see page 6208), and WebModules (see page 6208) properties to the constructor. The Web context is created without a session.

See Also

WebContext (see page 5977)

InitContext (see page 6205)

Create (see page 5952)

Request (see page 6208)

Response (see page 6208)

WebModules (see page 6208)

63.1.14.2.4 TMultiModuleWebAppServices.DoAfterDispatch Method

Generates an OnAfterDispatch (see page 6209) event.

Pascal

```
function DoAfterDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoAfterDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 6197) descendants call DoAfterDispatch to generate an OnAfterDispatch (see page 6209) event after dispatching a request message to the other components in the WebSnap application.

Request (see page 6208) is the current request message that was just handled.

Response (see page 6208) is the response message that was filled in by the rest of the WebSnap application.

DoAfterDispatch returns true if the WebSnap application should send the response message back to the caller.

See Also

OnAfterDispatch (see page 6209)

63.1.14.2.5 TMultiModuleWebAppServices.DoBeforeDispatch Method

Generates an OnBeforeDispatch (see page 6209) event.

Pascal

```
function DoBeforeDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoBeforeDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 6197) descendants call DoBeforeDispatch to generate an OnBeforeDispatch (see page 6209) event before dispatching a request message to the other components in the WebSnap application.

Request (see page 6208) is the current request message that needs to be handled.

Response (see page 6208) is the response message that needs to be filled out.

DoBeforeDispatch returns true if the OnBeforeDispatch (see page 6209) event handler has handled the event. When DoBeforeDispatch returns true, the WebSnap application does not need to continue processing the request.

See Also

OnBeforeDispatch (see page 6209)

63.1.14.2.6 TMultiModuleWebAppServices.DoneModule Method

Performs any final tasks before deactivating a Web module.

Pascal

```
procedure DoneModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall DoneModule(TComponent * AWebModule);
```

Description

The FinishContext (see page 6201) method calls DoneModule for every active Web module as part of the final cleanup before freeing the Web modules and Web context when the WebSnap application is finished handling an HTTP request. DoneModule notifies the Web module that it is about to be deactivated, so that it can perform any final tasks before it is freed or deactivated.

AWebModule is the Web module instance that is about to be deactivated.

See Also

WebModules (see page 6208)

InitModule (see page 6206)

FinishContext (see page 6201)

63.1.14.2.7 TMultiModuleWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller.

FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message.

FinishContext is not a virtual method. Descendant classes can't override this method to add additional final tasks. Instead, FinishContext calls the virtual ImplFinishContext (see page 6203) method, which descendant classes override to add their own final tasks.

See Also

InitContext (see page 6205)

ImplFinishContext (see page 6203)

63.1.14.2.8 TMultiModuleWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler returns the object that the WebSnap application should call when an exception is raised while trying to process a request message. The request handler must implement the IWebExceptionHandler interface, which includes only a single method: HandleException (see page 6202).

In TMultiModuleWebAppServices (see page 6197), GetExceptionHandler returns self this, because TMultiModuleWebAppServices (see page 6197) implements the IWebExceptionHandler interface.

See Also

HandleException (see page 6202)

63.1.14.2.9 TMultiModuleWebAppServices.HandleException Method

Responds to any exceptions not caught by the WebSnap application.

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

HandleException provides the last opportunity for a WebSnap application to handle an exception that passes through all the try blocks in the application code. If the HandleException method does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

HandleException calls the ImplHandleException (see page 6204) method to handle the exception. In TMultiModuleWebAppServices (see page 6197), ImplHandleException (see page 6204) does nothing. However, descendant classes override ImplHandleException (see page 6204) to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if HandleException handled the exception. If Handled returns false, the WebSnap application displays a

dialog box that indicates an error occurred.

See Also

ImplHandleException (see page 6204)

63.1.14.2.10 TMultiModuleWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext (see page 6205) method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext (see page 6201) is called to perform any final cleanup.

HandleRequest simply calls the ImplHandleRequest (see page 6204) method, which directs the processing of the request. In TMultiModuleWebAppServices (see page 6197), ImplHandleRequest (see page 6204) simply declares the request as handled while doing no processing. Descendant classes override ImplHandleRequest (see page 6204) to locate the components that should handle the request and forwarding the request to those components for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

ImplHandleRequest (see page 6204)

Request (see page 6208)

InitContext (see page 6205)

FinishContext (see page 6201)

63.1.14.2.11 TMultiModuleWebAppServices.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 6201) method.

Pascal

```
procedure ImplFinishContext; virtual;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

In TMultiModuleWebAppServices (see page 6197), ImplFinishContext calls the DoneModule (see page 6201) method for

every active Web module so that it can be freed or deactivated, and then frees the Web context for the current request.

Descendant classes override ImplFinishContext to add their own final tasks.

See Also

FinishContext (see page 6201)

DoneModule (see page 6201)

WebContext (see page 5977)

63.1.14.2.12 TMultiModuleWebAppServices.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 6202) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); virtual;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 6202) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code. If ImplHandleException does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

In TMultiModuleWebAppServices (see page 6197), ImplHandleException returns true, ignoring the exception that occurred. Descendant classes override this method to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if ImplHandleException handled the exception. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

HandleException (see page 6202)

63.1.14.2.13 TMultiModuleWebAppServices.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6203) method.

Pascal

```
function ImplHandleRequest: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 6203) method calls ImplHandleRequest to handle the current HTTP request message. In TMultiModuleWebAppServices (see page 6197), ImplHandleRequest returns true, indicating that the request is handled, but does nothing to fill out the response object. Descendant classes override this method to locate components in the WebSnap application that can process the request and forward the request to them for handling.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 6203)

Request (see page 6208)

Response (see page 6208)

63.1.14.2.14 TMultiModuleWebAppServices.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 6205) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); virtual;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 6205) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. Sets the WebModules (see page 6208), Request (see page 6208), and Response (see page 6208) properties to the values of the AWebModuleList, ARequest, and AResponse parameters, respectively.
2. Calls the CreateContext (see page 6200) method to create a Web context for the current request. This Web context can then be accessed using the global WebContext (see page 5977) function.
3. Calls the InitModules (see page 6207) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message.)
4. Hooks up the ModuleAdded (see page 6207) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 6205)

CreateContext (see page 6200)

WebContext (see page 5977)

Request (see page 6208)

Response (see page 6208)

WebModules (see page 6208)

InitModules (see page 6207)

63.1.14.2.15 TMultiModuleWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
```

```
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse
Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext (see page 5977) function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message. This becomes the value of the WebModules (see page 6208) property.

Request (see page 6208) describes the new request message. This becomes the value of the Request (see page 6208) property.

Response (see page 6208) is the response object that the WebSnap application should fill out when handling the current request. This becomes the value of the Response (see page 6208) property.

InitContext is not a virtual method. Descendant classes can't override this method to add additional initializations. Instead, InitContext calls the virtual ImplInitContext (see page 6205) method, which descendant classes override to add their own initial tasks.

See Also

ImplInitContext (see page 6205)

WebContext (see page 5977)

Request (see page 6208)

Response (see page 6208)

WebModules (see page 6208)

63.1.14.2.16 TMultiModuleWebAppServices.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall InitModule(TComponent * AWebModule);
```

Description

Both the ModuleAdded (see page 6207) and the InitModules (see page 6207) methods call InitModule to initialize a Web module after it is first created. InitModule assigns itself as the Web dispatcher to any content producers in the Web module, and notifies the Web module that it has been activated.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 6207)

ModuleAdded (see page 6207)

WebModules (see page 6208)

Dispatcher

63.1.14.2.17 TMultiModuleWebAppServices.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; virtual;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 6208) property lists, calling the InitModule (see page 6206) method to initialize each one.

See Also

InitModule (see page 6206)

WebModules (see page 6208)

InitContext (see page 6205)

63.1.14.2.18 TMultiModuleWebAppServices.ModuleAdded Method

Initializes a Web module when it is activated while processing the current request.

Pascal

```
procedure ModuleAdded(AWebModule: TComponent);
```

C++

```
__fastcall ModuleAdded(TComponent * AWebModule);
```

Description

TMultiModuleWebAppServices (see page 6197) assigns ModuleAdded as the OnModuleAdded event handler on the TWebModuleList that is the value of the WebModules (see page 6208) property. As a result, this method is called every time the WebSnap application activates a new Web module while processing the current request.

ModuleAdded calls the InitModule (see page 6206) method to initialize the new Web module before it is used.

AWebModule is the newly activated Web module.

See Also

InitModule (see page 6206)

OnModuleAdded

WebModules (see page 6208)

63.1.14.3 TMultiModuleWebAppServices Properties

The properties of the TMultiModuleWebAppServices class are listed here.

Protected Properties

	Name	Description
	Request (see page 6208)	Provides access to the current request object.
	Response (see page 6208)	Provides access to the current response object.
	WebModules (see page 6208)	Lists the Web modules available in the WebSnap application.

63.1.14.3.1 TMultiModuleWebAppServices.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3727) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6208)

63.1.14.3.2 TMultiModuleWebAppServices.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3727) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6208) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6208)

63.1.14.3.3 TMultiModuleWebAppServices.WebModules Property

Lists the Web modules available in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the

currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

When the application first receives a Web request message, the Web request handler initializes WebModules to include all Web modules that have the crAlways flag set in their factory's CreateMode property.

See Also

CreateMode

63.1.14.4 TMultiModuleWebAppServices Events

The events of the TMultiModuleWebAppServices class are listed here.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6209)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6209)	Occurs before TMultiModuleWebAppServices (see page 6197) passes an HTTP request onto the other components in the application for processing.

63.1.14.4.1 TMultiModuleWebAppServices.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write (see page 3789) an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices (see page 6197) introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 6209)

63.1.14.4.2 TMultiModuleWebAppServices.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices (see page 6197) passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write (see page 3789) an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices (see page 6197) introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

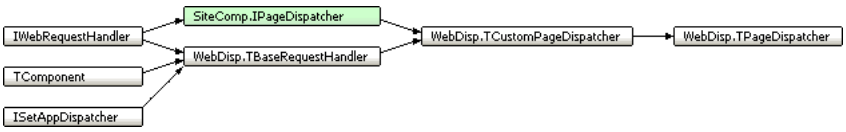
See Also

OnAfterDispatch (see page 6209)

63.1.15 TPageDispatcher Class

TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy



Pascal

```
TPageDispatcher = class(TCustomPageDispatcher);
```

C++

```
class TPageDispatcher : public TCustomPageDispatcher;
```

File

WebDisp

Description

Add a single TPageDispatcher object to the Web application module of a WebSnap application. Typically, the WebSnap application wizard adds a page dispatcher to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The page dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to a URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 6212) property.

See Also

TWebDispatcher

63.1.15.1 TPageDispatcher Members

The following tables list the members exposed by TPageDispatcher.

Protected Methods

	Name	Description
	DispatchPageName (see page 6164)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6164)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6165)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6165)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6165)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6166)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	HandleRequest (see page 6141)	Handles an HTTP request.
	ImplDispatchPageName (see page 6166)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 6166)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 6166)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.
V	ImplGetLoginRequired (see page 6167)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
V	ImplHandleRequest (see page 6141)	Handles application-specific processing of an HTTP request.
V	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
V	ImplPageNameOfRequest (see page 6167)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6167)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6168)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6168)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6168)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6168)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAppDispatcher (see page 6142)	Sets the AppDispatcher (see page 6142) property.
	ValidatePage (see page 6169)	This is ValidatePage, a member of class TCustomPageDispatcher.

Protected Properties

	Name	Description
R	AppDispatcher (see page 6142)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6171)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeRedirectToPage (see page 6172)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Public Methods

	Name	Description
	DispatchPageName (see page 1002)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 1003)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1003)	Returns the value of the DefaultPageName (see page 1005) property.
	GetLoginRequired (see page 1003)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1004)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1004)	Fills out a response message so that it redirects the caller to another page in the application.

Public Properties

	Name	Description
R	DefaultPageName (see page 1005)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 6213)	Occurs after the page dispatcher successfully dispatches an HTTP request message.

	OnBeforeDispatchPage (see page 6213)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnCanViewPage (see page 6214)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6215)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6215)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6216)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Published Properties

	Name	Description
	DefaultPage (see page 6212)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

63.1.15.2 TPageDispatcher Properties

The properties of the TPageDispatcher class are listed here.

Published Properties

	Name	Description
	DefaultPage (see page 6212)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

63.1.15.2.1 TPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property UnicodeString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

63.1.15.3 TPageDispatcher Events

The events of the TPageDispatcher class are listed here.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 6213)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnBeforeDispatchPage (see page 6213)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

	OnCanViewPage (see page 6214)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6215)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6215)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6216)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

63.1.15.3.1 TPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied event occurs instead. If the page could not be located, an OnPageNotFound event occurs instead.

See Also

OnBeforeDispatchPage (see page 6171)

OnPageNotFound (see page 6174)

OnCanViewPage (see page 6172)

OnIsLoginRequired (see page 6173)

OnPageAccessDenied (see page 6174)

63.1.15.3.2 TPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave

the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 6170)

OnPageNotFound (see page 6174)

OnCanViewPage (see page 6172)

OnIsLoginRequired (see page 6173)

OnPageAccessDenied (see page 6174)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

63.1.15.3.3 TPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied event, where the application can handle the login process.

b. If the OnPageAccessDenied event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the dispatcher calls on a TEndUserAdapter component to check the user's access rights.

   Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 6173)

OnBeforeDispatchPage (see page 6171)

OnPageAccessDenied (see page 6174)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

ViewAccess

FindPageInfo (see page 5941)

63.1.15.3.4 TPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied event handler, or if the OnPageAccessDenied event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 6172)

OnBeforeDispatchPage (see page 6171)

OnPageAccessDenied (see page 6174)

LoginPage (see page 5503)

63.1.15.3.5 TPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 6173)

OnCanViewPage (see page 6172)

WebContext (see page 5977)

Request (see page 5948)

Response (see page 5949)

LoginPage (see page 5503)

63.1.15.3.6 TPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 6171)

OnAfterDispatchPage (see page 6170)

OnPageAccessDenied (see page 6174)

EPageDispatcherException (see page 6121)

WebContext (see page 5977)

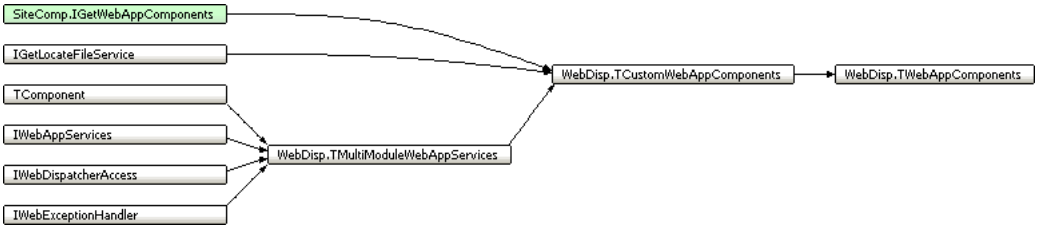
Request (see page 5948)

Response (see page 5949)

63.1.16 TWebAppComponents Class

TWebAppComponents holds the application components that provide the WebSnap application's functionality.

Class Hierarchy



Pascal

```
TWebAppComponents = class(TCustomWebAppComponents);
```

C++

```
class TWebAppComponents : public TCustomWebAppComponents;
```

File

WebDisp

Description

Use TWebAppComponents to centralize access to the main WebSnap components. It initializes the Web context for new request messages and locates the appropriate components to handle the request message. TWebAppComponents publishes properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include

PageDispatcher (see page 6222), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 6219), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 6222), which manages the sessions for each caller.

ApplicationAdapter (see page 6220), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 6221), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 6223), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 6221), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 6220), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fills in these properties for any components that you select.

Add TWebAppComponents to the application module of the WebSnap application to provide centralized access to the other components that govern the behavior of the WebSnap application.

63.1.16.1 TWebAppComponents Members

The following tables list the members exposed by TWebAppComponents.

Protected Methods

	Name	Description
	Access_Request (see page 6199)	Returns the value of the Access_Request property.
	Access_Response (see page 6199)	Returns the value of the Access_Response property.
V	CreateContext (see page 6178)	Creates the Web context for the current request message.
V	CreateContext (see page 6200)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6200)	Generates an OnAfterDispatch (see page 6209) event.
	DoBeforeDispatch (see page 6200)	Generates an OnBeforeDispatch (see page 6209) event.
V	DoneModule (see page 6201)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6201)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 6179)	Returns the value of the AdapterDispatcher (see page 6185) property.
	GetExceptionHandler (see page 6202)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetLocateFileService (see page 6179)	Returns the value of the LocateFileService (see page 6187) property.
	GetPageDispatcher (see page 6179)	Returns the value of the PageDispatcher (see page 6188) property.
	GetWebApplicationInfo (see page 6180)	Returns the value of the ApplicationAdapter (see page 6186) property.
	GetWebEndUser (see page 6180)	Returns the value of the EndUserAdapter (see page 6187) property.
	GetWebUserList (see page 6180)	Returns the value of the UserListService (see page 6189) property.
	HandleException (see page 6202)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6203)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6181)	Provides the underlying implementation of the FinishContext (see page 6201) method.
V	ImplFinishContext (see page 6203)	Provides the underlying implementation of the FinishContext (see page 6201) method.
V	ImplHandleException (see page 6181)	Provides the underlying implementation of the HandleException (see page 6202) method.
V	ImplHandleException (see page 6204)	Provides the underlying implementation of the HandleException (see page 6202) method.
V	ImplHandleRequest (see page 6182)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
V	ImplHandleRequest (see page 6204)	Provides the underlying implementation for the HandleRequest (see page 6203) method.
V	ImplInitContext (see page 6182)	Provides the underlying implementation for the InitContext (see page 6205) method.
V	ImplInitContext (see page 6205)	Provides the underlying implementation for the InitContext (see page 6205) method.
	InitContext (see page 6205)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6183)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModule (see page 6206)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6184)	Initializes all the currently active Web modules.
V	InitModules (see page 6207)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6184)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	ModuleAdded (see page 6207)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 6185)	Updates the TCustomWebAppComponents (see page 6175) properties when components are added to or removed from the application.

Protected Properties

	Name	Description
R	Request (see page 6208)	Provides access to the current request object.
R	Response (see page 6208)	Provides access to the current response object.

	WebModules (see page 6208)	Lists the Web modules available in the WebSnap application.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 980)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 980)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 980)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 980)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 980)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 981)	This is GetWebUserList, a member of class IGetWebAppComponents.

Public Properties

	Name	Description
	WebApplicationInfo (see page 981)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
	WebEndUser (see page 981)	This is WebEndUser, a member of class IGetWebAppComponents.

Published Events

	Name	Description
	OnAfterDispatch (see page 6223)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6224)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 6224)	Occurs when an exception escapes all try blocks in the WebSnap application.

Published Properties

	Name	Description
	AdapterDispatcher (see page 6219)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6220)	Provides access to the application adapter.
	DispatchActions (see page 6220)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6221)	Provides access to an end-user adapter.
	LocateFileService (see page 6221)	Provides access to the file location service.
	PageDispatcher (see page 6222)	Provides access to the current page dispatcher object.
	Sessions (see page 6222)	Provides access to the sessions service.
	UserListService (see page 6223)	Provides access to the file location service.

63.1.16.2 TWebAppComponents Properties

The properties of the TWebAppComponents class are listed here.

Published Properties

	Name	Description
	AdapterDispatcher (see page 6219)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6220)	Provides access to the application adapter.
	DispatchActions (see page 6220)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6221)	Provides access to an end-user adapter.
	LocateFileService (see page 6221)	Provides access to the file location service.
	PageDispatcher (see page 6222)	Provides access to the current page dispatcher object.
	Sessions (see page 6222)	Provides access to the sessions service.
	UserListService (see page 6223)	Provides access to the file location service.

63.1.16.2.1 TWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher object. If the WebSnap application does not include a TAdapterDispatcher component, or some other component that implements the IAdapterDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 6122)

DispatchActions (see page 6186)

63.1.16.2.2 TWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 5427)

63.1.16.2.3 TWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule

TWebDispatcher

63.1.16.2.4 TWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 5750)

TEndUserSessionAdapter (see page 5779)

63.1.16.2.5 TWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the

templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 6194)

63.1.16.2.6 TWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher object. If the WebSnap application does not include a TPageDispatcher component, or some other component that implements the IPageDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 6210)

AdapterDispatcher (see page 6185)

63.1.16.2.7 TWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that implements the ISessionsService interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 6722)

63.1.16.2.8 TWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 6759)

TPageDispatcher (see page 6210)

63.1.16.3 TWebAppComponents Events

The events of the TWebAppComponents class are listed here.

Published Events

	Name	Description
	OnAfterDispatch (see page 6223)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6224)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 6224)	Occurs when an exception escapes all try blocks in the WebSnap application.

63.1.16.3.1 TWebAppComponents.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back

to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 6209)

63.1.16.3.2 TWebAppComponents.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

See Also

OnAfterDispatch (see page 6209)

63.1.16.3.3 TWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 6209)

63.1.17 TWebSession Class

TWebSession stores temporary information associated with the user in a WebSnap application.

Class Hierarchy

Pascal

```
TWebSession = class(TAbstractWebSession, IWebSessionAccess);
```

C++

```
class TWebSession : public TAbstractWebSession, public IWebSessionAccess;
```

File

WebDisp

Description

TWebSession represents an active Web session. Use its Values property to get or set application-specific information about the current user. If your application uses sessions to store user information, you need to include a TSessionsService component in the Web application module.

A Web session stores information that persists over multiple calls from the same user. This information is stored as a set of Variant (see page 3430) values that are indexed by name. You can use these named Variants (see page 4997) to store application-defined information that is associated with a single user.

There is no need to instantiate session objects. A Session is created automatically when the WebSnap application receives an HTTP request message. The new session is given a unique identifier, which is added (in a cookie) to response messages, so that the same session can be used for subsequent requests from the same caller. The session persists until either the user logs out, a specified time elapses with no request messages from the caller, or the application explicitly calls the session's Terminate (see page 6229) method.

See Also

TSessionsService (see page 6722)

TAbstractWebContext (see page 5935)

TInvalidWebSession (see page 6190)

63.1.17.1 TWebSession Members

The following tables list the members exposed by TWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 6227)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6227)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6228)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6228)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6228)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Create (see page 6226)	This is the overview for the Create constructor overload.
	GetSessionsService (see page 6232)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6232)	This is GetWebSession, a member of class IWebSessionAccess.
V	Terminate (see page 6229)	Terminates the session.
V	UpdateResponse (see page 6229)	Adds the session ID to a cookie in a response message.

63.1.17.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWebSession.Create (ISessionsService, TSessionID) (see page 6226)	This is Create, a member of class TWebSession.
	TWebSession.Create (ISessionsService, TWebRequest, TWebResponse) (see page 6226)	This is Create, a member of class TWebSession.

63.1.17.2.1 TWebSession.Create Constructor (ISessionsService, TSessionID)

Pascal

```
constructor Create(ASessions: ISessionsService; AID: TSessionID); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TSessionID AID);
```

Description

This is Create, a member of class TWebSession.

63.1.17.2.2 TWebSession.Create Constructor (ISessionsService, TWebRequest, TWebResponse)

Pascal

```
constructor Create(ASessions: ISessionsService; ARequest: TWebRequest; AResponse:
TWebResponse); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TWebRequest ARequest, TWebResponse
AResponse);
```

Description

This is Create, a member of class TWebSession.

63.1.17.3 TWebSession Methods

The methods of the TWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 6227)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6227)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6228)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6228)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6228)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 6229)	Terminates the session.
	UpdateResponse (see page 6229)	Adds the session ID to a cookie in a response message.

63.1.17.3.1 TWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID

63.1.17.3.2 TWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes

SetTimeoutMinutes

63.1.17.3.3 TWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const UnicodeString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values

SetValue

63.1.17.3.4 TWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes

TimeoutMinutes

63.1.17.3.5 TWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue

Values

63.1.17.3.6 TWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Call Terminate to explicitly terminate a session. Once a session has been terminated, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimoutMinutes minutes.

See Also

TimeoutMinutes

Values

63.1.17.3.7 TWebSession.UpdateResponse Method

Adds the session ID to a cookie in a response message.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

Call UpdateResponse to add the value of the SessionID property to one of the response message's cookies. This allows the user's next request message to use the Values stored in this session.

AResponse is the Web response message to which a cookie with the session ID should be added.

See Also

SessionID

Values

63.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebSessionAccess (see page 6231)	This is class WebDisp.IWebSessionAccess.

63.2.1 IWebSessionAccess Interface

Class Hierarchy

WebDisp.IWebSessionAccess

Pascal

```
IWebSessionAccess = interface;
```

C++

```
__interface IWebSessionAccess;
```

File

WebDisp

Description

This is class WebDisp.IWebSessionAccess.

63.2.1.1 IWebSessionAccess Members

The following tables list the members exposed by IWebSessionAccess.

Public Methods

	Name	Description
	GetSessionsService (see page 6232)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6232)	This is GetWebSession, a member of class IWebSessionAccess.

63.2.1.2 IWebSessionAccess Methods

The methods of the IWebSessionAccess class are listed here.

Public Methods

	Name	Description
	GetSessionsService (see page 6232)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6232)	This is GetWebSession, a member of class IWebSessionAccess.

63.2.1.2.1 IWebSessionAccess.GetSessionsService Method

Pascal

```
function GetSessionsService: ISessionsService;
```

C++

```
__fastcall ISessionsService GetSessionsService();
```

Description

This is GetSessionsService, a member of class IWebSessionAccess.

63.2.1.2.2 IWebSessionAccess.GetWebSession Method

Pascal

```
function GetWebSession: IWebSession;
```

C++

```
__fastcall IWebSession GetWebSession();
```

Description

This is GetWebSession, a member of class IWebSessionAccess.

63.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 6233)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 6233)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 6234)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 6234)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 6234)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 6235)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 6235)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 6235)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 6236)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 6236)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 6236)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 6237)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 6237)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 6237)	Redirects a browser to another specified page.

63.3.1 WebDisp.AdapterRequestParamsAsStrings Function

Pascal

```
function AdapterRequestParamsAsStrings(AAdapterRequest: IUnknown): TStrings;
```

C++

```
TStrings AdapterRequestParamsAsStrings(IUnknown AAdapterRequest);
```

File

WebDisp

Description

This is function WebDisp.AdapterRequestParamsAsStrings.

63.3.2 WebDisp.DispatchPageName Function

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.DispatchPageName.

63.3.3 WebDisp.EncodeParamNameValues Function

Pascal

```
function EncodeParamNameValues(ParamValues: TStrings): string;
```

C++

```
UnicodeString EncodeParamNameValues(TStrings ParamValues);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamNameValues.

63.3.4 WebDisp.EncodeParamsAsHTTPQuery Function

Pascal

```
function EncodeParamsAsHTTPQuery(AParams: TStrings): string;
```

C++

```
UnicodeString EncodeParamsAsHTTPQuery(TStrings AParams);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamsAsHTTPQuery.

63.3.5 WebDisp.ExpandScriptException Function

Pascal

```
function ExpandScriptException(E: EScriptError): string;
```

C++

```
UnicodeString ExpandScriptException(EScriptError E);
```

File

WebDisp

Description

This is function WebDisp.ExpandScriptException.

63.3.6 WebDisp.ExtractParamNameValues Function

Pascal

```
procedure ExtractParamNameValues(const AParam: string; AList: TStrings);
```

C++

```
ExtractParamNameValues(const UnicodeString AParam, TStrings AList);
```

File

WebDisp

Description

This is function WebDisp.ExtractParamNameValues.

63.3.7 WebDisp.FindDefaultAppPageName Function

Pascal

```
function FindDefaultAppPageName: string;
```

C++

```
UnicodeString FindDefaultAppPageName();
```

File

WebDisp

Description

This is function WebDisp.FindDefaultAppPageName.

63.3.8 WebDisp.PageNameOfRequest Function

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
UnicodeString PageNameOfRequest(TWebRequest ARequest);
```

File

WebDisp

Description

This is function WebDisp.PageNameOfRequest.

63.3.9 WebDisp.RaiseDispatchBlankPageNameException Function

Pascal

```
procedure RaiseDispatchBlankPageNameException;
```

C++

```
RaiseDispatchBlankPageNameException();
```

File

WebDisp

Description

This is function WebDisp.RaiseDispatchBlankPageNameException.

63.3.10 WebDisp.RaisePageDoesNotProvideContent Function

Pascal

```
procedure RaisePageDoesNotProvideContent(const APageName: string);
```

C++

```
RaisePageDoesNotProvideContent(const UnicodeString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageDoesNotProvideContent.

63.3.11 WebDisp.RaisePageNotFound Function

Pascal

```
procedure RaisePageNotFound(const APageName: string);
```

C++

```
RaisePageNotFound(const UnicodeString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageNotFound.

63.3.12 WebDisp.RedirectToPageName Function

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams, TWebResponse
AResponse, TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.RedirectToPageName.

63.3.13 WebDisp.RegisterAdapterRequestIdentifier Function

Pascal

```
procedure RegisterAdapterRequestIdentifier(const AIdentifier: string);
```

C++

```
RegisterAdapterRequestIdentifier(const UnicodeString AIdentifier);
```

File

WebDisp

Description

This is function WebDisp.RegisterAdapterRequestIdentifier.

63.3.14 WebDisp.SafeRedirect Function

Redirects a browser to another specified page.

Pascal

```
procedure SafeRedirect(AResponse: TWebResponse; const AURL: string);
```

C++

```
SafeRedirect(TWebResponse AResponse, const UnicodeString AURL);
```

File

WebDisp

Description

Use the SafeRedirect function to send a response which redirects a Web browser to another page. AResponse is the response object which will send the redirect. AURL is the URL for the page that the browser is being redirected to.

SafeRedirect differs from other redirect methods (such as an override of TWebResponse (see page 6245)'s SendRedirect method) in that it allows browsers to accept cookies. Other redirect methods don't allow cookies to be set during a redirect.

63.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPageAccessDenied (see page 6239)	This is record WebDisp.TPageAccessDenied.

63.4.1 WebDisp.TPageAccessDenied Enumeration

Pascal

```
TPageAccessDenied = (
  adLoginRequired,
  adCantView
);
```

C++

```
enum TPageAccessDenied {
  adLoginRequired,
  adCantView
};
```

File

WebDisp

Description

This is record WebDisp.TPageAccessDenied.

63.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActivateWebModuleEvent (see page 6240)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 6240)	This is type WebDisp.TCanViewPageEvent.
TDispatchActionEvent (see page 6241)	This is type WebDisp.TDispatchActionEvent.
TDispatchActionHandledEvent (see page 6241)	This is type WebDisp.TDispatchActionHandledEvent.
TDispatchImageEvent (see page 6241)	This is type WebDisp.TDispatchImageEvent.
TDispatchImageHandledEvent (see page 6242)	This is type WebDisp.TDispatchImageHandledEvent.
TDispatchPageEvent (see page 6242)	This is type WebDisp.TDispatchPageEvent.
TDispatchPageHandledEvent (see page 6242)	This is type WebDisp.TDispatchPageHandledEvent.
TDispatchPageParamsHandledEvent (see page 6243)	This is type WebDisp.TDispatchPageParamsHandledEvent.
TGetAdapterRequestParamsEvent (see page 6243)	This is type WebDisp.TGetAdapterRequestParamsEvent.
TLocateFileServiceFindFileEvent (see page 6243)	This is type WebDisp.TLocateFileServiceFindFileEvent.
TLocateFileServiceFindStreamEvent (see page 6244)	This is type WebDisp.TLocateFileServiceFindStreamEvent.
TLoginPagedHandledEvent (see page 6244)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 6244)	This is type WebDisp.TLoginRequiredEvent.
TPageAccessDeniedEvent (see page 6245)	This is type WebDisp.TPageAccessDeniedEvent.
TWebRequest (see page 6245)	This is type WebDisp.TWebRequest.
TWebResponse (see page 6245)	This is type WebDisp.TWebResponse.

63.5.1 WebDisp.TActivateWebModuleEvent Type

Pascal

```
TActivateWebModuleEvent = procedure (Sender: TObject; WebModule: TComponent) of object;
```

C++

```
(Sender: TObject; WebModule: TComponent) ( TActivateWebModuleEvent)();
```

File

WebDisp

Description

This is type WebDisp.TActivateWebModuleEvent.

63.5.2 WebDisp.TCanViewPageEvent Type

Pascal

```
TCanViewPageEvent = procedure (Sender: TObject; const PageName: string; var CanView: Boolean;
var AHandled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var CanView: Boolean; var AHandled: Boolean) (
TCanViewPageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TCanViewPageEvent.

63.5.3 WebDisp.TDispatchActionEvent Type

Pascal

```
TDispatchActionEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of
object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TDispatchActionEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchActionEvent.

63.5.4 WebDisp.TDispatchActionHandledEvent Type

Pascal

```
TDispatchActionHandledEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchActionHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchActionHandledEvent.

63.5.5 WebDisp.TDispatchImageEvent Type

Pascal

```
TDispatchImageEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings) ( TDispatchImageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchImageEvent.

63.5.6 WebDisp.TDispatchImageHandledEvent Type

Pascal

```
TDispatchImageHandledEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchImageHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchImageHandledEvent.

63.5.7 WebDisp.TDispatchPageEvent Type

Pascal

```
TDispatchPageEvent = procedure (Sender: TObject; const PageName: string) of object;
```

C++

```
(Sender: TObject; const PageName: string) ( TDispatchPageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageEvent.

63.5.8 WebDisp.TDispatchPageHandledEvent Type

Pascal

```
TDispatchPageHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TDispatchPageHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageHandledEvent.

63.5.9 WebDisp.TDispatchPageParamsHandledEvent Type

Pascal

```
TDispatchPageParamsHandledEvent = procedure (Sender: TObject; const PageName: string; Params:
TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Params: TStrings; var Handled: Boolean) (
TDispatchPageParamsHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageParamsHandledEvent.

63.5.10 WebDisp.TGetAdapterRequestParamsEvent Type

Pascal

```
TGetAdapterRequestParamsEvent = procedure (Sender: TObject; Handler: IUnknown; var Identifer:
string; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Handler: IUnknown; var Identifer: string; Params: TStrings) (
TGetAdapterRequestParamsEvent)();
```

File

WebDisp

Description

This is type WebDisp.TGetAdapterRequestParamsEvent.

63.5.11 WebDisp.TLocateFileServiceFindFileEvent Type

Pascal

```
TLocateFileServiceFindFileEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundFile: string; var AHandled: Boolean) of object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundFile: string;
var AHandled: Boolean) ( TLocateFileServiceFindFileEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLocateFileServiceFindFileEvent.

63.5.12 WebDisp.TLocateFileServiceFindStreamEvent Type

Pascal

```
TLocateFileServiceFindStreamEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundStream: TStream; var AOwned: Boolean; var AHandled: Boolean) of
object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundStream: TStream;
var AOwned: Boolean; var AHandled: Boolean) ( TLocateFileServiceFindStreamEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLocateFileServiceFindStreamEvent.

63.5.13 WebDisp.TLoginPagedHandledEvent Type

Pascal

```
TLoginPagedHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TLoginPagedHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLoginPagedHandledEvent.

63.5.14 WebDisp.TLoginRequiredEvent Type

Pascal

```
TLoginRequiredEvent = procedure (Sender: TObject; const PageName: string; var Required:
```

```
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Required: Boolean) ( TLoginRequiredEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLoginRequiredEvent.

63.5.15 WebDisp.TPageAccessDeniedEvent Type

Pascal

```
TPageAccessDeniedEvent = procedure (Sender: TObject; const PageName: string; Reason:
TPageAccessDenied; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Reason: TPageAccessDenied; var Handled: Boolean) (
TPageAccessDeniedEvent)();
```

File

WebDisp

Description

This is type WebDisp.TPageAccessDeniedEvent.

63.5.16 WebDisp.TWebRequest Type

Pascal

```
TWebRequest = HTTPApp.TWebRequest;
```

C++

```
HTTPApp.TWebRequest TWebRequest;
```

File

WebDisp

Description

This is type WebDisp.TWebRequest.

63.5.17 WebDisp.TWebResponse Type

Pascal

```
TWebResponse = HTTPApp.TWebResponse;
```

C++

```
HTTPApp.TWebResponse TWebResponse;
```

File

WebDisp

Description

This is type WebDisp.TWebResponse.

63.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sFailurePageParamName (see page 6247)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 6247)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 6248)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 6248)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 6248)	This is constant WebDisp.sSuccessPageParamName.

63.6.1 WebDisp.sFailurePageParamName Constant

Pascal

```
sFailurePageParamName = '__fp';
```

C++

```
sFailurePageParamName = '__fp';
```

File

WebDisp

Description

This is constant WebDisp.sFailurePageParamName.

63.6.2 WebDisp.sPageParamName Constant

Pascal

```
sPageParamName = '__p';
```

C++

```
sPageParamName = '__p';
```

File

WebDisp

Description

This is constant WebDisp.sPageParamName.

63.6.3 WebDisp.sRequestIDParamName Constant

Pascal

```
sRequestIDParamName = '__id';
```

C++

```
sRequestIDParamName = '__id';
```

File

WebDisp

Description

This is constant WebDisp.sRequestIDParamName.

63.6.4 WebDisp.sSessionCookieName Constant

Pascal

```
sSessionCookieName = 'WebBrokerSessionID';
```

C++

```
sSessionCookieName = 'WebBrokerSessionID';
```

File

WebDisp

Description

This is constant WebDisp.sSessionCookieName.

63.6.5 WebDisp.sSuccessPageParamName Constant

Pascal

```
sSuccessPageParamName = '__sp';
```

C++

```
sSuccessPageParamName = '__sp';
```

File

WebDisp

Description

This is constant WebDisp.sSuccessPageParamName.

64 WebFact Namespace

This is namespace WebFact.

Topics

Name	Description
Classes (see page 6250)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6275)	The following table lists structs, records, enums in this documentation.
Types (see page 6276)	The following table lists types in this documentation.

Classes

	Name	Description
	TBaseWebModuleFactory (see page 6250)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 6255)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 6260)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 6265)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 6267)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 6269)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 6271)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 6273)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TWebPageAccessFlags (see page 6275)	This is record WebFact.TWebPageAccessFlags.

Types

Name	Description
TWebPageAccess (see page 6276)	This is type WebFact.TWebPageAccess.

64.1 Classes

The following table lists classes in this documentation.

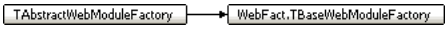
Classes

	Name	Description
	TBaseWebModuleFactory (see page 6250)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 6255)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 6260)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 6265)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 6267)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 6269)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 6271)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 6273)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

64.1.1 TBaseWebModuleFactory Class

TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.

Class Hierarchy



Pascal

```
TBaseWebModuleFactory = class(TAbstractWebModuleFactory);
```

C++

```
class TBaseWebModuleFactory : public TAbstractWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instance. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instance for reuse with subsequent request messages.

TBaseWebModuleFactory descendants generate Web modules that do generate Web pages. These Web modules can contain components, such as adapters, that perform tasks for the WebSnap application.

Do not create instances of TBaseWebModuleFactory. Although TBaseWebModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebModuleFactory descendant such as TWebDataModuleFactory (see page 6269) or TWebAppDataModuleFactory (see page 6265).

See Also

TWebAppDataModuleFactory (see page 6265)

TWebDataModuleFactory (see page 6269)

TWebRequestHandler (see page 6600)

64.1.1.1 TBaseWebModuleFactory Members

The following tables list the members exposed by TBaseWebModuleFactory.

Protected Methods

	Name	Description
	Create (see page 6251)	Creates and initializes an instance of TBaseWebModuleFactory (see page 6250).
	GetCacheMode (see page 6252)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6252)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6253)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6253)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6254)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6253)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6254)	Prevents the WebSnap application from freeing the Web module instance.

64.1.1.2 TBaseWebModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebModuleFactory (see page 6250).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode;
ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web module factory. Typically, Web module factory objects are created in the initialization section of the unit for the Web module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web module.

AComponentClass is the class of the Web module that this factory object creates. It becomes the value of the ComponentClass property.

ACreateMode describes whether the factory should create instances of the Web module for every request message, or only when the application explicitly requests the Web module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

ComponentClass

CreateMode

CacheMode

64.1.1.3 TBaseWebModuleFactory Methods

The methods of the TBaseWebModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 6252)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6252)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6253)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6253)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6254)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6253)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6254)	Prevents the WebSnap application from freeing the Web module instance.

64.1.1.3.1 TBaseWebModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 6251)

64.1.1.3.2 TBaseWebModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 6251)

64.1.1.3.3 TBaseWebModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 6251)

64.1.1.3.4 TBaseWebModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory (see page 6250), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory (see page 6265)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6267)

64.1.1.3.5 TBaseWebModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

64.1.1.3.6 TBaseWebModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

64.1.1.3.7 TBaseWebModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebModuleFactory (see page 6250), PreventDestruction changes the CacheMode property to caCache.

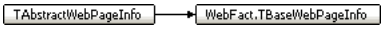
See Also

CacheMode

64.1.2 TBaseWebPageInfo Class

TBaseWebPageInfo stores information about a page registered with a Web page module factory.

Class Hierarchy



Pascal

```
TBaseWebPageInfo = class(TAbstractWebPageInfo);
```

C++

```
class TBaseWebPageInfo : public TAbstractWebPageInfo;
```

File

WebFact

Description

The TBaseWebPageInfo component stores basic information about an HTML page such as the filename, page name, page title, descriptive text, publish status, and access rights. This information is stored with the Web page module factory.

An instance of a TBaseWebPageInfo descendant, such as TWebPageInfo (see page 6271), is supplied to the constructor of the factory object that creates instances of a Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard. You can change this generated code, which appears in the generated unit for the Web page module, to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageInfo (see page 6271)

TAbstractWebContext (see page 5935)

TWebAppPageModuleFactory (see page 6267)

TWebPageModuleFactory (see page 6273)

WebContext

64.1.2.1 TBaseWebPageInfo Members

The following tables list the members exposed by TBaseWebPageInfo.

Protected Methods

	Name	Description
	Create (see page 6256)	Creates and initializes a TBaseWebPageInfo (see page 6255) object.
	GetIsLoginRequired (see page 6257)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6257)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6257)	Returns the value of the PageDescription property.
	GetPageFile (see page 6258)	Returns the value of the PageFile property.
	GetPageHREF (see page 6258)	Returns the value of the PageHREF property.

	GetPageName (see page 6258)	Returns the value of the PageName property.
	GetPageTitle (see page 6259)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6259)	Returns the value of the ViewAccess property.
	SetFactory (see page 6260)	Connects a specified Web page module factory with the Web page information object.

64.1.2.2 TBaseWebPageInfo.Create Constructor

Creates and initializes a TBaseWebPageInfo (see page 6255) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess; const AFile: string; const AName: string; const
ATitle: string; const ADescription: string; const AViewAccess: string);
```

C++

```
__fastcall TBaseWebPageInfo(TWebPageAccess AAccess, const UnicodeString AFile, const
UnicodeString AName, const UnicodeString ATitle, const UnicodeString ADescription, const
UnicodeString AViewAccess);
```

Description

Typically, you do not need to create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

AFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

AName is the logical name of the associated Web page. It becomes the value of the PageName property.

ATitle is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

pagefile

pagename

pagetitle

PageDescription

viewaccess

IsPublished

IsLoginRequired

64.1.2.3 TBaseWebPageInfo Methods

The methods of the TBaseWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetIsLoginRequired (see page 6257)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6257)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6257)	Returns the value of the PageDescription property.
	GetPageFile (see page 6258)	Returns the value of the PageFile property.
	GetPageHREF (see page 6258)	Returns the value of the PageHREF property.
	GetPageName (see page 6258)	Returns the value of the PageName property.
	GetPageTitle (see page 6259)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6259)	Returns the value of the ViewAccess property.
	SetFactory (see page 6260)	Connects a specified Web page module factory with the Web page information object.

64.1.2.3.1 TBaseWebPageInfo.GetIsLoginRequired Method

Returns the value of the IsLoginRequired property.

Pascal

```
function GetIsLoginRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsLoginRequired();
```

Description

GetIsLoginRequired is the protected read implementation of the IsLoginRequired property. It returns true if wpLoginRequired is included in the access flags passed to the constructor.

See Also

IsLoginRequired

Create (see page 6256)

64.1.2.3.2 TBaseWebPageInfo.GetIsPublished Method

Returns the value of the IsPublished property.

Pascal

```
function GetIsPublished: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsPublished();
```

Description

GetIsPublished is the protected read implementation of the IsPublished property. It returns true if wpPublished is included in the access flags passed to the constructor.

See Also

IsPublished

Create (see page 6256)

64.1.2.3.3 TBaseWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the protected read implementation of the PageDescription property. It returns the description string that is passed to the constructor.

See Also

PageDescription

Create (see page 6256)

64.1.2.3.4 TBaseWebPageInfo.GetPageFile Method

Returns the value of the PageFile property.

Pascal

```
function GetPageFile: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageFile();
```

Description

GetPageFile is the protected read implementation of the PageFile property.

See Also

pagefile

64.1.2.3.5 TBaseWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the protected read implementation of the PageHREF property. It returns the HTTP path string that is passed to the constructor.

See Also

PageHREF

Create (see page 6256)

64.1.2.3.6 TBaseWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageName();
```

Description

GetPageName is the protected read implementation of the PageName property. It returns the page name that is passed to the constructor.

See Also

pagename

Create (see page 6256)

64.1.2.3.7 TBaseWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the protected read implementation of the PageTitle property. It returns the title that is passed to the constructor.

See Also

pagetitle

Create (see page 6256)

64.1.2.3.8 TBaseWebPageInfo.GetViewAccess Method

Returns the value of the ViewAccess property.

Pascal

```
function GetViewAccess: string; override;
```

C++

```
virtual __fastcall UnicodeString GetViewAccess();
```

Description

GetViewAccess is the protected read implementation of the ViewAccess property. It returns the view access string that is passed to the constructor.

See Also

viewaccess

Create (see page 6256)

64.1.2.3.9 TBaseWebPageInfo.SetFactory Method

Connects a specified Web page module factory with the Web page information object.

Pascal

```
procedure SetFactory(AFactory: TAbstractWebPageModuleFactory); override;
```

C++

```
virtual __fastcall SetFactory(TAbstractWebPageModuleFactory AFactory);
```

Description

SetFactory is the protected write implementation of the Factory property.

AFactory is the Web page module factory object that instantiates the Web page module that this object describes.

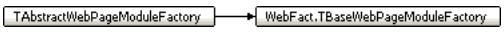
See Also

Factory

64.1.3 TBaseWebPageModuleFactory Class

TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.

Class Hierarchy



Pascal

```
TBaseWebPageModuleFactory = class(TAbstractWebPageModuleFactory);
```

C++

```
class TBaseWebPageModuleFactory : public TAbstractWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instances. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instances for reuse with subsequent request messages.

TBaseWebPageModuleFactory descendants generate Web page modules. In addition to holding components that perform tasks for the WebSnap application, Web page modules can generate a Web page. The Web page is described by a Web page information object, which the Web page module factory makes available as its WebPageInfo property.

Do not create instances of TBaseWebPageModuleFactory. Although TBaseWebPageModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebPageModuleFactory descendant such as TWebPageModuleFactory (see page 6273) or TWebAppPageModuleFactory (see page 6267).

See Also

TWebAppPageModuleFactory (see page 6267)

TWebPageModuleFactory (see page 6273)

TWebRequestHandler (see page 6600)

TWebPageInfo (see page 6271)

64.1.3.1 TBaseWebPageModuleFactory Members

The following tables list the members exposed by TBaseWebPageModuleFactory.

Protected Methods

	Name	Description
	Create (see page 6261)	Creates and initializes an instance of TBaseWebPageModuleFactory (see page 6260).
	GetCacheMode (see page 6262)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6262)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6263)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6263)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6264)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6263)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6264)	Prevents the WebSnap application from freeing the Web module instance.

64.1.3.2 TBaseWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebPageModuleFactory (see page 6260).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode; ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web page module factory. Typically, Web page module factory objects are created in the initialization section of the unit for the Web page module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web page module.

AComponentClass is the class of the Web page module that this factory object creates. It becomes the value of the ComponentClass property.

AWebPageInfo is the Web page information object that describes the Web page that this factory's Web page module generates. It becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

WebPageInfo

ComponentClass

CreateMode

CacheMode

64.1.3.3 TBaseWebPageModuleFactory Methods

The methods of the TBaseWebPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 6262)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6262)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6263)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6263)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6264)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6263)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6264)	Prevents the WebSnap application from freeing the Web module instance.

64.1.3.3.1 TBaseWebPageModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 6261)

64.1.3.3.2 TBaseWebPageModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 6261)

64.1.3.3.3 TBaseWebPageModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 6261)

64.1.3.3.4 TBaseWebPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory (see page 6260), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory (see page 6267)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6267)

64.1.3.3.5 TBaseWebPageModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

64.1.3.3.6 TBaseWebPageModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

64.1.3.3.7 TBaseWebPageModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebPageModuleFactory (see page 6260), PreventDestruction changes the CacheMode property to caCache.

See Also

CacheMode

64.1.4 TWebAppDataModuleFactory Class

TWebAppDataModule is the factory object for WebSnap application data modules.

Class Hierarchy

Pascal

```
TWebAppDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebAppDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebAppDataModuleFactory contains creation flags. Creation flags determine whether the module should be cached for future requests or destroyed after each request and created for each new request.

TWebAppDataModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppDataModule (see page 6550)

TWebAppPageModuleFactory (see page 6267)

TWebDataModuleFactory (see page 6269)

TWebRequestHandler (see page 6600)

TWebPageModuleFactory (see page 6273)

64.1.4.1 TWebAppDataModuleFactory Members

The following tables list the members exposed by TWebAppDataModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6252)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6252)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6253)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6266)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6254)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6266)	Creates and initializes an instance of TWebAppDataModuleFactory (see page 6265).
	GetModule (see page 6253)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6254)	Prevents the WebSnap application from freeing the Web module instance.

64.1.4.2 TWebAppDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppDataModuleFactory (see page 6265).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACacheMode: TWebModuleCacheMode =
caCache);
```

C++

```
__fastcall TWebAppDataModuleFactory(TComponentClass AComponentClass, TWebModuleCacheMode
ACacheMode = caCache);
```

Description

Call Create to instantiate a Web application module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application module's unit.

AComponentClass is the class of the Web data module that this factory object creates. It becomes the value of the ComponentClass property.

ACacheMode indicates whether the application can cache Web application module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

64.1.4.3 TWebAppDataModuleFactory Methods

The methods of the TWebAppDataModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetIsAppModule (see page 6266)	Returns the value of the IsAppModule property.

64.1.4.3.1 TWebAppDataModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6267)

64.1.5 TWebAppPageModuleFactory Class

TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TWebAppPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebAppPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

TWebAppPageModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppPageModule (see page 6554)

TWebAppDataModuleFactory (see page 6265)

TWebPageModuleFactory (see page 6273)

TWebRequestHandler (see page 6600)

TWebPageInfo (see page 6271)

TWebDataModuleFactory (see page 6269)

64.1.5.1 TWebAppPageModuleFactory Members

The following tables list the members exposed by TWebAppPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6262)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6262)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6263)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6269)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6264)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6268)	Creates and initializes an instance of TWebAppPageModuleFactory (see page 6267).
	GetModule (see page 6263)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6264)	Prevents the WebSnap application from freeing the Web module instance.

64.1.5.2 TWebAppPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppPageModuleFactory (see page 6267).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebAppPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application page module's unit.

AComponentClass is the class of the Web application page module that this factory object creates.

AWebPageInfo describes the attributes of the Web application page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACacheMode indicates whether the factory can cache Web application page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

WebPageInfo

64.1.5.3 TWebAppPageModuleFactory Methods

The methods of the TWebAppPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetIsAppModule (see page 6269)	Returns the value of the IsAppModule property.

64.1.5.3.1 TWebAppPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6267)

64.1.6 TWebDataModuleFactory Class

TWebDataModuleFactory is the factory class for WebSnap data modules.

Class Hierarchy

Pascal

```
TWebDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. (In Delphi, the factory object is typically created in the initialization section of the module source file; in C++, it is added in a #pragma startup.)The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebDataModuleFactory includes caching flags and creation flags. Caching flags determine whether the module should be cached for future request, or destroyed after each request and created for each new request. Creation flags determine whether the module should be automatically created for every request or created on demand.

The WebSnap application may register multiple Web data module classes. Each class is registered with an instance of TWebDataModuleFactory.

See Also

TWebDataModule (see page 6562)

TWebAppDataModuleFactory (see page 6265)

TWebAppPageModuleFactory (see page 6267)

TWebRequestHandler (see page 6600)

TWebPageModuleFactory (see page 6273)

64.1.6.1 TWebDataModuleFactory Members

The following tables list the members exposed by TWebDataModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6252)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6252)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6253)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6253)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6254)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6270)	Creates and initializes an instance of TWebDataModuleFactory (see page 6269).
	GetModule (see page 6253)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6254)	Prevents the WebSnap application from freeing the Web module instance.

64.1.6.2 TWebDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebDataModuleFactory (see page 6269).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode =
crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebDataModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web data module factory. Typically, the Web module wizard generates the call to the constructor in the initialization section of the Web data module's unit.

AComponentClass is the class of the Web data module that this factory object creates.

ACreateMode describes whether the factory should create instances of the Web data module for every request message, or only when the application explicitly requests the Web data module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web data module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

64.1.7 TWebPageInfo Class

TWebPageInfo describes a WebSnap page.

Class Hierarchy

Pascal

```
TWebPageInfo = class(TBaseWebPageInfo);
```

C++

```
class TWebPageInfo : public TBaseWebPageInfo;
```

File

WebFact

Description

The TWebPageInfo module stores information relating to a web page, such as the template filename, page name, title, description, and access rights. This information is part of the web page module factory. This information is stored as part of the factory rather than as part of the module so that it can be accessed without creating the web page module.

An instance of a TWebPageInfo is supplied to the constructor of the factory object that creates instances of the Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard (in Delphi, the generated code appears in the initialization section of the generated unit for the Web page module; in C++, it appears in a #pragma startup). You can change the generated code to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageModuleFactory (see page 6273)

TWebAppPageModuleFactory (see page 6267)

TWebModuleFactoryList (see page 6588)

TAbstractWebContext (see page 5935)

WebContext

64.1.7.1 TWebPageInfo Members

The following tables list the members exposed by TWebPageInfo.

Protected Methods

	Name	Description
	GetIsLoginRequired (see page 6257)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6257)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6257)	Returns the value of the PageDescription property.
	GetPageFile (see page 6258)	Returns the value of the PageFile property.
	GetPageHREF (see page 6258)	Returns the value of the PageHREF property.
	GetPageName (see page 6258)	Returns the value of the PageName property.
	GetPageTitle (see page 6259)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6259)	Returns the value of the ViewAccess property.
	SetFactory (see page 6260)	Connects a specified Web page module factory with the Web page information object.

Public Methods

	Name	Description
	Create (see page 6272)	Creates and initializes a TWebPageInfo (see page 6271) object.

64.1.7.2 TWebPageInfo.Create Constructor

Creates and initializes a TWebPageInfo (see page 6271) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess = [wpPublished]; const APageFile: string = '.html';
const APageName: string = ''; const ACaption: string = ''; const ADescription: string = '';
const AViewAccess: string = '');
```

C++

```
__fastcall TWebPageInfo(TWebPageAccess AAccess = [wpPublished], const UnicodeString APageFile
= '.html', const UnicodeString APageName = '', const UnicodeString ACaption = '', const
UnicodeString ADescription = '', const UnicodeString AViewAccess = '');
```

Description

Typically, you do not need create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

APageFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

APageName is the logical name of the associated Web page. It becomes the value of the PageName property.

ACaption is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

pagefile

pagename

pagetitle

PageDescription

viewaccess

IsPublished

IsLoginRequired

64.1.8 TWebPageModuleFactory Class

TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TWebPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that specify how the WebSnap application should manage the creation, destruction, and use of the Web module. TWebPageModuleFactory also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Note: In the Delphi language, the factory object is created in the initialization section of the module source file.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

WebSnap applications can contain multiple Web page module factories (hence multiple Web page modules), each of which creates the Web page module for a different Web page that the application can create.

See Also

TWebPageModule (see page 6569)

TWebAppDataModuleFactory (see page 6265)

TWebAppPageModuleFactory (see page 6267)

TWebRequestHandler (see page 6600)

TWebPageInfo (see page 6271)

TWebDataModuleFactory (see page 6269)

64.1.8.1 TWebPageModuleFactory Members

The following tables list the members exposed by TWebPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6262)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6262)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6263)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6263)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6264)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6274)	Creates and initializes an instance of TWebPageModuleFactory (see page 6273).
	GetModule (see page 6263)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6264)	Prevents the WebSnap application from freeing the Web module instance.

64.1.8.2 TWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebPageModuleFactory (see page 6273).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode = crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode =
caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the Web page module wizard generates the call to the constructor in the Web page module's unit. In Delphi, the constructor call is placed in the initialization section of the unit. In C++, the call is placed in a #pragma startup.

AComponentClass is the class of the Web page module that this factory object creates.

AWebPageInfo describes the attributes of the Web page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

WebPageInfo

64.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebPageAccessFlags (see page 6275)	This is record WebFact.TWebPageAccessFlags.

64.2.1 WebFact.TWebPageAccessFlags Enumeration

Pascal

```
TWebPageAccessFlags = (
  wpPublished,
  wpLoginRequired
);
```

C++

```
enum TWebPageAccessFlags {
  wpPublished,
  wpLoginRequired
};
```

File

WebFact

Description

This is record WebFact.TWebPageAccessFlags.

64.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWebPageAccess (see page 6276)	This is type WebFact.TWebPageAccess.

64.3.1 WebFact.TWebPageAccess Type

Pascal

```
TWebPageAccess = set of TWebPageAccessFlags;
```

C++

```
TWebPageAccessFlags TWebPageAccess;
```

File

WebFact

Description

This is type WebFact.TWebPageAccess.

65 WebForm Namespace

This is namespace WebForm.

Topics

Name	Description
Classes (see page 6280)	The following table lists classes in this documentation.
Interfaces (see page 6476)	The following table lists interfaces in this documentation.
Functions (see page 6485)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6488)	The following table lists structs, records, enums in this documentation.
Types (see page 6494)	The following table lists types in this documentation.
Constants (see page 6496)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 6281)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 6284)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 6286)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 6294)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 6301)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 6307)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 6312)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6339) or TAdapterDisplayColumn (see page 6333).
	TAdapterDataDisplay (see page 6324)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 6333)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 6339)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 6349)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 6355)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 6365)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 6367)	TAdapterFieldGroup is a container for Adapter (see page 6370) display field components such as TAdapterDisplayField (see page 6339). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 6373)	TAdapterForm generates an HTML form displaying Adapter fields.

	TAdapterFormLayout (see page 6376)	This is class WebForm.TAdapterFormLayout.
	TAdapterGrid (see page 6386)	TAdapterGrid is a container for Adapter (see page 6389) display field components such as TAdapterDisplayColumn (see page 6333). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 6392)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 6397)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 6409)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 6421)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6307).
	TCustomAdapterDisplayColumn (see page 6428)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 6434)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 6442)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 6445)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 6449)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 6456)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 6462)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.
	TDesigntimeWarnings (see page 6470)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 6472)	This is class WebForm.TGeneratedFunctions.

Constants

Name	Description
sDisplayText (see page 6496)	This is constant WebForm.sDisplayText.
sFieldText (see page 6496)	This is constant WebForm.sFieldText.
sFormGet (see page 6497)	This is constant WebForm.sFormGet.
sFormPost (see page 6497)	This is constant WebForm.sFormPost.
sMultipartForm (see page 6497)	This is constant WebForm.sMultipartForm.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 6485)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 6485)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 6486)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 6486)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 6486)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 6487)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 6487)	This is function WebForm.MakeAdapterVariableName.

Interfaces

	Name	Description
	IAdapterDisplay (see page 6476)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 6476)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 6477)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 6478)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 6479)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 6481)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 6482)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 6483)	This is class WebForm.IGetVariablesContainerOfDisplay.

Structs, Records, Enums

	Name	Description
	TActionButtonHideOption (see page 6488)	TActionButtonHideOptions (see page 6494) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 6489)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 6489)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 6490)	TDisplayFieldHideOptions (see page 6494) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 6491)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 6491)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 6492)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 6493)	This is record WebForm.TLayoutState.

Types

Name	Description
TActionButtonHideOptions (see page 6494)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 6494)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 6494)	This is type WebForm.TDisplayFieldHideOptions.

65.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 6281)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 6284)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 6286)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 6294)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 6301)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 6307)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 6312)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6339) or TAdapterDisplayColumn (see page 6333).
	TAdapterDataDisplay (see page 6324)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 6333)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 6339)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 6349)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 6355)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 6365)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 6367)	TAdapterFieldGroup is a container for Adapter (see page 6370) display field components such as TAdapterDisplayField (see page 6339). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 6373)	TAdapterForm generates an HTML form displaying Adapter fields.
	TAdapterFormLayout (see page 6376)	This is class WebForm.TAdapterFormLayout.
	TAdapterGrid (see page 6386)	TAdapterGrid is a container for Adapter (see page 6389) display field components such as TAdapterDisplayColumn (see page 6333). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 6392)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 6397)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 6409)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 6421)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6307).
	TCustomAdapterDisplayColumn (see page 6428)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 6434)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 6442)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 6445)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 6449)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 6456)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 6462)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

	TDesigntimeWarnings (see page 6470)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 6472)	This is class WebForm.TGeneratedFunctions.

65.1.1 TAbstractControlRequirements Class

TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy

Pascal

```
TAbstractControlRequirements = class;
```

C++

```
class TAbstractControlRequirements;
```

File

WebForm

Description

Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 6312) and TAdapterDataDisplay (see page 6324) descendants) use descendants of TAbstractControlRequirements. The descendants of TAbstractControlRequirements store the variable and function declarations the adapter form must include before the control group can assemble HTML controls for the fields of the adapter it represents.

TAbstractControlRequirements introduces methods to

Store a specified function or variable (along with its value).

Query whether the function or variable declaration has already been added to the list of requirements.

Store an indication of whether the control requires that it be contained in a multipart form.

TAbstractControlRequirements is abstract or, in C++ terminology, pure virtual. It does not implement any of the methods it declares. WebSnap applications use TAbstractControlRequirements descendants such as TControlRequirements (see page 6392), which provide implementations for these methods.

See Also

TAdapterControlGroup (see page 6312)

TAdapterDataDisplay (see page 6324)

TAdapterForm (see page 6373)

TControlRequirements (see page 6392)

65.1.1.1 TAbstractControlRequirements Members

The following tables list the members exposed by TAbstractControlRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 6282)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6282)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6283)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6283)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6284)	Stores the fact that the generated HTML form should be multipart.

65.1.1.2 TAbstractControlRequirements Methods

The methods of the TAbstractControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6282)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6282)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6283)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6283)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6284)	Stores the fact that the generated HTML form should be multipart.

65.1.1.2.1 TAbstractControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody) = 0;
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the form.

See Also

IsFunctionDefined (see page 6283)

DefineVariable (see page 6282)

65.1.1.2.2 TAbstractControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue,
Boolean AAdapter) = 0;
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

DefineFunction (see page 6282)

IsVariableDefined (see page 6283)

Adapter (see page 6322)

65.1.1.2.3 TAbstractControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName) = 0;
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 6282) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 6282)

IsVariableDefined (see page 6283)

65.1.1.2.4 TAbstractControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName) = 0;
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 6282) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 6282)

IsFunctionDefined (see page 6283)

65.1.1.2.5 TAbstractControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; virtual; abstract;
```

C++

```
virtual __fastcall RequiresMultipartForm() = 0;
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

65.1.2 TAbstractFormRequirements Class

Class Hierarchy

Pascal

```
TAbstractFormRequirements = class;
```

C++

```
class TAbstractFormRequirements;
```

File

WebForm

Description

This is class WebForm.TAbstractFormRequirements.

65.1.2.1 TAbstractFormRequirements Members

The following tables list the members exposed by TAbstractFormRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 6285)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 6285)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 6285)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 6286)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

65.1.2.2 TAbstractFormRequirements Methods

The methods of the TAbstractFormRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6285)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 6285)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 6285)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 6286)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

65.1.2.2.1 TAbstractFormRequirements.DefineFunction Method

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody) = 0;
```

Description

This is DefineFunction, a member of class TAbstractFormRequirements.

65.1.2.2.2 TAbstractFormRequirements.DefineVariable Method

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue) = 0;
```

Description

This is DefineVariable, a member of class TAbstractFormRequirements.

65.1.2.2.3 TAbstractFormRequirements.IsFunctionDefined Method

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName) = 0;
```

Description

This is IsFunctionDefined, a member of class TAbstractFormRequirements.

65.1.2.2.4 TAbstractFormRequirements.IsVariableDefined Method

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName) = 0;
```

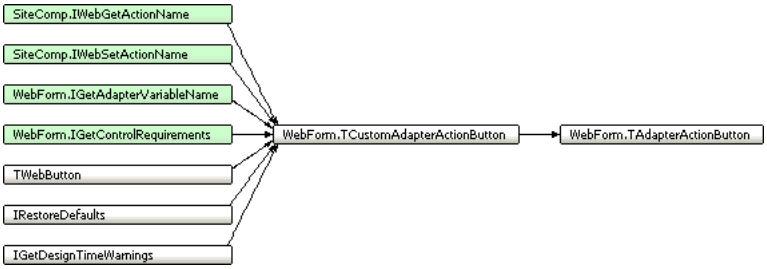
Description

This is IsVariableDefined, a member of class TAbstractFormRequirements.

65.1.3 TAdapterActionButton Class

TAdapterActionButton generates HTML controls for executing adapter actions.

Class Hierarchy



Pascal

```
TAdapterActionButton = class(TCustomAdapterActionButton);
```

C++

```
class TAdapterActionButton : public TCustomAdapterActionButton;
```

File

WebForm

Description

Add TAdapterActionButton to the hierarchy of Web items that an adapter page producer (TAdapterPageProducer) maintains to include a control in the HTML form that executes an adapter action. When the end user clicks the control, it sends a request message that executes the adapter action. Use the ActionName (see page 6288) property to specify the adapter action that the generated control executes.

TAdapterActionButton can generate a variety of controls, including labeled buttons, images that the user can click, and hypertext links. Use the DisplayType (see page 6290) property to specify the type of control you want. Other properties customize the appearance of the control, or indicate when it is included in HTML forms.

See Also

TAdapterPageProducer (see page 1076)

TCustomAdapterAction (see page 5565)

TAdapterForm (see page 6373)

65.1.3.1 TAdapterActionButton Members

The following tables list the members exposed by TAdapterActionButton.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6400)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6400)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6400)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6400)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 6401)	Returns the value of the Caption property.
	GetControlRequirements (see page 6401)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6401)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6401)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
V	GetLayoutAttributes (see page 6402)	Returns the layout object that specifies how the generated control should look.
V	ImplContent (see page 6402)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 6402)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 6403)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
V	Notification (see page 6403)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6403)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 6404)	This is SetActionName, a member of class TCustomAdapterActionButton.
V	SetCaption (see page 6404)	Sets the value of the Caption property.

Public Methods

	Name	Description
V	Create (see page 6398)	Creates and initializes an instance of TCustomAdapterActionButton (see page 6397).
V	Destroy (see page 6399)	Frees an instance of TCustomAdapterActionButton (see page 6397).
	GetActionName (see page 1029)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	SetActionName (see page 1056)	This is SetActionName, a member of class IWebSetActionName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	ActionName (see page 6288)	Specifies the name of the action taken by the Web button.
	Align (see page 6288)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 6289)	Specifies the caption that appears on the button.
	Custom (see page 6289)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 6290)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6290)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6291)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6291)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6291)	Specifies the height of the generated element if it is a clickable image.

	ImageProducer (see page 6292)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6292)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6293)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 6293)	Adds style attributes to the generated button control.
	StyleRule (see page 6293)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 6294)	Specifies the vertical alignment of the generated HTML control.

65.1.3.2 TAdapterActionButton Properties

The properties of the TAdapterActionButton class are listed here.

Published Properties

	Name	Description
	ActionName (see page 6288)	Specifies the name of the action taken by the Web button.
	Align (see page 6288)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 6289)	Specifies the caption that appears on the button.
	Custom (see page 6289)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 6290)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6290)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6291)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6291)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6291)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6292)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6292)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6293)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 6293)	Adds style attributes to the generated button control.
	StyleRule (see page 6293)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 6294)	Specifies the vertical alignment of the generated HTML control.

65.1.3.2.1 TAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;__property UnicodeString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 5572)

65.1.3.2.2 TAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6409)

DisplayColumns (see page 6405)

ItemCount (see page 5463)

65.1.3.2.3 TAdapterActionButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom

GetCaption

SetCaption

65.1.3.2.4 TAdapterActionButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value

has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption

Style

65.1.3.2.5 TAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 5463)

DisplayType (see page 6406)

65.1.3.2.6 TAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer property to a component that can provide the image or images.

See Also

ImageProducer (see page 6408)

Caption

GetDisplayStyle (see page 5445)

65.1.3.2.7 TAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property UnicodeString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 6408)

pagename

65.1.3.2.8 TAdapterActionButton.HideOptions Property

Controls when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton always generates a control for its associated adapter action.

See Also

GetVisible (see page 5450)

WebEnabled (see page 5461)

HasExecuteAccess (see page 5451)

65.1.3.2.9 TAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6408)

ImageWidth (see page 6408)

DisplayType (see page 6406)

65.1.3.2.10 TAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 6406)

ImageHeight (see page 6407)

ImageWidth (see page 6408)

65.1.3.2.11 TAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6408)

ImageHeight (see page 6407)

DisplayType (see page 6406)

65.1.3.2.12 TAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 6406)

pagename

65.1.3.2.13 TAdapterActionButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom

StyleRule

65.1.3.2.14 TAdapterActionButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style

Styles

StylesFile

Styles (see page 1097)

StylesFile (see page 1098)

65.1.3.2.15 TAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6405)

DisplayColumns (see page 6405)

ItemCount (see page 5463)

65.1.4 TAdapterActionsListHelper Class

TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.

Class Hierarchy

Pascal

```
TAdapterActionsListHelper = class(TObject);
```

C++

```
class TAdapterActionsListHelper : public TObject;
```

File

WebForm

Description

AdapterActionsListHelper is a helper object that works with components that generate a collection of controls for executing adapter actions. Such components include TAdapterCommandColumn (see page 6301) and TAdapterCommandGroup (see page 6307). TAdapterActionsListHelper handles common tasks that allow the display component to work with the IDE. These tasks include locating child Web items for specified actions, listing the child Web items (adapter action buttons) of the display component, and specifying the text of menu commands that operate on the set of child Web items.

See Also

TAdapterCommandColumn (see page 6301)

TAdapterCommandGroup (see page 6307)

TAdapterActionButton (see page 6286)

65.1.4.1 TAdapterActionsListHelper Members

The following tables list the members exposed by TAdapterActionsListHelper.

Protected Methods

	Name	Description
	FindAction (see page 6297)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 6297)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 6298)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 6298)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 6299)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 6299)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 6300)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 6300)	Indicates whether there is a child Web item that corresponds to a specified action.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6296)	Creates and initializes an instance of TAdapterActionsListHelper (see page 6294).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AddDefaultActions (see page 6301)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

65.1.4.2 TAdapterActionsListHelper.Create Constructor

Creates and initializes an instance of TAdapterActionsListHelper (see page 6294).

Pascal

```
constructor Create(AComponent: TComponent; AHelper: TWebContainerComponentHelper);
```

C++

```
__fastcall TAdapterActionsListHelper(TComponent * AComponent, TWebContainerComponentHelper
AHelper);
```

Description

Do not instantiate TAdapterActionsListHelper (see page 6294). It is created by display components that use it to handle their interactions with the IDE.

AComponent is the adapter display component that uses TAdapterActionsListHelper (see page 6294) to interact with the IDE. This component is a Web item, such as TAdapterCommandColumn (see page 6301) or TAdapterCommandGroup (see page 6307), that assembles a set of HTML controls for executing adapter actions.

AHelper is the helper object that ACompnent uses to manage its list of child Web items.

See Also

TAdapterCommandColumn (see page 6301)

TAdapterCommandGroup (see page 6307)

65.1.4.3 TAdapterActionsListHelper Methods

The methods of the TAdapterActionsListHelper class are listed here.

Protected Methods

	Name	Description
	FindAction (see page 6297)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 6297)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 6298)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 6298)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 6299)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 6299)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 6300)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 6300)	Indicates whether there is a child Web item that corresponds to a specified action.

65.1.4.3.1 TAdapterActionsListHelper.FindAction Method

Returns the child Web item that corresponds to the specified adapter action.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

FindAction searches through the list of child Web items for a component that corresponds to the specified adapter action.

AName is the name of the adapter action to locate.

FindAction returns the child Web item (typically a TAdapterActionButton (see page 6286) instance) that generates a control for the specified action. If there is no child Web item for the specified action, FindAction returns nil (Delphi) or NULL (C++).

See Also

GetActionsList (see page 6297)

GetVisibleActions (see page 6300)

IsActionInUse (see page 6300)

ActionName (see page 6405)

65.1.4.3.2 TAdapterActionsListHelper.GetActionsList Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
procedure GetActionsList(List: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings List);
```

Description

GetActionsList searches through the variables that have been defined for child Web items that represent adapter actions. It adds the names of any actions it finds to the list, along with the class of the Web item that represents each action.

List is the list to fill with the names of adapter actions for which there are child Web items.

See Also

FindAction (see page 6297)

GetVisibleActions (see page 6300)

IsActionInUse (see page 6300)

ActionName (see page 6405)

65.1.4.3.3 TAdapterActionsListHelper.GetAddAllFieldsItem Method

Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

The IDE calls GetAddAllFieldsItem to customize the label of the command that adds all the default items to the list of child Web items. Because TAdapterActionsListHelper (see page 6294) is specialized for controls that contain adapter action buttons, it returns the string 'Add All Commands'.

See Also

GetAddFieldsDlgCaption (see page 6298)

GetAddFieldsItem (see page 6299)

GetNewFieldPrefix (see page 6299)

65.1.4.3.4 TAdapterActionsListHelper.GetAddFieldsDlgCaption Method

Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

The IDE calls GetAddFieldsDlgCaption to label the dialog that lists available child Web items. Because TAdapterActionsListHelper (see page 6294) is specialized for controls that contain adapter action buttons, it returns the string

'Add Commands'.

See Also

GetAddAllFieldsItem (see page 6298)

GetAddFieldsItem (see page 6299)

GetNewFieldPrefix (see page 6299)

65.1.4.3.5 TAdapterActionsListHelper.GetAddFieldsItem Method

Returns the string that the IDE uses for the command to display the Add Commands dialog.

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

The IDE calls GetAddFieldsItem to customize the label of the command that brings up a dialog where you can select child Web items to add. Because TAdapterActionsListHelper (see page 6294) is specialized for controls that contain adapter action buttons, it returns the string 'Add Commands ...'.

See Also

GetAddFieldsDlgCaption (see page 6298)

GetAddAllFieldsItem (see page 6298)

GetNewFieldPrefix (see page 6299)

65.1.4.3.6 TAdapterActionsListHelper.GetNewFieldPrefix Method

Returns the string that is used to start the names of all automatically-generated Web items.

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

When the IDE adds new child Web items, it generates names of the components it adds. Because TAdapterActionsListHelper (see page 6294) is specialized for controls that contain adapter action buttons, the names of those TAdapterActionButton (see page 6286) components are generated by prepending the string that this method returns to the name of the action the button executes.

GetNewFieldPrefix returns the string 'Cmd'.

See Also

GetAddFieldsDlgCaption (see page 6298)

GetAddAllFieldsItem (see page 6298)

GetAddFieldsItem (see page 6299)

65.1.4.3.7 TAdapterActionsListHelper.GetVisibleActions Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

GetVisibleActions returns the list of child Web items that generate HTML controls for the associated component. Each element in this list is an adapter action button (TAdapterActionButton (see page 6286) object).

If you have explicitly added child Web items, the returned list contains the child Web items you added. Otherwise, if the AddDefaultActions (see page 6301) property is true, the list contains adapter action buttons for the default actions of the adapter. If AddDefaultActions (see page 6301) is false, the list is empty.

See Also

FindAction (see page 6297)

GetActionsList (see page 6297)

IsActionInUse (see page 6300)

TAdapterActionButton (see page 6286)

AddDefaultActions (see page 6301)

65.1.4.3.8 TAdapterActionsListHelper.IsActionInUse Method

Indicates whether there is a child Web item that corresponds to a specified action.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

IsActionInUse indicates whether the associated component already has a child Web item that executes a specified adapter action.

AName is the name of the adapter action.

IsActionInUse returns true if there is a child Web item that executes the specified action, false if there is no child Web item for the action.

See Also

FindAction (see page 6297)

GetActionsList (see page 6297)

GetVisibleActions (see page 6300)

ActionName (see page 6405)

65.1.4.4 TAdapterActionsListHelper Properties

The properties of the TAdapterActionsListHelper class are listed here.

Public Properties

	Name	Description
	AddDefaultActions (see page 6301)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

65.1.4.4.1 TAdapterActionsListHelper.AddDefaultActions Property

Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

Pascal

```
property AddDefaultActions: Boolean;
```

C++

```
__property Boolean AddDefaultActions;
```

Description

AddDefaultCommands determines what controls appear in the generated set of HTML controls if you do not explicitly add any adapter action buttons to the WebFieldControls property of the component that uses this TAdapterActionsListHelper (see page 6294). When AddDefaultCommands is true (the default), the associated component automatically adds controls for every default action of its adapter when you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the associated component only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5226)

AddDefaultCommands (see page 6416)

AddDefaultCommands (see page 6426)

65.1.5 TAdapterCommandColumn Class

TAdapterCommandColumn generates a column of action controls in an adapter grid.

Class Hierarchy

Pascal

```
TAdapterCommandColumn = class(TCustomAdapterCommandColumn);
```

C++

```
class TAdapterCommandColumn : public TCustomAdapterCommandColumn;
```

File

WebForm

Description

Add TAdapterCommandColumn as a child Web item to an adapter grid (TAdapterGrid (see page 6386)). TAdapterCommandColumn adds a column to the grid that contains controls that execute adapter actions. Specify the controls in the cells of this column by adding adapter action buttons (TAdapterActionButton (see page 6286) objects) as child Web items of the TAdapterCommandColumn object. Each cell in the column contains all the adapter action buttons you add as child Web items. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 6304), WebFieldControls, and VisibleCommands (see page 6420) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 6304) and CaptionAttributes (see page 6305) properties, which determine the appearance of the label at the top of the column.

The Align (see page 6304), VAlign (see page 6307), and DisplayColumns (see page 6306) properties, which determine the layout of controls inside each cell.

The Style (see page 6306) and StyleRule (see page 6306) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterGrid (see page 6386)

TAdapterEditColumn (see page 6355)

TAdapterDisplayColumn (see page 6333)

TAdapterActionButton (see page 6286)

65.1.5.1 TAdapterCommandColumn Members

The following tables list the members exposed by TAdapterCommandColumn.

Protected Methods

	Name	Description
	FormatColumnData (see page 6412)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6413)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6413)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6413)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6414)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6414)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6415)	Responds to changes that affect the associated adapter.

Protected Properties

	Name	Description
	AdapterActionsListHelper (see page 6416)	Identifies the helper object that enables this component to work with the editor at design time.

Public Methods

	Name	Description
	Create (see page 6411)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6409).
	Destroy (see page 6412)	Frees an instance of TCustomAdapterCommandColumn (see page 6409).
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.

Public Properties

	Name	Description
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
	VisibleCommands (see page 6420)	Lists the child Web items that generate controls that appear in cells of the column.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6304)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6304)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6304)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6305)	Specifies the attributes of the Caption property.
	Custom (see page 6305)	Represents a tag option and its value.
	DisplayColumns (see page 6306)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6306)	Adds style attributes to the generated caption.
	StyleRule (see page 6306)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6307)	Specifies the vertical alignment of the action control buttons within the column.

65.1.5.2 TAdapterCommandColumn Properties

The properties of the TAdapterCommandColumn class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6304)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6304)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6304)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6305)	Specifies the attributes of the Caption property.
	Custom (see page 6305)	Represents a tag option and its value.
	DisplayColumns (see page 6306)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6306)	Adds style attributes to the generated caption.

	StyleRule (see page 6306)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6307)	Specifies the vertical alignment of the action control buttons within the column.

65.1.5.2.1 TAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5226)

VisibleCommands (see page 6420)

65.1.5.2.2 TAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6420)

DisplayColumns (see page 6418)

65.1.5.2.3 TAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 6418)

65.1.5.2.4 TAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 6417)

65.1.5.2.5 TAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 6419)

StyleRule (see page 6419)

65.1.5.2.6 TAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one adapter action button. TCustomAdapterCommandColumn places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls

65.1.5.2.7 TAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes property instead.

See Also

Custom (see page 6418)

StyleRule (see page 6419)

CaptionAttributes (see page 6418)

65.1.5.2.8 TAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

See Also

Custom (see page 6418)

Style (see page 6419)

Styles (see page 1097)

StylesFile (see page 1098)

65.1.5.2.9 TAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

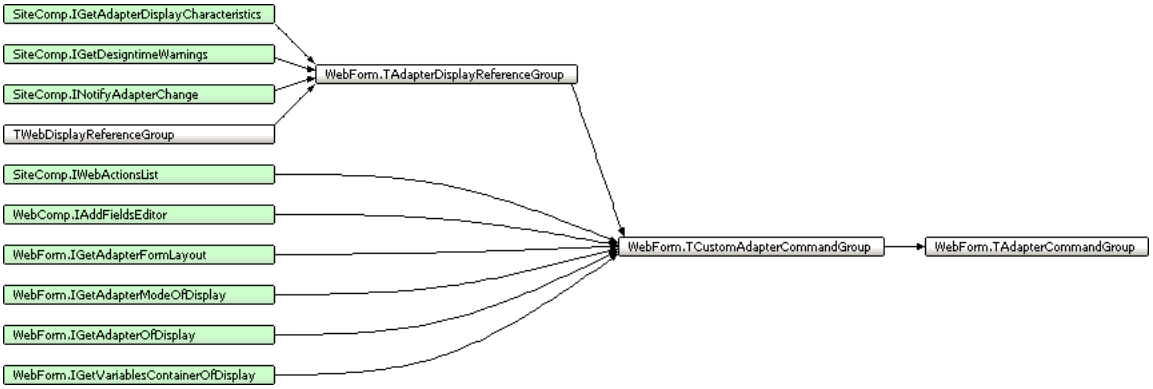
Align (see page 6417)

DisplayColumns (see page 6418)

65.1.6 TAdapterCommandGroup Class

TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.

Class Hierarchy



Pascal

```
TAdapterCommandGroup = class(TCustomAdapterCommandGroup);
```

C++

```
class TAdapterCommandGroup : public TCustomAdapterCommandGroup;
```

File

WebForm

Description

TAdapterCommandGroup is added to a TAdapterPageProducer to cause the page producer to generate a group of HTML elements that executes adapter actions. First add a TAdapterForm (see page 6373) to the TAdapterPageProducer and then add TAdapterCommandGroup as a child of TAdapterForm (see page 6373). TAdapterCommandGroup does not reference an adapter directly. Instead, it references a component that references an adapter. Typically, TAdapterCommandGroup references either TAdapterGrid (see page 6386) or TAdapterFieldGroup (see page 6367). TAdapterCommandGroup contains child components, such as TAdapterActionButton (see page 6286), that are responsible for rendering individual HTML elements, such as buttons. By default, TAdapterCommandGroup will automatically add one child component for each action in the adapter.

See Also

TAdapterDisplayReferenceGroup (see page 6349)

TCustomAdapterCommandGroup (see page 6421)

TAdapterForm (see page 6373)

TAdapterFieldGroup (see page 6367)

TAdapterGrid (see page 6386)

TAdapterActionButton (see page 6286)

65.1.6.1 TAdapterCommandGroup Members

The following tables list the members exposed by TAdapterCommandGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6352)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6352)	Responds when the DisplayComponent (see page 6355) property changes.

	GetAdapter (see page 6424)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 6353)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 6424)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6424)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6424)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetDesigntimeWarnings (see page 6353)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
V	ImplCanAddClass (see page 6425)	Checks the inheritance of the given component.
V	ImplContent (see page 6425)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6376) component.
	NotifyAdapterChange (see page 6353)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6354)	Responds when the associated display component changes.
V	SetWebDisplayComponent (see page 6354)	Sets the value of the DisplayComponent (see page 6355) property.

Protected Properties

	Name	Description
R	Adapter (see page 6426)	Provides read-only access to the Adapter that this command group uses.
R	AdapterActionsListHelper (see page 6426)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Methods

	Name	Description
V	Create (see page 6351)	Creates an instance of TWebDisplayReferenceGroup.
V	Create (see page 6423)	Creator for TCustomAdapterCommandGroup (see page 6421).
V	Destroy (see page 6351)	Frees an instance of TAdapterDisplayReferenceGroup (see page 6349).
V	Destroy (see page 6423)	Frees an instance of TAdapterDisplayReferenceGroup.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.

Public Properties

	Name	Description
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 6427)	Provides read-only access to the TWebComponentList (see page 5999) that stores the list of visible components.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6310)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

	Custom (see page 6310)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DisplayComponent (see page 6311)	Associates the group with a component that generates the display controls.
	Style (see page 6311)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6311)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

65.1.6.2 TAdapterCommandGroup Properties

The properties of the TAdapterCommandGroup class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6310)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6310)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DisplayComponent (see page 6311)	Associates the group with a component that generates the display controls.
	Style (see page 6311)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6311)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

65.1.6.2.1 TAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

65.1.6.2.2 TAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as Valign="top."

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as Valign="top." The content of this string will be written as it is when generating HTML.

65.1.6.2.3 TAdapterCommandGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid or TAdapterFieldGroup.

TAdapterDisplayReferenceGroup descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

65.1.6.2.4 TAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

65.1.6.2.5 TAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

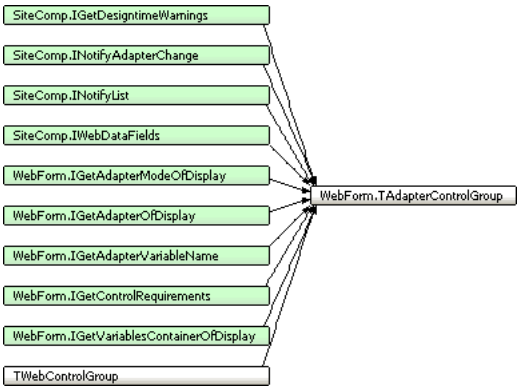
Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

65.1.7 TAdapterControlGroup Class

TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6339) or TAdapterDisplayColumn (see page 6333).

Class Hierarchy



Pascal

```
TAdapterControlGroup = class(TWebControlGroup, IWebDataFields, INotifyAdapterChange,
IGetAdapterVariableName, IGetAdapterOfDisplay, INotifyList, IGetVariablesContainerOfDisplay,
IGetDesigntimeWarnings, IGetAdapterModeOfDisplay, IGetControlRequirements);
```

C++

```
class TAdapterControlGroup : public TWebControlGroup, public IWebDataFields, public
INotifyAdapterChange, public IGetAdapterVariableName, public IGetAdapterOfDisplay, public
INotifyList, public IGetVariablesContainerOfDisplay, public IGetDesigntimeWarnings, public
IGetAdapterModeOfDisplay, public IGetControlRequirements;
```

File

WebForm

Description

Do not create instances of TAdapterControlGroup. Instead, use a descendant of TAdapterControlGroup that manages particular types of display field components. These descendants include TAdapterFieldGroup (see page 6367) and TAdapterGrid (see page 6386).

Descendants of TAdapterControlGroup are children of TAdapterPageProducer form components. Use the Web page editor to add TAdapterControlGroup descendants to an adapter form when building an HTML form.

See Also

TAdapterFieldGroup (see page 6367)

TAdapterGrid (see page 6386)

TAdapterPageProducer (see page 1076)

TAdapterForm (see page 6373)

TAdapter (see page 5179)

65.1.7.1 TAdapterControlGroup Members

The following tables list the members exposed by TAdapterControlGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.
	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

Public Methods

	Name	Description
	Create (see page 6313)	Creates and initializes an instance of TAdapterControlGroup (see page 6312).
	Destroy (see page 6314)	Frees an instance of TAdapterControlGroup (see page 6312).
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

Public Properties

	Name	Description
	Adapter (see page 6322)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6323)	Specifies the mode of the adapter that the controls in the group support.
	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6323)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

65.1.7.2 TAdapterControlGroup.Create Constructor

Creates and initializes an instance of TAdapterControlGroup (see page 6312).

Pascal

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

C++

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

Description

Call Create to instantiate TAdapterControlGroup (see page 6312) at runtime. TAdapterControlGroup (see page 6312) objects added at design time are instantiated automatically.

See Also

Owner

65.1.7.3 TAdapterControlGroup.Destroy Destructor

Frees an instance of TAdapterControlGroup (see page 6312).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterControlGroup();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.7.4 TAdapterControlGroup Methods

The methods of the TAdapterControlGroup class are listed here.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.

	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

65.1.7.4.1 TAdapterControlGroup.AdapterChange Method

Responds when the adapter associated with this control group changes.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

AdapterChange is called automatically when the value of the Adapter (see page 6322) property changes, or when the application makes changes to the adapter that is the value of the Adapter (see page 6322) property.

AdapterChange performs the following tasks to respond to the change:

Updates the list of default adapter display field components that is the value of VisibleFields (see page 6324) when you have not explicitly assigned child Web items.

Notifies all of the child Web items listed in the WebFieldControls property that the adapter has changed.

Notifies any objects that registered interest by calling the AddNotify (see page 6315) method that the adapter has changed.

See Also

Adapter (see page 6322)

AddNotify (see page 6315)

WebFieldControls

VisibleFields (see page 6324)

65.1.7.4.2 TAdapterControlGroup.AddNotify Method

Registers interest so that the adapter control group notifies an object of changes to its adapter.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call AddNotify to register an interest in when the control group's adapter changes. AddNotify adds the specified object to an internal list of interested objects. When the Adapter (see page 6322) property changes, it calls every object in this internal list to notify it of the change.

There is no need for the child Web items listed in the WebFieldControls property to call AddNotify. These objects are notified

automatically.

ANotify is the object that registers an interest in change notifications.

See Also

AdapterChange (see page 6315)

RemoveNotify (see page 6321)

Adapter (see page 6322)

65.1.7.4.3 TAdapterControlGroup.CreateDefaultFields Method

Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.

Pascal

```
procedure CreateDefaultFields;
```

C++

```
__fastcall CreateDefaultFields();
```

Description

TAdapterControlGroup (see page 6312) calls CreateDefaultFields internally when it needs to generate child Web items for the fields of the adapter that the Adapter (see page 6322) property specifies. The resulting list becomes the value of the VisibleFields (see page 6324) property if you have not explicitly added any child Web items to the WebFieldControls property and AddDefaultFields (see page 6323) is true.

See Also

VisibleFields (see page 6324)

Adapter (see page 6322)

AddDefaultFields (see page 6323)

WebFieldControls

65.1.7.4.4 TAdapterControlGroup.FindField Method

Returns the child Web item that represents the value of a specified adapter field.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

FindField searches the list of child Web items for one that represents the specified adapter field. If it finds a Web item that represents the specified field, it returns that adapter display field component. Otherwise, FindField returns nil (Delphi) or NULL (C++).

AName is the name of a field that the adapter control group's adapter contains.

See Also

VisibleFields (see page 6324)

GetFieldsList (see page 6319)

Adapter (see page 6322)

65.1.7.4.5 TAdapterControlGroup.GetAdapter Method

Returns the value of the Adapter (see page 6322) property.

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

GetAdapter is the protected read implementation of the Adapter (see page 6322) property.

See Also

Adapter (see page 6322)

65.1.7.4.6 TAdapterControlGroup.GetAdapterModeOfAdapter Method

Returns the current mode of the control group's adapter.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

GetAdapterModeOfAdapter returns the current mode of the adapter that the Adapter (see page 6322) property specifies. If the adapter's mode is not currently set, GetAdapterModeOfAdapter returns the value of the AdapterMode (see page 6323) property.

See Also

AdapterMode (see page 6323)

Adapter (see page 6322)

GetAdapterModeOfDisplay (see page 6317)

65.1.7.4.7 TAdapterControlGroup.GetAdapterModeOfDisplay Method

Returns the value of the AdapterMode (see page 6323) property.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

GetAdapterModeOfDisplay returns the value of the AdapterMode (see page 6323) property. This is the value that influences the type and behavior of the HTML controls that child Web items generate. It may differ from the current mode of the control

group's adapter, which the GetAdapterModeOfAdapter (see page 6317) method returns.

See Also

AdapterMode (see page 6323)

GetAdapterModeOfAdapter (see page 6317)

65.1.7.4.8 TAdapterControlGroup.GetAdapterVariableName Method

Returns the variable name that refers to the control group's adapter in server-side script.

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

TAdapterControlGroup (see page 6312) uses GetAdapterVariableName internally when generating server-side script that refers to its adapter.

See Also

Adapter (see page 6322)

65.1.7.4.9 TAdapterControlGroup.GetControlRequirements Method

Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The adapter form object that includes this control group as a child Web item calls GetControlRequirements to obtain declarations for the variables and functions that the control group uses to assemble HTML controls for its adapter's fields.

ARequirements is the control requirements object that collects a list of required variables and functions.

GetControlRequirements simply calls the ImplGetControlRequirements (see page 6320) method, which adds a variable that represents the control group's adapter to ARequirements if it is not already there.

See Also

ImplGetControlRequirements (see page 6320)

TAdapterForm (see page 6373)

GetAdapterVariableName (see page 6318)

65.1.7.4.10 TAdapterControlGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the adapter control group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the adapter control group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

If the Adapter (see page 6322) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds a reference to Adapter (see page 6322) to the design-time warnings object so that it can obtain any warning messages that the adapter wants to add.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

See Also

Adapter (see page 6322)

TAdapterPageProducer (see page 1076)

65.1.7.4.11 TAdapterControlGroup.GetFieldsList Method

Fills a string list with the names of all of the associated adapter's fields.

Pascal

```
procedure GetFieldsList(List: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings List);
```

Description

The CreateDefaultFields (see page 6316) method calls GetFieldsList to obtain a list of adapter fields for which it needs to create default child Web items. GetFieldsList obtains a list of the names of its fields from the adapter, and adds them to the specified string list. For any field that is a "default" field (as defined by the adapter), the string in the list is associated with the class of the child Web item that generates a control to represent that field.

List is a string list to fill with the names of (and Web item classes for) the adapter's default fields.

See Also

CreateDefaultFields (see page 6316)

Adapter (see page 6322)

65.1.7.4.12 TAdapterControlGroup.GetVisibleFields Method

Returns the value of the VisibleFields (see page 6324) property.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

GetVisibleFields is the protected implementation of the VisibleFields (see page 6324) property.

See Also

VisibleFields (see page 6324)

65.1.7.4.13 TAdapterControlGroup.ImplGetControlRequirements Method

Provides the underlying implementation for the GetControlRequirements (see page 6318) method.

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The GetControlRequirements (see page 6318) method calls ImplGetControlRequirements to add the variables and functions the control group needs to have declared to the control requirements object. This method adds the requirement of a variable with the name that the GetAdapterVariableName (see page 6318) method returns to represent the adapter whose fields the control group displays.

ARequirements is the control requirements object that collects a list of required variables and functions.

See Also

GetControlRequirements (see page 6318)

GetAdapterVariableName (see page 6318)

65.1.7.4.14 TAdapterControlGroup.IsFieldInUse Method

Indicates whether one of the adapter control group's child Web items represents a specified adapter field.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

IsFieldInUse returns true if one of its child Web items represents the adapter field with the specified name, false otherwise.

AName is the name of a field that the adapter control group's adapter contains.

See Also

FindField (see page 6316)

VisibleFields (see page 6324)

Adapter (see page 6322)

65.1.7.4.15 TAdapterControlGroup.Notification Method

Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

In TAdapterControlGroup (see page 6312), Notification checks whether AComponent is the value of the Adapter (see page 6322) property. If so, and Operation is opRemove, Notification sets the Adapter (see page 6322) property to nil (Delphi) or NULL (C++).

See Also

Adapter (see page 6322)

65.1.7.4.16 TAdapterControlGroup.NotifyAdapterChange Method

Responds when the control group's adapter notifies it of a change.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the application makes changes to the adapter that is the value of the Adapter (see page 6322) property. This method calls the AdapterChange (see page 6315) method, which updates the list of default fields to be represented, and forwards the notification on to other Web items that may need to adjust themselves.

Sender is the component that sent the notification.

See Also

Adapter (see page 6322)

AdapterChange (see page 6315)

65.1.7.4.17 TAdapterControlGroup.RemoveNotify Method

Unregisters interest in changes to the control group's adapter.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call RemoveNotify to unregister an interest in when the control group's adapter changes. The object must have previously registered interest by a call to the AddNotify (see page 6315) method. RemoveNotify removes the specified object from the internal list of interested objects.

ANotify is the object that had previously registered an interest in change notifications.

See Also

AdapterChange (see page 6315)

AddNotify (see page 6315)

65.1.7.4.18 TAdapterControlGroup.SetAdapter Method

Sets the value of the Adapter (see page 6322) property.

Pascal

```
procedure SetAdapter(const Value: TComponent);
```

C++

```
__fastcall SetAdapter(const TComponent * Value);
```

Description

SetAdapter is the protected write implementaiton of the Adapter (see page 6322) property.

See Also

Adapter (see page 6322)

65.1.7.5 TAdapterControlGroup Properties

The properties of the TAdapterControlGroup class are listed here.

Public Properties

	Name	Description
	Adapter (see page 6322)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6323)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6323)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

65.1.7.5.1 TAdapterControlGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5536)

GetFieldsList (see page 6319)

AdapterMode (see page 6323)

WebFieldControls

65.1.7.5.2 TAdapterControlGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert (see page 3698), Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter (see page 6322) property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6322)

WebFieldControls

65.1.7.5.3 TAdapterControlGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter (see page 6322) page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter (see page 6322) property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no

field values.

See Also

VisibleFields (see page 6324)

WebFieldControls

Adapter (see page 6322)

65.1.7.5.4 TAdapterControlGroup.VisibleFields Property

Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Read (see page 3727) VisibleFields to determine the adapter display field components that generate HTML controls for this adapter control group. If you have explicitly added child Web items to the control group, or if the AddDefaultFields (see page 6323) property is false, VisibleFields is the same as the value of the WebFieldControls property. If you have not added any child Web items and AddDefaultFields (see page 6323) is true, VisibleFields is a list of adapter display field components that generate controls for the default fields of the control group's adapter.

See Also

WebFieldControls

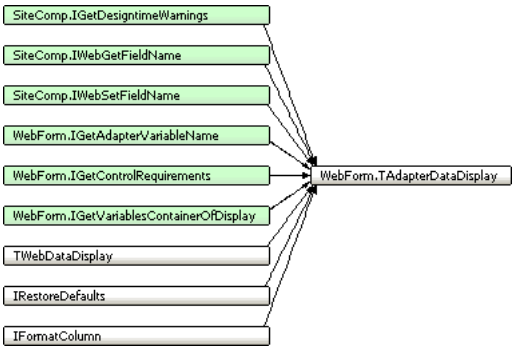
AddDefaultFields (see page 6323)

Adapter (see page 6322)

65.1.8 TAdapterDataDisplay Class

TAdapterDataDisplay is the base class for components that display adapter fields as form fields.

Class Hierarchy



Pascal

```
TAdapterDataDisplay = class(TWebDataDisplay, IWebGetFieldName, IWebSetFieldName,
```

```
  IGetAdapterVariableName, IRestoreDefaults, IGetControlRequirements,
  IGetVariablesContainerOfDisplay, IGetDesigntimeWarnings, IFormatColumn);
```

C++

```
  class TAdapterDataDisplay : public TWebDataDisplay, public IWebGetFieldName, public
  IWebSetFieldName, public IGetAdapterVariableName, public IRestoreDefaults, public
  IGetControlRequirements, public IGetVariablesContainerOfDisplay, public
  IGetDesigntimeWarnings, public IFormatColumn;
```

File

WebForm

Description

TAdapterDataDisplay is the base class for components that display adapter fields as HTML form fields. Its descendants include the following classes:

TAdapterDisplayField (see page 6339), which supports free-form display and editing of adapter field data.

TAdapterDisplayColumn (see page 6333), which supports grid display of adapter field data.

TAdapterEditColumn (see page 6355), which supports grid editing of adapter field data.

See Also

TAdapterDisplayField (see page 6339)

TAdapterDisplayColumn (see page 6333)

TAdapterEditColumn (see page 6355)

65.1.8.1 TAdapterDataDisplay Members

The following tables list the members exposed by TAdapterDataDisplay.

Protected Methods

	Name	Description
	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6326)	Creates and initializes an instance of TWebDataDisplay.

Public Properties

	Name	Description
	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6332)	Specifies the horizontal alignment of the component relative to its containing cell.
	FieldName (see page 6332)	Specifies the name of the associated dataset field.
	HideOptions (see page 6332)	When field should be hidden.
	VAlign (see page 6332)	Specifies the vertical alignment of the generated display control.

65.1.8.2 TAdapterDataDisplay.Create Constructor

Creates and initializes an instance of TWebDataDisplay.

Pascal

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

C++

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

Description

Call Create to instantiate TWebDataDisplay at runtime. TWebDataDisplay descendants added at design time are instantiated automatically.

65.1.8.3 TAdapterDataDisplay Methods

The methods of the TAdapterDataDisplay class are listed here.

Protected Methods

	Name	Description
	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

65.1.8.3.1 TAdapterDataDisplay.FieldConditions Method

Pascal

```
function FieldConditions(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString FieldConditions(TWebContentOptions Options);
```

Description

This is FieldConditions, a member of class TAdapterDataDisplay.

65.1.8.3.2 TAdapterDataDisplay.FindAssociatedField Method

Pascal

```
function FindAssociatedField: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedField();
```

Description

This is FindAssociatedField, a member of class TAdapterDataDisplay.

65.1.8.3.3 TAdapterDataDisplay.FormatCaption Method

Pascal

```
function FormatCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString FormatCaption();
```

Description

This is FormatCaption, a member of class TAdapterDataDisplay.

65.1.8.3.4 TAdapterDataDisplay.FormatColumnData Method

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnData(const UnicodeString Content, TWebContentOptions
Options);
```

Description

This is FormatColumnData, a member of class TAdapterDataDisplay.

65.1.8.3.5 TAdapterDataDisplay.FormatColumnHeading Method

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnHeading(TWebContentOptions Options);
```

Description

This is FormatColumnHeading, a member of class TAdapterDataDisplay.

65.1.8.3.6 TAdapterDataDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.

65.1.8.3.7 TAdapterDataDisplay.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TAdapterDataDisplay.

65.1.8.3.8 TAdapterDataDisplay.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption

65.1.8.3.9 TAdapterDataDisplay.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TAdapterDataDisplay.

65.1.8.3.10 TAdapterDataDisplay.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.

65.1.8.3.11 TAdapterDataDisplay.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class TAdapterDataDisplay.

65.1.8.3.12 TAdapterDataDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class TAdapterDataDisplay.

65.1.8.3.13 TAdapterDataDisplay.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

Caption

ILayoutWebContent (see page 6057)

65.1.8.3.14 TAdapterDataDisplay.ImplFormatColumnData Method

Pascal

```
function ImplFormatColumnData(const Content: string; Options: TWebContentOptions): string;
virtual;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnData(const UnicodeString Content,
TWebContentOptions Options);
```

Description

This is ImplFormatColumnData, a member of class TAdapterDataDisplay.

65.1.8.3.15 TAdapterDataDisplay.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.

65.1.8.3.16 TAdapterDataDisplay.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.

65.1.8.3.17 TAdapterDataDisplay.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.

65.1.8.3.18 TAdapterDataDisplay.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TAdapterDataDisplay.

65.1.8.3.19 TAdapterDataDisplay.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(Value: string); override;
```

C++

```
virtual __fastcall SetCaption(UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption

65.1.8.3.20 TAdapterDataDisplay.SetFieldName Method

Pascal

```
procedure SetFieldName(const Value: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString Value);
```

Description

This is SetFieldName, a member of class TAdapterDataDisplay.

65.1.8.4 TAdapterDataDisplay Properties

The properties of the TAdapterDataDisplay class are listed here.

Public Properties

	Name	Description
	Align (see page 6332)	Specifies the horizontal alignment of the component relative to its containing cell.

	FieldName (see page 6332)	Specifies the name of the associated dataset field.
	HideOptions (see page 6332)	When field should be hidden.
	VAlign (see page 6332)	Specifies the vertical alignment of the generated display control.

65.1.8.4.1 TAdapterDataDisplay.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6332)

65.1.8.4.2 TAdapterDataDisplay.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

65.1.8.4.3 TAdapterDataDisplay.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

65.1.8.4.4 TAdapterDataDisplay.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

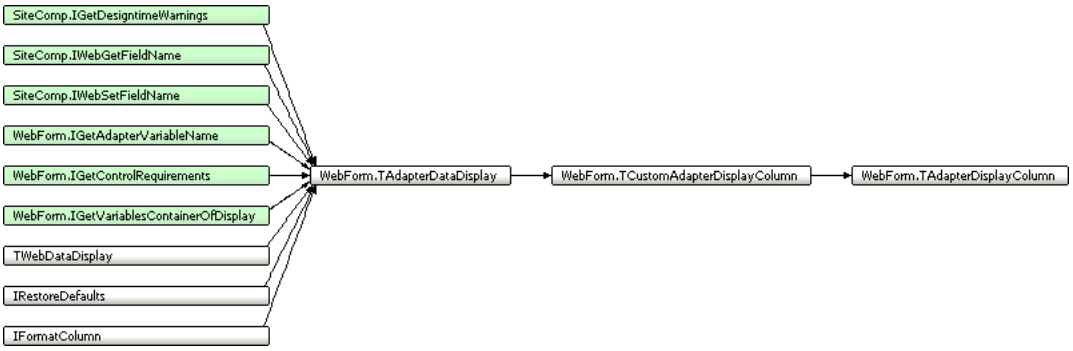
See Also

Align (see page 6332)

65.1.9 TAdapterDisplayColumn Class

TAdapterDisplayColumn displays an Adapter field in a grid.

Class Hierarchy



Pascal

```
TAdapterDisplayColumn = class(TCustomAdapterDisplayColumn);
```

C++

```
class TAdapterDisplayColumn : public TCustomAdapterDisplayColumn;
```

File

WebForm

Description

Add TAdapterDisplayColumn as a child of a TAdapterGrid (see page 6386) to display an Adapter field. The TAdapterGrid (see page 6386) specifies the Adapter used by TAdapterDisplayColumn.

Use the properties and methods of TAdapterDisplayColumn to:

Specify the Adapter field whose value the component displays.

Specify a caption for the column.

Specify when the column should be hidden.

Customize the appearance of the column.

TAdapterDisplayColumn supports multiple display presentation: text, list, and image. The display type can be set explicitly or TAdapterDisplayColumn will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayColumn is a minimal implementation of TCustomAdapterDisplayColumn (see page 6428). To extend this component, create your own concrete descendant of TCustomAdapterDisplayColumn (see page 6428).

See Also

TCustomAdapterDisplayColumn (see page 6428)

65.1.9.1 TAdapterDisplayColumn Members

The following tables list the members exposed by TAdapterDisplayColumn.

Protected Methods

	Name	Description
	CanAddClass (see page 6430)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ControlContent (see page 6431)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.
V	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 6431)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6431)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6431)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplCanAddClass (see page 6432)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6432)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
V	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6432)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
V	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 6432)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
V	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6326)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 6430)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6428).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.

	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
V	IsColumn (see page 6432)	Indicates whether the component generates the controls that appear in the column of a data grid.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6335)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6336)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6336)	Customizes the appearance of the generated control's label.
	Custom (see page 6336)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 6337)	Sets the display type for the column.
	FieldName (see page 6337)	Specifies the name of the associated dataset field.
	HideOptions (see page 6338)	When field should be hidden.
	ImageHeight (see page 6338)	Specifies the height of the image in the column.
	ImageWidth (see page 6338)	Specifies the width of the image in the column.
	Style (see page 6338)	Adds style attributes to the generated display control.
	StyleRule (see page 6339)	This is StyleRule, a member of class TAdapterDisplayColumn.
	VAlign (see page 6339)	Specifies the vertical alignment of the generated display control.

65.1.9.2 TAdapterDisplayColumn Properties

The properties of the TAdapterDisplayColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 6335)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6336)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6336)	Customizes the appearance of the generated control's label.
	Custom (see page 6336)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 6337)	Sets the display type for the column.
	FieldName (see page 6337)	Specifies the name of the associated dataset field.
	HideOptions (see page 6338)	When field should be hidden.
	ImageHeight (see page 6338)	Specifies the height of the image in the column.
	ImageWidth (see page 6338)	Specifies the width of the image in the column.
	Style (see page 6338)	Adds style attributes to the generated display control.
	StyleRule (see page 6339)	This is StyleRule, a member of class TAdapterDisplayColumn.
	VAlign (see page 6339)	Specifies the vertical alignment of the generated display control.

65.1.9.2.1 TAdapterDisplayColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6332)

65.1.9.2.2 TAdapterDisplayColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

65.1.9.2.3 TAdapterDisplayColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

65.1.9.2.4 TAdapterDisplayColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

stylerule

CaptionAttributes

65.1.9.2.5 TAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 6433)

ImageWidth (see page 6434)

65.1.9.2.6 TAdapterDisplayColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

65.1.9.2.7 TAdapterDisplayColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

65.1.9.2.8 TAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 6434)

65.1.9.2.9 TAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in pixels.

See Also

ImageHeight (see page 6433)

65.1.9.2.10 TAdapterDisplayColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

stylerule

CaptionAttributes

Setting Web Item Properties

65.1.9.2.11 TAdapterDisplayColumn.StyleRule Property

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

This is StyleRule, a member of class TAdapterDisplayColumn.

65.1.9.2.12 TAdapterDisplayColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

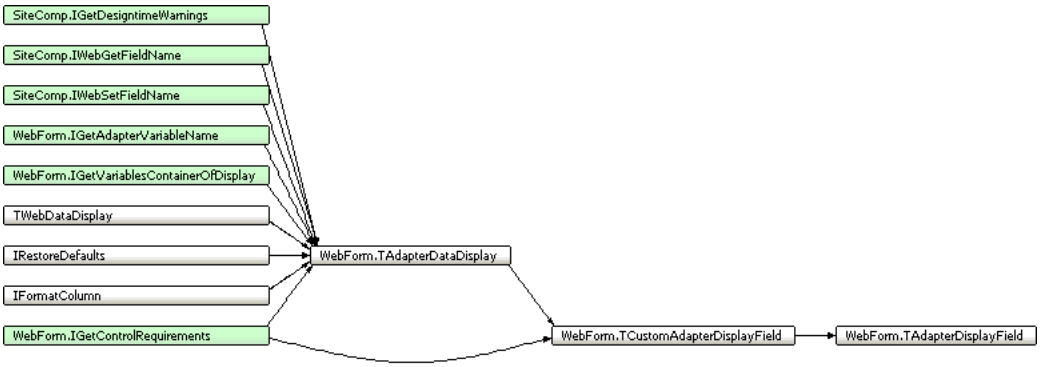
See Also

Align (see page 6332)

65.1.10 TAdapterDisplayField Class

TAdapterDisplayField displays an Adapter field as a form field.

Class Hierarchy



Pascal

```
TAdapterDisplayField = class(TCustomAdapterDisplayField);
```

C++

```
class TAdapterDisplayField : public TCustomAdapterDisplayField;
```

File

WebForm

Description

Add TAdapterDisplayField as a child of a TAdapterFieldGroup (see page 6367) to display an Adapter field. The TAdapterFieldGroup (see page 6367) specifies the Adapter used by TAdapterDisplayField.

Use the properties and methods of TAdapterDisplayField to:

Specify the Adapter field whose value the component displays.

Specify a caption for the field.

Specify when the field should be hidden.

Customize the appearance of the column.

Specify whether the field is used for input or display only.

TAdapterDisplayField supports multiple display type: text, list, and image. The display type can be set explicitly or TAdapterDisplayField will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayField supports multiple input types: Text, Password, Select, MultipleSelect, Radio, CheckBox, TextArea, and File. The input type can be set explicitly or TAdapterDisplayField will automatically choose an input type that is appropriate for the Adapter field.

TAdapterDisplayField instances are not used in grids. TAdapterEditColumn (see page 6355) provides equivalent functionality in a grid context.

TAdapterDisplayField is a minimal implementation of TCustomAdapterDisplayField (see page 6434). To extend the features of this component, define your own descendant of TCustomAdapterDisplayField (see page 6434).

See Also

TAdapterEditColumn (see page 6355)

65.1.10.1 TAdapterDisplayField Members

The following tables list the members exposed by TAdapterDisplayField.

Protected Methods

	Name	Description
	AddAttributes (see page 6437)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6437)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
V	GetLayoutAttributes (see page 6437)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 6437)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6438)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 6438)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
V	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6326)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 6436)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6434).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6343)	Specifies the horizontal alignment of the component relative to its containing cell.

	Caption (see page 6343)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6343)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6344)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6344)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6344)	Specifies the number of columns in a display field.
	DisplayRows (see page 6345)	Specifies the number of rows in a display field.
	DisplayType (see page 6345)	Sets the display type for the column.
	DisplayWidth (see page 6345)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6346)	Specifies the name of the associated dataset field.
	HideOptions (see page 6346)	When field should be hidden.
	ImageHeight (see page 6346)	Specifies the height of the image in the column.
	ImageWidth (see page 6347)	Specifies the width of the image in the column.
	InputType (see page 6347)	Specifies the input type for the display field.
	MaxLength (see page 6347)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6348)	Indicates the number of rows that can be selected.
	Style (see page 6348)	Adds style attributes to the generated display control.
	StyleRule (see page 6348)	This is StyleRule, a member of class TAdapterDisplayField.
	TextAreaWrap (see page 6349)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6349)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6349)	Indicates the viewing state of the adapter field.

65.1.10.2 TAdapterDisplayField Properties

The properties of the TAdapterDisplayField class are listed here.

Published Properties

	Name	Description
	Align (see page 6343)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6343)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6343)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6344)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6344)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6344)	Specifies the number of columns in a display field.
	DisplayRows (see page 6345)	Specifies the number of rows in a display field.
	DisplayType (see page 6345)	Sets the display type for the column.
	DisplayWidth (see page 6345)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6346)	Specifies the name of the associated dataset field.
	HideOptions (see page 6346)	When field should be hidden.
	ImageHeight (see page 6346)	Specifies the height of the image in the column.
	ImageWidth (see page 6347)	Specifies the width of the image in the column.
	InputType (see page 6347)	Specifies the input type for the display field.
	MaxLength (see page 6347)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6348)	Indicates the number of rows that can be selected.
	Style (see page 6348)	Adds style attributes to the generated display control.
	StyleRule (see page 6348)	This is StyleRule, a member of class TAdapterDisplayField.
	TextAreaWrap (see page 6349)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6349)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6349)	Indicates the viewing state of the adapter field.

65.1.10.2.1 TAdapterDisplayField.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6332)

65.1.10.2.2 TAdapterDisplayField.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

65.1.10.2.3 TAdapterDisplayField.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

65.1.10.2.4 TAdapterDisplayField.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition;
```

C++

```
__property CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes

Caption

65.1.10.2.5 TAdapterDisplayField.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

stylerule

CaptionAttributes

65.1.10.2.6 TAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6439)

65.1.10.2.7 TAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 6439)

65.1.10.2.8 TAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

InputType (see page 6441)

65.1.10.2.9 TAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

65.1.10.2.10 TAdapterDisplayField.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

65.1.10.2.11 TAdapterDisplayField.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

65.1.10.2.12 TAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6440)

ImageWidth (see page 6440)

65.1.10.2.13 TAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 6440)

65.1.10.2.14 TAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

DisplayType (see page 6439)

65.1.10.2.15 TAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

65.1.10.2.16 TAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

65.1.10.2.17 TAdapterDisplayField.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

stylerule

CaptionAttributes

Setting Web Item Properties

65.1.10.2.18 TAdapterDisplayField.StyleRule Property

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

This is StyleRule, a member of class TAdapterDisplayField.

65.1.10.2.19 TAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 6440)

65.1.10.2.20 TAdapterDisplayField.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 6332)

65.1.10.2.21 TAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

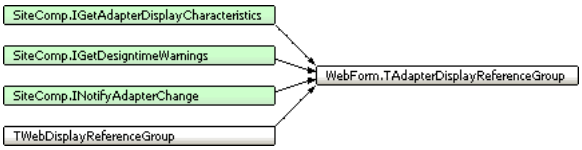
Description

Use ViewMode to specify the viewing state of the field.

65.1.11 TAdapterDisplayReferenceGroup Class

TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.

Class Hierarchy



Pascal

```
TAdapterDisplayReferenceGroup = class(TWebDisplayReferenceGroup, IGetDesigntimeWarnings,
INotifyAdapterChange, IGetAdapterDisplayCharacteristics);
```

C++

```
class TAdapterDisplayReferenceGroup : public TWebDisplayReferenceGroup, public
IGetDesigntimeWarnings, public INotifyAdapterChange, public IGetAdapterDisplayCharacteristics;
```

File

WebForm

Description

Use TAdapterDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of controls. The TAdapterDisplayReferenceGroup is associated with another component that generates HTML elements to display the data from an adapter. This component is the value of the DisplayComponent (see page 6355) property. The child Web items of TAdapterDisplayReferenceGroup generate the individual controls in the collection this class arranges. Each of those controls operates on the display element produced by the component that is the value of the DisplayComponent (see page 6355) property.

Do not create instances of TAdapterDisplayReferenceGroup. TAdapterDisplayReferenceGroup only manages the Web items that generate controls, the association to a display generator, and the logic for working with the WebSnap editors in the IDE. It does not include any special logic for working with the display generator's adapter. To add a component that organizes controls in an adapter form, use a descendant of TAdapterDisplayReferenceGroup such as TAdapterCommandGroup (see page 6307).

See Also

TAdapterCommandGroup (see page 6307)

65.1.11.1 TAdapterDisplayReferenceGroup Members

The following tables list the members exposed by TAdapterDisplayReferenceGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6352)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6352)	Responds when the DisplayComponent (see page 6355) property changes.
	GetAdapterDisplayCharacteristics (see page 6353)	Indicates the display characteristics of the HTML this component generates.
	GetDesigntimeWarnings (see page 6353)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	NotifyAdapterChange (see page 6353)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6354)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6354)	Sets the value of the DisplayComponent (see page 6355) property.

Public Methods

	Name	Description
	Create (see page 6351)	Creates an instance of TWebDisplayReferenceGroup.
	Destroy (see page 6351)	Frees an instance of TAdapterDisplayReferenceGroup (see page 6349).

Public Properties

	Name	Description
	DisplayComponent (see page 6355)	Associates the group with a component that generates the display controls.

65.1.11.2 TAdapterDisplayReferenceGroup.Create Constructor

Creates an instance of TWebDisplayReferenceGroup.

Pascal

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

C++

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

Description

Call TWebDisplayReferenceGroup to instantiate TWebDisplayReferenceGroup at runtime. TWebDisplayReferenceGroup components added at design time are instantiated automatically.

65.1.11.3 TAdapterDisplayReferenceGroup.Destroy Destructor

Frees an instance of TAdapterDisplayReferenceGroup (see page 6349).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDisplayReferenceGroup();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.11.4 TAdapterDisplayReferenceGroup Methods

The methods of the TAdapterDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	AdapterChange (see page 6352)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6352)	Responds when the DisplayComponent (see page 6355) property changes.
	GetAdapterDisplayCharacteristics (see page 6353)	Indicates the display characteristics of the HTML this component generates.
	GetDesigntimeWarnings (see page 6353)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	NotifyAdapterChange (see page 6353)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6354)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6354)	Sets the value of the DisplayComponent (see page 6355) property.

65.1.11.4.1 TAdapterDisplayReferenceGroup.AdapterChange Method

Responds to changes in the adapter on which the controls in this group operate.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

TAdapterDisplayReferenceGroup (see page 6349) calls AdapterChange internally whenever the adapter associated with its display component changes. This happens in response to the following occurrences:

The component specified by DisplayComponent (see page 6355) notifies this one that it has changed.

Another component notifies this one of a change to the adapter by calling the NotifyAdapterChange (see page 6353) method.

AdapterChange responds to the change by regenerating its list of default child Web items and passing the notification on to every child Web item in its WebFieldControls property list.

See Also

NotifyAdapterChange (see page 6353)

DisplayComponent (see page 6355)

DisplayComponentChange (see page 6352)

WebFieldControls

65.1.11.4.2 TAdapterDisplayReferenceGroup.DisplayComponentChange Method

Responds when the DisplayComponent (see page 6355) property changes.

Pascal

```
procedure DisplayComponentChange; virtual;
```

C++

```
virtual __fastcall DisplayComponentChange();
```

Description

The component specified by the DisplayComponent (see page 6355) property calls DisplayComponentChange when its properties change. This allows the TAdapterDisplayReferenceGroup (see page 6349) object to update itself to reflect any changes.

In TAdapterDisplayReferenceGroup (see page 6349), DisplayComponentChange calls the AdapterChange (see page 6352) method, to ensure that its Web items are up-to-date in case the change was to the associated adapter.

See Also

AdapterChange (see page 6352)

DisplayComponent (see page 6355)

65.1.11.4.3 TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics Method

Indicates the display characteristics of the HTML this component generates.

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

Adapter actions call GetAdapterDisplayCharacteristics from their IsDefaultAction methods to determine whether their function should be included in the adapter's default set of actions. If the adapter action is appropriate to operate on an HTML form with the indicated characteristics, the adapter action returns true from its IsDefaultAction method.

In TAdapterDisplayReferenceGroup (see page 6349), GetAdapterDisplayCharacteristics delegates this task by fetching the display characteristics from the associated display component.

See Also

IsDefaultAction (see page 5459)

DisplayComponent (see page 6355)

65.1.11.4.4 TAdapterDisplayReferenceGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the reference group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the display reference group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DisplayComponent (see page 6355) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds the display component to the list of objects that can supply design-time warnings.

See Also

DisplayComponent (see page 6355)

TAdapterPageProducer (see page 1076)

65.1.11.4.5 TAdapterDisplayReferenceGroup.NotifyAdapterChange Method

Responds when other components notify this one of changes to the adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

Other components in the WebSnap application call NotifyAdapterChange when they detect possible changes to the adapter on which this group's controls operate.

Sender is the component that detected the possible change and sent the notification.

NotifyAdapterChange calls the AdapterChange (see page 6352) method, which responds to the notification.

See Also

AdapterChange (see page 6352)

DisplayComponent (see page 6355)

65.1.11.4.6 TAdapterDisplayReferenceGroup.OnDisplayComponentChange Method

Responds when the associated display component changes.

Pascal

```
procedure OnDisplayComponentChange(ASender: TObject);
```

C++

```
__fastcall OnDisplayComponentChange(TObject * ASender);
```

Description

The component specified by the DisplayComponent (see page 6355) property calls OnDisplayComponentChange to notify this component of changes to its properties or child Web items.

ASender is the display component that calls this method.

OnDisplayComponentChange calls the DisplayComponentChange (see page 6352) method, which responds to the change.

See Also

DisplayComponentChange (see page 6352)

DisplayComponent (see page 6355)

65.1.11.4.7 TAdapterDisplayReferenceGroup.SetWebDisplayComponent Method

Sets the value of the DisplayComponent (see page 6355) property.

Pascal

```
procedure SetWebDisplayComponent(const AComponent: TComponent); override;
```

C++

```
virtual __fastcall SetWebDisplayComponent(const TComponent * AComponent);
```

Description

SetWebDisplayComponent is the protected write implementation of the DisplayComponent (see page 6355) property.

AComponent is the new value of the DisplayComponent (see page 6355) property.

See Also

DisplayComponent (see page 6355)

65.1.11.5 TAdapterDisplayReferenceGroup Properties

The properties of the TAdapterDisplayReferenceGroup class are listed here.

Public Properties

	Name	Description
	DisplayComponent (see page 6355)	Associates the group with a component that generates the display controls.

65.1.11.5.1 TAdapterDisplayReferenceGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid (see page 6386) or TAdapterFieldGroup (see page 6367).

TAdapterDisplayReferenceGroup (see page 6349) descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

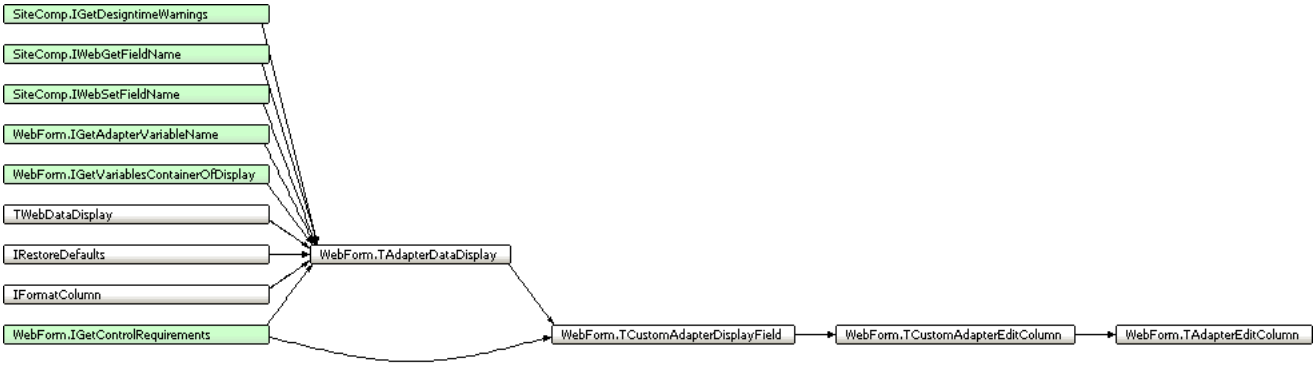
TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

65.1.12 TAdapterEditColumn Class

TAdapterEditColumn displays an adapter field as a form field in a grid.

Class Hierarchy



Pascal

```
TAdapterEditColumn = class(TCustomAdapterEditColumn);
```

C++

```
class TAdapterEditColumn : public TCustomAdapterEditColumn;
```

File

WebForm

Description

TAdapterEditColumn has the same functionality as TAdapterDisplayField (see page 6339), but is designed to work with grids. TAdapterEditColumn is a minimal implementation of TCustomAdapterEditColumn (see page 6442).

See Also

TCustomAdapterEditColumn (see page 6442)

TAdapterDisplayField (see page 6339)

65.1.12.1 TAdapterEditColumn Members

The following tables list the members exposed by TAdapterEditColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 6437)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6437)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
V	GetLayoutAttributes (see page 6437)	Returns the value of the LayoutAttributes property.

	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 6437)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 6444)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6438)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 6438)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
V	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6326)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 6436)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6434).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6358)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6358)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6359)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6359)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6360)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6360)	Specifies the number of columns in a display field.
	DisplayRows (see page 6360)	Specifies the number of rows in a display field.
	DisplayType (see page 6361)	Sets the display type for the column.
	DisplayWidth (see page 6361)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6361)	Specifies the name of the associated dataset field.
	HideOptions (see page 6362)	When field should be hidden.
	ImageHeight (see page 6362)	Specifies the height of the image in the column.
	ImageWidth (see page 6362)	Specifies the width of the image in the column.
	InputType (see page 6362)	Specifies the input type for the display field.
	MaxLength (see page 6363)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6363)	Indicates the number of rows that can be selected.
	Style (see page 6363)	Adds style attributes to the generated display control.
	StyleRule (see page 6364)	This is StyleRule, a member of class TAdapterEditColumn.
	TextAreaWrap (see page 6364)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6364)	Specifies the vertical alignment of the generated display control.

	ViewMode (see page 6365)	Indicates the viewing state of the adapter field.

65.1.12.2 TAdapterEditColumn Properties

The properties of the TAdapterEditColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 6358)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6358)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6359)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6359)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6360)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6360)	Specifies the number of columns in a display field.
	DisplayRows (see page 6360)	Specifies the number of rows in a display field.
	DisplayType (see page 6361)	Sets the display type for the column.
	DisplayWidth (see page 6361)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6361)	Specifies the name of the associated dataset field.
	HideOptions (see page 6362)	When field should be hidden.
	ImageHeight (see page 6362)	Specifies the height of the image in the column.
	ImageWidth (see page 6362)	Specifies the width of the image in the column.
	InputType (see page 6362)	Specifies the input type for the display field.
	MaxLength (see page 6363)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6363)	Indicates the number of rows that can be selected.
	Style (see page 6363)	Adds style attributes to the generated display control.
	StyleRule (see page 6364)	This is StyleRule, a member of class TAdapterEditColumn.
	TextAreaWrap (see page 6364)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6364)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6365)	Indicates the viewing state of the adapter field.

65.1.12.2.1 TAdapterEditColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6332)

65.1.12.2.2 TAdapterEditColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

65.1.12.2.3 TAdapterEditColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

65.1.12.2.4 TAdapterEditColumn.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition;
```

C++

```
__property CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes

Caption

65.1.12.2.5 TAdapterEditColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

stylerule

CaptionAttributes

65.1.12.2.6 TAdapterEditColumn.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6439)

65.1.12.2.7 TAdapterEditColumn.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 6439)

65.1.12.2.8 TAdapterEditColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

InputType (see page 6441)

65.1.12.2.9 TAdapterEditColumn.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

65.1.12.2.10 TAdapterEditColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

65.1.12.2.11 TAdapterEditColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

65.1.12.2.12 TAdapterEditColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6440)

ImageWidth (see page 6440)

65.1.12.2.13 TAdapterEditColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 6440)

65.1.12.2.14 TAdapterEditColumn.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

DisplayType (see page 6439)

65.1.12.2.15 TAdapterEditColumn.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

65.1.12.2.16 TAdapterEditColumn.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

65.1.12.2.17 TAdapterEditColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

stylerule

CaptionAttributes

Setting Web Item Properties

65.1.12.2.18 TAdapterEditColumn.StyleRule Property

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

This is StyleRule, a member of class TAdapterEditColumn.

65.1.12.2.19 TAdapterEditColumn.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 6440)

65.1.12.2.20 TAdapterEditColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 6332)

65.1.12.2.21 TAdapterEditColumn.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

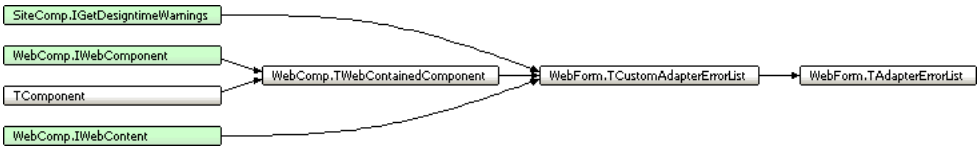
Description

Use ViewMode to specify the viewing state of the field.

65.1.13 TAdapterErrorList Class

TAdapterErrorList is the error list display component.

Class Hierarchy



Pascal

```
TAdapterErrorList = class(TCustomAdapterErrorList);
```

C++

```
class TAdapterErrorList : public TCustomAdapterErrorList;
```

File

WebForm

Description

Use TAdapterErrorList to display errors relating to a specific adapter component.

TAdapterErrorList is a minimal implementation of TCustomAdapterErrorList (see page 6445). To extend its features, create your own concrete descendant of TCustomAdapterErrorList (see page 6445).

Note: Do not confuse TAdapterErrorList with TAdapterErrorsList.

See Also

TAdapterErrorsList (see page 5248)

65.1.13.1 TAdapterErrorList Members

The following tables list the members exposed by TAdapterErrorList.

Protected Methods

	Name	Description
	Content (see page 6447)	Returns the HTML that the Web item generates.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 6447)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
V	ImplContent (see page 6448)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Notification (see page 6448)	Forwards notification messages to all owned components.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
R	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	Content (see page 6076)	Returns the HTML that the Web item generates.
V	Create (see page 6013)	Creates and initializes an instance of TWebContainedComponent (see page 6012).
V	Create (see page 6446)	Creates an instance of TCustomAdapterErrorList (see page 6445).
V	Destroy (see page 6014)	Frees an instance of TWebContainedComponent (see page 6012).
V	Destroy (see page 6446)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6445).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
V	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
W	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.

	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

Published Properties

	Name	Description
	Adapter (see page 6367)	Specifies a related adapter.

65.1.13.2 TAdapterErrorList Properties

The properties of the TAdapterErrorList class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6367)	Specifies a related adapter.

65.1.13.2.1 TAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

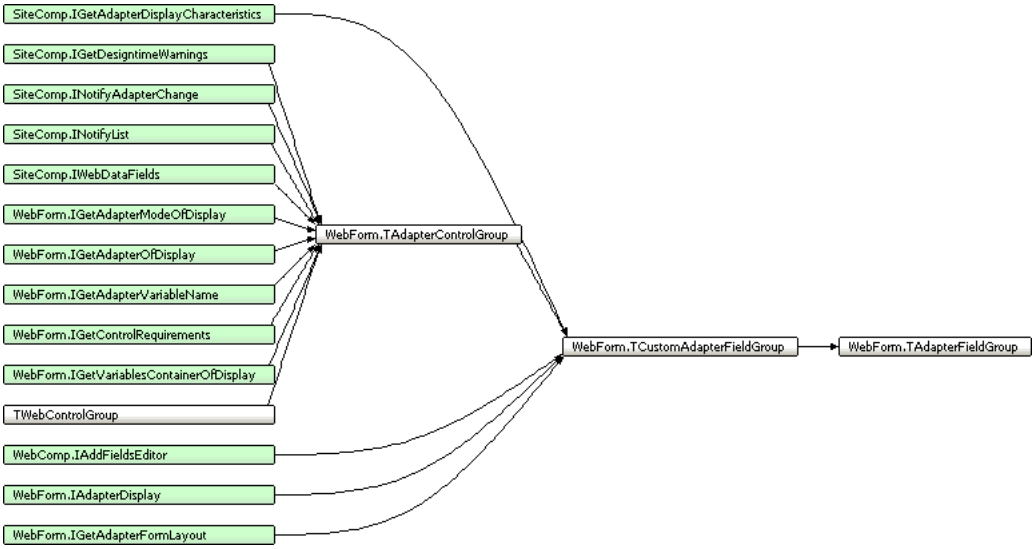
Description

Adapter specifies the adapter component for which this component reports errors.

65.1.14 TAdapterFieldGroup Class

TAdapterFieldGroup is a container for Adapter (see page 6370) display field components such as TAdapterDisplayField (see page 6339). TAdapterFieldGroup labels and fields in an HTML form.

Class Hierarchy



Pascal

```
TAdapterFieldGroup = class(TCustomAdapterFieldGroup);
```

C++

```
class TAdapterFieldGroup : public TCustomAdapterFieldGroup;
```

File

WebForm

Description

Add TAdapterFieldGroup as a child of TAdapterForm (see page 6373).

TAdapterFieldGroup is associated with an Adapter (see page 6370). All of its child display field components will use the same Adapter (see page 6370).

If no display fields have been explicitly added to the TAdapterFieldGroup, display fields associated with all of the Adapter (see page 6370) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 6371) property of TAdapterFieldGroup may change the default display type or input type of Adapter (see page 6370) fields. This may cause the display field children of TAdapterFieldGroup to change appearance.

Use the properties and methods of TAdapterFieldGroup to:

Specify the associated Adapter (see page 6370).

Specify when the Adapter (see page 6370) Mode.

Specify whether default fields should be added.

See Also

TAdapterDisplayField (see page 6339)

TAdapterForm (see page 6373)

65.1.14.1 TAdapterFieldGroup Members

The following tables list the members exposed by TAdapterFieldGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterDisplayCharacteristics (see page 6452)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6452)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6453)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6453)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6453)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetLayoutAttributes (see page 6453)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6454)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.
	ImplCanAddClass (see page 6454)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6454)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	Create (see page 6313)	Creates and initializes an instance of TAdapterControlGroup (see page 6312).
	Create (see page 6451)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6449).
	Destroy (see page 6314)	Frees an instance of TAdapterControlGroup (see page 6312).
	Destroy (see page 6451)	Frees an instance of TCustomAdapterFieldGroup (see page 6449).
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Published Properties

	Name	Description
	Adapter (see page 6370)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6371)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6371)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6372)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6372)	Adds style attributes to the generated button group.
	StyleRule (see page 6373)	Applies a style in the page producer's style sheet to the button group.

65.1.14.2 TAdapterFieldGroup Properties

The properties of the TAdapterFieldGroup class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6370)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6371)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6371)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6372)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6372)	Adds style attributes to the generated button group.
	StyleRule (see page 6373)	Applies a style in the page producer's style sheet to the button group.

65.1.14.2.1 TAdapterFieldGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5536)

GetFieldsList (see page 6319)

AdapterMode (see page 6323)

WebFieldControls

65.1.14.2.2 TAdapterFieldGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6322)

WebFieldControls

65.1.14.2.3 TAdapterFieldGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 6324)

WebFieldControls

Adapter (see page 6322)

65.1.14.2.4 TAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 6455)

65.1.14.2.5 TAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 6455)

StyleRule (see page 6456)

65.1.14.2.6 TAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 6455)

Styles (see page 1097)

StylesFile (see page 1098)

65.1.15 TAdapterForm Class

TAdapterForm generates an HTML form displaying Adapter fields.

Class Hierarchy

Pascal

```
TAdapterForm = class(TCustomAdapterForm);
```

C++

```
class TAdapterForm : public TCustomAdapterForm;
```

File

WebForm

Description

TAdapter form is a basic implementation of TCustomAdapterForm (see page 6456). Add TAdapterForm to a TAdapterPageProducer to display Adapter fields users can view and/or edit, and Adapter actions that users can execute.

Add child components to TAdapterForm to display Adapter fields and Adapter actions. Child components include TAdapterGrid (see page 6386), TAdapterFieldGroup (see page 6367), and TAdapterCommandGroup (see page 6307). Add TAdapterErrors to TAdapterForm to display errors that are generated by an Adapter.

Use the properties and methods of TAdapterForm to:

Customize the appearance of the HTML form.

Indicate the HTTP method used to submit the HTML form.

See Also

TAdapterPageProducer (see page 1076)

TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

TAdapterCommandGroup (see page 6307)

TAdapterErrorList (see page 6365)

65.1.15.1 TAdapterForm Members

The following tables list the members exposed by TAdapterForm.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6458)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6458)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6459)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6459)	Returns the name of the javascript variable that refers to the generated form.
	GetLayoutAttributes (see page 6459)	Returns an object that specifies how the generated group of controls should look.
	ImplCanAddClass (see page 6460)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6460)	Provides the underlying implementation for the Content method.

Public Methods

	Name	Description
	Create (see page 6457)	Creates and initializes an instance of TCustomAdapterForm (see page 6456).
	Destroy (see page 6458)	Frees an instance of TCustomAdapterForm (see page 6456).
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetFormRequirements (see page 6483)	This is GetFormRequirements, a member of class IGetFormRequirements.

Published Properties

	Name	Description
	Custom (see page 6375)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6375)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6376)	Adds style attributes to the generated form.
	StyleRule (see page 6376)	Applies a style in the page producer's style sheet to the form.

65.1.15.2 TAdapterForm Properties

The properties of the TAdapterForm class are listed here.

Published Properties

	Name	Description
	Custom (see page 6375)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6375)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6376)	Adds style attributes to the generated form.
	StyleRule (see page 6376)	Applies a style in the page producer's style sheet to the form.

65.1.15.2.1 TAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 6462)

65.1.15.2.2 TAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

65.1.15.2.3 TAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 6461)

StyleRule (see page 6462)

65.1.15.2.4 TAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style property.

See Also

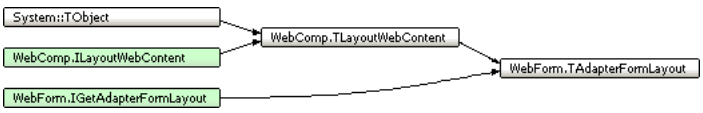
Style (see page 6462)

Styles (see page 1097)

StylesFile (see page 1098)

65.1.16 TAdapterFormLayout Class

Class Hierarchy



Pascal

```
TAdapterFormLayout = class(TLayoutWebContent, IGetAdapterFormLayout);
```

C++

```
class TAdapterFormLayout : public TLayoutWebContent, public IGetAdapterFormLayout;
```

File

WebForm

Description

This is class WebForm.TAdapterFormLayout.

65.1.16.1 TAdapterFormLayout Members

The following tables list the members exposed by TAdapterFormLayout.

Protected Methods

	Name	Description
	_AddRef (see page 5991)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 5991)	Responds when the caller releases a reference to this object's interface.
	EndButtons (see page 6379)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 6380)	This is EndField, a member of class TAdapterFormLayout.
	EndFields (see page 6380)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 6380)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 6380)	This is EndTable, a member of class TAdapterFormLayout.
	GetAdapterFormLayout (see page 6381)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 6381)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 6381)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 6381)	This is GetTableIndent, a member of class TAdapterFormLayout.
	ImplLayoutButton (see page 6381)	Provides the underlying implementation for the LayoutButton method.
	ImplLayoutField (see page 6382)	Provides the underlying implementation for the LayoutField method.
	ImplLayoutLabelAndField (see page 6382)	Provides the underlying implementation for the LayoutLabelAndField method.
	ImplLayoutTable (see page 6383)	Provides the underlying implementation for the LayoutTable method.
	LayoutButton (see page 5994)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 5994)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 5994)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 5995)	This is LayoutTable, a member of class TLayoutWebContent.
	NextColumn (see page 6384)	This is NextColumn, a member of class TAdapterFormLayout.
	QueryInterface (see page 5995)	Returns a reference to a specified interface if the object supports that interface.
	StartButtons (see page 6384)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 6384)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 6384)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 6384)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6378)	Creates a new TLayoutWebContent object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndLayout (see page 6380)	This is EndLayout, a member of class TAdapterFormLayout.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	LayoutButton (see page 6058)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 6058)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 6058)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6059)	This is LayoutTable, a member of class ILayoutWebContent.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BreakButtons (see page 6385)	This is BreakButtons, a member of class TAdapterFormLayout.
R	ChildTableIndent (see page 6385)	This is ChildTableIndent, a member of class TAdapterFormLayout.
	ColumnCount (see page 6385)	This is ColumnCount, a member of class TAdapterFormLayout.
R	FieldIndent (see page 6386)	This is FieldIndent, a member of class TAdapterFormLayout.
R	ParentLayout (see page 5996)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.
	TableHeader (see page 6386)	This is TableHeader, a member of class TAdapterFormLayout.
R	TableIndent (see page 6386)	This is TableIndent, a member of class TAdapterFormLayout.

65.1.16.2 TAdapterFormLayout.Create Constructor

Creates a new TLayoutWebContent object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TAdapterFormLayout(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout property.

See Also

ParentLayout (see page 5996)

65.1.16.3 TAdapterFormLayout Methods

The methods of the TAdapterFormLayout class are listed here.

Protected Methods

	Name	Description
	EndButtons (see page 6379)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 6380)	This is EndField, a member of class TAdapterFormLayout.
	EndFields (see page 6380)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 6380)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 6380)	This is EndTable, a member of class TAdapterFormLayout.
	GetAdapterFormLayout (see page 6381)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 6381)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 6381)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 6381)	This is GetTableIndent, a member of class TAdapterFormLayout.
V	ImplLayoutButton (see page 6381)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 6382)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 6382)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 6383)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 6384)	This is NextColumn, a member of class TAdapterFormLayout.
	StartButtons (see page 6384)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 6384)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 6384)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 6384)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
	EndLayout (see page 6380)	This is EndLayout, a member of class TAdapterFormLayout.

65.1.16.3.1 TAdapterFormLayout.EndButtons Method

Pascal

```
function EndButtons: string;
```

C++

```
__fastcall UnicodeString EndButtons();
```

Description

This is EndButtons, a member of class TAdapterFormLayout.

65.1.16.3.2 TAdapterFormLayout.EndField Method

Pascal

```
function EndField: string;
```

C++

```
__fastcall UnicodeString EndField();
```

Description

This is EndField, a member of class TAdapterFormLayout.

65.1.16.3.3 TAdapterFormLayout.EndFields Method

Pascal

```
function EndFields: string;
```

C++

```
__fastcall UnicodeString EndFields();
```

Description

This is EndFields, a member of class TAdapterFormLayout.

65.1.16.3.4 TAdapterFormLayout.EndLayout Method

Pascal

```
function EndLayout: string;
```

C++

```
__fastcall UnicodeString EndLayout();
```

Description

This is EndLayout, a member of class TAdapterFormLayout.

65.1.16.3.5 TAdapterFormLayout.EndRow Method

Pascal

```
function EndRow: string;
```

C++

```
__fastcall UnicodeString EndRow();
```

Description

This is EndRow, a member of class TAdapterFormLayout.

65.1.16.3.6 TAdapterFormLayout.EndTable Method

Pascal

```
function EndTable: string;
```

C++

```
__fastcall UnicodeString EndTable();
```

Description

This is EndTable, a member of class TAdapterFormLayout.

65.1.16.3.7 TAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TAdapterFormLayout.

65.1.16.3.8 TAdapterFormLayout.GetChildTableIndent Method

Pascal

```
function GetChildTableIndent: string;
```

C++

```
__fastcall UnicodeString GetChildTableIndent();
```

Description

This is GetChildTableIndent, a member of class TAdapterFormLayout.

65.1.16.3.9 TAdapterFormLayout.GetFieldIndent Method

Pascal

```
function GetFieldIndent: string;
```

C++

```
__fastcall UnicodeString GetFieldIndent();
```

Description

This is GetFieldIndent, a member of class TAdapterFormLayout.

65.1.16.3.10 TAdapterFormLayout.GetTableIndent Method

Pascal

```
function GetTableIndent: string;
```

C++

```
__fastcall UnicodeString GetTableIndent();
```

Description

This is GetTableIndent, a member of class TAdapterFormLayout.

65.1.16.3.11 TAdapterFormLayout.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutButton(const UnicodeString HTMLButton,
TLayoutAttributes Attributes);
```

Description

The LayoutButton method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton method.

See Also

LayoutButton (see page 5994)

65.1.16.3.12 TAdapterFormLayout.ImplLayoutField Method

Provides the underlying implementation for the LayoutField method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutField(const UnicodeString HTMLField,
TLayoutAttributes Attributes);
```

Description

The LayoutField method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField method.

See Also

LayoutField (see page 5994)

65.1.16.3.13 TAdapterFormLayout.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

The LayoutLabelAndField method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField method.

See Also

LayoutLabelAndField (see page 5994)

65.1.16.3.14 TAdapterFormLayout.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutTable(const UnicodeString HTMLTable,
TLayoutAttributes Attributes);
```

Description

The LayoutTable method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable method.

See Also

layouttable (see page 5995)

65.1.16.3.15 TAdapterFormLayout.NextColumn Method

Pascal

```
function NextColumn(SubItemCount: Integer): string;
```

C++

```
__fastcall UnicodeString NextColumn(int SubItemCount);
```

Description

This is NextColumn, a member of class TAdapterFormLayout.

65.1.16.3.16 TAdapterFormLayout.StartButtons Method

Pascal

```
function StartButtons: string;
```

C++

```
__fastcall UnicodeString StartButtons();
```

Description

This is StartButtons, a member of class TAdapterFormLayout.

65.1.16.3.17 TAdapterFormLayout.StartFields Method

Pascal

```
function StartFields(ColCount: Integer): string;
```

C++

```
__fastcall UnicodeString StartFields(int ColCount);
```

Description

This is StartFields, a member of class TAdapterFormLayout.

65.1.16.3.18 TAdapterFormLayout.StartRow Method

Pascal

```
function StartRow: string;
```

C++

```
__fastcall UnicodeString StartRow();
```

Description

This is StartRow, a member of class TAdapterFormLayout.

65.1.16.3.19 TAdapterFormLayout.StartTable Method

Pascal

```
function StartTable: string;
```

C++

```
__fastcall UnicodeString StartTable();
```

Description

This is StartTable, a member of class TAdapterFormLayout.

65.1.16.4 TAdapterFormLayout Properties

The properties of the TAdapterFormLayout class are listed here.

Public Properties

	Name	Description
	BreakButtons (see page 6385)	This is BreakButtons, a member of class TAdapterFormLayout.
R	ChildTableIndent (see page 6385)	This is ChildTableIndent, a member of class TAdapterFormLayout.
	ColumnCount (see page 6385)	This is ColumnCount, a member of class TAdapterFormLayout.
R	FieldIndent (see page 6386)	This is FieldIndent, a member of class TAdapterFormLayout.
	TableHeader (see page 6386)	This is TableHeader, a member of class TAdapterFormLayout.
R	TableIndent (see page 6386)	This is TableIndent, a member of class TAdapterFormLayout.

65.1.16.4.1 TAdapterFormLayout.BreakButtons Property

Pascal

```
property BreakButtons: Boolean;
```

C++

```
__property Boolean BreakButtons;
```

Description

This is BreakButtons, a member of class TAdapterFormLayout.

65.1.16.4.2 TAdapterFormLayout.ChildTableIndent Property

Pascal

```
property ChildTableIndent: string;
```

C++

```
__property UnicodeString ChildTableIndent;
```

Description

This is ChildTableIndent, a member of class TAdapterFormLayout.

65.1.16.4.3 TAdapterFormLayout.ColumnCount Property

Pascal

```
property ColumnCount: Integer;
```

C++

```
__property int ColumnCount;
```

Description

This is ColumnCount, a member of class TAdapterFormLayout.

65.1.16.4.4 TAdapterFormLayout.FieldIndent Property

Pascal

```
property FieldIndent: string;
```

C++

```
__property UnicodeString FieldIndent;
```

Description

This is FieldIndent, a member of class TAdapterFormLayout.

65.1.16.4.5 TAdapterFormLayout.TableHeader Property

Pascal

```
property TableHeader: string;
```

C++

```
__property UnicodeString TableHeader;
```

Description

This is TableHeader, a member of class TAdapterFormLayout.

65.1.16.4.6 TAdapterFormLayout.TableIndent Property

Pascal

```
property TableIndent: string;
```

C++

```
__property UnicodeString TableIndent;
```

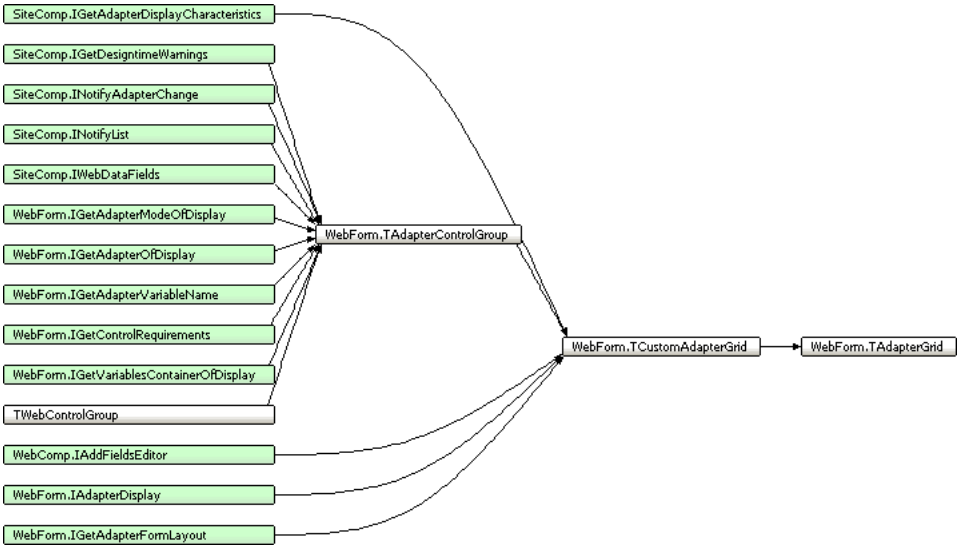
Description

This is TableIndent, a member of class TAdapterFormLayout.

65.1.17 TAdapterGrid Class

TAdapterGrid is a container for Adapter (see page 6389) display field components such as TAdapterDisplayColumn (see page 6333). TAdapterGrid generates an HTML table.

Class Hierarchy



Pascal

```
TAdapterGrid = class(TCustomAdapterGrid);
```

C++

```
class TAdapterGrid : public TCustomAdapterGrid;
```

File

WebForm

Description

Add TAdapterGrid as a child of TAdapterForm (see page 6373). TAdapterGrid is associated with an Adapter (see page 6389). All of it's child display field components will use the same Adapter (see page 6389).

If no display fields have been explicitly added to the TAdapterGrid, display fields associated with all of the Adapter (see page 6389) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 6390) property of TAdapterGrid may change the default display type or input type of Adapter (see page 6389) fields. This may cause the display field children of TAdapterGrid to change appearance.

Use the properties and methods of TAdapterGrid to:

Specify the associated Adapter (see page 6389).

Specify when the Adapter (see page 6389) Mode.

Specify whether default fields should be added.

Customize the appearance of the HTML table.

See Also

TAdapterDisplayColumn (see page 6333)

TAdapterForm (see page 6373)

65.1.17.1 TAdapterGrid Members

The following tables list the members exposed by TAdapterGrid.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterDisplayCharacteristics (see page 6466)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6466)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6466)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6467)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6467)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetNewFieldPrefix (see page 6467)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.
	ImplCanAddClass (see page 6467)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6468)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	Create (see page 6313)	Creates and initializes an instance of TAdapterControlGroup (see page 6312).
	Create (see page 6465)	Creates and initializes an instance of TCustomAdapterGrid (see page 6462).
	Destroy (see page 6314)	Frees an instance of TAdapterControlGroup (see page 6312).
	Destroy (see page 6465)	Frees an instance of TAdapterControlGroup.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.

	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Published Properties

	Name	Description
	Adapter (see page 6389)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6390)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6390)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6391)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6391)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6392)	Determines the basic display characteristics of the table.

65.1.17.2 TAdapterGrid Properties

The properties of the TAdapterGrid class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6389)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6390)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6390)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6391)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6391)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6392)	Determines the basic display characteristics of the table.

65.1.17.2.1 TAdapterGrid.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5536)

GetFieldsList (see page 6319)

AdapterMode (see page 6323)

WebFieldControls

65.1.17.2.2 TAdapterGrid.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6322)

WebFieldControls

65.1.17.2.3 TAdapterGrid.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 6324)

WebFieldControls

Adapter (see page 6322)

65.1.17.2.4 TAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes or RowAttributes property instead.

See Also

TableAttributes (see page 6469)

RowAttributes (see page 6469)

65.1.17.2.5 TAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes or HeadingAttributes property instead.

See Also

TableAttributes (see page 6469)

HeadingAttributes (see page 6468)

65.1.17.2.6 TAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes property instead.

See Also

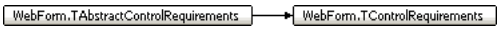
RowAttributes (see page 6469)

HeadingAttributes (see page 6468)

65.1.18 TControlRequirements Class

TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy



Pascal

```
TControlRequirements = class(TAbstractControlRequirements);
```

C++

```
class TControlRequirements : public TAbstractControlRequirements;
```

File

WebForm

Description

Adapter forms create instances of TControlRequirements to store information about variables and functions that are needed for generating an HTML form to display the controls they contain. Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 6312) and TAdapterDataDisplay (see page 6324) descendants) use the TControlRequirements instance that the form provides to store these variable and function declarations.

Typically, applications do not need to create or use TControlRequirements instances. Component writers that create Web items for an adapter form (that is, components that generate HTML controls to reflect the fields of an adapter) use the methods of TControlRequirements when implementing the GetControlRequirements method.

See Also

TAdapterControlGroup (see page 6312)

TAdapterDataDisplay (see page 6324)

TAdapterForm (see page 6373)

65.1.18.1 TControlRequirements Members

The following tables list the members exposed by TControlRequirements.

Public Methods

	Name	Description
	Create (see page 6393)	Creates and initializes an instance of TControlRequirements (see page 6392).
V	DefineFunction (see page 6394)	Adds the declaration of a function to those that this control requirements object stores.
V	DefineVariable (see page 6395)	Adds the declaration of a variable to those that this control requirements object stores.
V	Destroy (see page 6394)	Frees an instance of TControlRequirements (see page 6392).
V	IsFunctionDefined (see page 6395)	Indicates whether a function with the specified name has already been added to the list of control requirements.
V	IsVariableDefined (see page 6396)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
V	RequiresMultipartForm (see page 6396)	Stores the fact that the generated HTML form should be multipart.

Public Properties

	Name	Description
R	NeedMultipartForm (see page 6396)	Indicates whether the form that the adapter form generates must be multipart.

65.1.18.2 TControlRequirements.Create Constructor

Creates and initializes an instance of TControlRequirements (see page 6392).

Pascal

```
constructor Create;
```

C++

```
__fastcall TControlRequirements();
```

Description

Applications should have no need to instantiate TControlRequirements (see page 6392). The adapter form creates instances of this object to assemble and store information from their Web items.

See Also

TAdapterForm (see page 6373)

65.1.18.3 TControlRequirements.Destroy Destructor

Frees an instance of TControlRequirements (see page 6392).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlRequirements();
```

Description

Applications should have no need to create or destroy TControlRequirements (see page 6392) instances. These objects are created and freed by the adapter form.

See Also

TAdapterForm (see page 6373)

65.1.18.4 TControlRequirements Methods

The methods of the TControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6394)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6395)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6395)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6396)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6396)	Stores the fact that the generated HTML form should be multipart.

65.1.18.4.1 TControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); override;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody);
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the

form.

See Also

IsFunctionDefined (see page 6395)

DefineVariable (see page 6395)

65.1.18.4.2 TControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
override;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue,
Boolean AAdapter);
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

IsVariableDefined (see page 6396)

DefineFunction (see page 6394)

65.1.18.4.3 TControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName);
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 6394) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 6394)

IsVariableDefined (see page 6396)

65.1.18.4.4 TControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName);
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 6395) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 6395)

IsFunctionDefined (see page 6395)

65.1.18.4.5 TControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; override;
```

C++

```
virtual __fastcall RequiresMultipartForm();
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

See Also

NeedMultipartForm (see page 6396)

65.1.18.5 TControlRequirements Properties

The properties of the TControlRequirements class are listed here.

Public Properties

	Name	Description
	NeedMultipartForm (see page 6396)	Indicates whether the form that the adapter form generates must be multipart.

65.1.18.5.1 TControlRequirements.NeedMultipartForm Property

Indicates whether the form that the adapter form generates must be multipart.

Pascal

```
property NeedMultipartForm: Boolean;
```

C++

```
__property Boolean NeedMultipartForm;
```

Description

NeedMultipartForm indicates whether any control has called the RequiresMultipartForm (see page 6396) method to indicate that the form that contains it must be multipart. A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

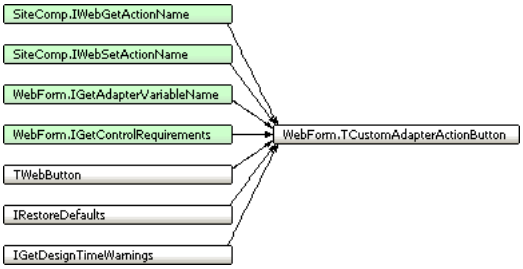
See Also

RequiresMultipartForm (see page 6396)

65.1.19 TCustomAdapterActionButton Class

TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterActionButton = class(TWebButton, IWebGetActionName, IWebSetActionName,
IRestoreDefaults, IGetControlRequirements, IGetAdapterVariableName, IGetDesignTimeWarnings);
```

C++

```
class TCustomAdapterActionButton : public TWebButton, public IWebGetActionName, public
IWebSetActionName, public IRestoreDefaults, public IGetControlRequirements, public
IGetAdapterVariableName, public IGetDesignTimeWarnings;
```

File

WebForm

Description

Use TCustomAdapterActionButton as a base class when defining Web items that generate HTML controls for adapter actions. TCustomAdapterActionButton descendants are added to Web items that appear in the hierarchy of a TAdapterForm (see page 6373) object. The adapter action button generates an HTML control that appears in the HTML form. When the end user clicks that control, it sends a request message that executes the adapter action.

TCustomAdapterActionButton introduces properties to link this component to the adapter action it executes, and for defining the type and appearance of the generated control in the HTML form.

See Also

TAdapterActionButton (see page 6286)

TCustomAdapterAction (see page 5565)

TAdapterForm (see page 6373)

65.1.19.1 TCustomAdapterActionButton Members

The following tables list the members exposed by TCustomAdapterActionButton.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6400)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6400)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6400)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6400)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 6401)	Returns the value of the Caption property.
	GetControlRequirements (see page 6401)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6401)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6401)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
V	GetLayoutAttributes (see page 6402)	Returns the layout object that specifies how the generated control should look.
V	ImplContent (see page 6402)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 6402)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 6403)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
V	Notification (see page 6403)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6403)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 6404)	This is SetActionName, a member of class TCustomAdapterActionButton.
V	SetCaption (see page 6404)	Sets the value of the Caption property.

Public Methods

	Name	Description
V	Create (see page 6398)	Creates and initializes an instance of TCustomAdapterActionButton (see page 6397).
V	Destroy (see page 6399)	Frees an instance of TCustomAdapterActionButton (see page 6397).

Public Properties

	Name	Description
	ActionName (see page 6405)	Specifies the name of the action taken by the Web button.
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6405)	Specifies the horizontal alignment of the generated HTML control.
	DisplayColumns (see page 6405)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6406)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6406)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6407)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6407)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6408)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6408)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6408)	Specifies the name of the page to display after successfully executing the action.
	VAlign (see page 6409)	Specifies the vertical alignment of the generated HTML control.

65.1.19.2 TCustomAdapterActionButton.Create Constructor

Creates and initializes an instance of TCustomAdapterActionButton (see page 6397).

Pascal

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

C++

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

Description

Call Create to instantiate an adapter action button at runtime. Adapter action buttons added to an adapter form at design time are created automatically.

AOwner is the component, typically a Web module, that is responsible for freeing the TCustomAdapterActionButton (see page 6397) instance. This becomes the value of the Owner property.

See Also

Owner

65.1.19.3 TCustomAdapterActionButton.Destroy Destructor

Frees an instance of TCustomAdapterActionButton (see page 6397).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterActionButton();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterActionButton (see page 6397) is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.19.4 TCustomAdapterActionButton Methods

The methods of the TCustomAdapterActionButton class are listed here.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6400)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6400)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6400)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6400)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
	GetCaption (see page 6401)	Returns the value of the Caption property.
	GetControlRequirements (see page 6401)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6401)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6401)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
	GetLayoutAttributes (see page 6402)	Returns the layout object that specifies how the generated control should look.

	ImplContent (see page 6402)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6402)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
	ImplRestoreDefaults (see page 6403)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
	Notification (see page 6403)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6403)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 6404)	This is SetActionName, a member of class TCustomAdapterActionButton.
	SetCaption (see page 6404)	Sets the value of the Caption property.

65.1.19.4.1 TCustomAdapterActionButton.FindAssociatedAction Method

Pascal

```
function FindAssociatedAction: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedAction();
```

Description

This is FindAssociatedAction, a member of class TCustomAdapterActionButton.

65.1.19.4.2 TCustomAdapterActionButton.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

This is GetActionName, a member of class TCustomAdapterActionButton.

65.1.19.4.3 TCustomAdapterActionButton.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.

65.1.19.4.4 TCustomAdapterActionButton.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.

65.1.19.4.5 TCustomAdapterActionButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption

SetCaption

65.1.19.4.6 TCustomAdapterActionButton.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TCustomAdapterActionButton.

65.1.19.4.7 TCustomAdapterActionButton.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.

65.1.19.4.8 TCustomAdapterActionButton.GetHTMLFormVarName Method

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.

65.1.19.4.9 TCustomAdapterActionButton.GetLayoutAttributes Method

Returns the layout object that specifies how the generated control should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebButton, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout of buttons does not make use of layout attributes.

See Also

Content

65.1.19.4.10 TCustomAdapterActionButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content

ILayoutWebContent (see page 6057)

65.1.19.4.11 TCustomAdapterActionButton.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.

65.1.19.4.12 TCustomAdapterActionButton.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.

65.1.19.4.13 TCustomAdapterActionButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

65.1.19.4.14 TCustomAdapterActionButton.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TCustomAdapterActionButton.

65.1.19.4.15 TCustomAdapterActionButton.SetActionName Method

Pascal

```
procedure SetActionName(const AValue: string);
```

C++

```
__fastcall SetActionName(const UnicodeString AValue);
```

Description

This is SetActionName, a member of class TCustomAdapterActionButton.

65.1.19.4.16 TCustomAdapterActionButton.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption

GetCaption

65.1.19.5 TCustomAdapterActionButton Properties

The properties of the TCustomAdapterActionButton class are listed here.

Public Properties

	Name	Description
	ActionName (see page 6405)	Specifies the name of the action taken by the Web button.
	Align (see page 6405)	Specifies the horizontal alignment of the generated HTML control.
	DisplayColumns (see page 6405)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6406)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6406)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6407)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6407)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6408)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6408)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6408)	Specifies the name of the page to display after successfully executing the action.
	VAlign (see page 6409)	Specifies the vertical alignment of the generated HTML control.

65.1.19.5.1 TCustomAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 5572)

65.1.19.5.2 TCustomAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6409)

DisplayColumns (see page 6405)

ItemCount (see page 5463)

65.1.19.5.3 TCustomAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton (see page 6397) object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton (see page 6397) starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 5463)

DisplayType (see page 6406)

65.1.19.5.4 TCustomAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton (see page 6397) uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer (see page 6408) property to a component that can provide the image or images.

See Also

ImageProducer (see page 6408)

Caption

GetDisplayStyle (see page 5445)

65.1.19.5.5 TCustomAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property UnicodeString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 6408)

pagename

65.1.19.5.6 TCustomAdapterActionButton.HideOptions Property

Controls when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton (see page 6397) always generates a control for its associated adapter action.

See Also

GetVisible (see page 5450)

WebEnabled (see page 5461)

HasExecuteAccess (see page 5451)

65.1.19.5.7 TCustomAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType (see page 6406) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6408)

ImageWidth (see page 6408)

DisplayType (see page 6406)

65.1.19.5.8 TCustomAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType (see page 6406) is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 6406)

ImageHeight (see page 6407)

ImageWidth (see page 6408)

65.1.19.5.9 TCustomAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType (see page 6406) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6408)

ImageHeight (see page 6407)

DisplayType (see page 6406)

65.1.19.5.10 TCustomAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 6406)

pagename

65.1.19.5.11 TCustomAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6405)

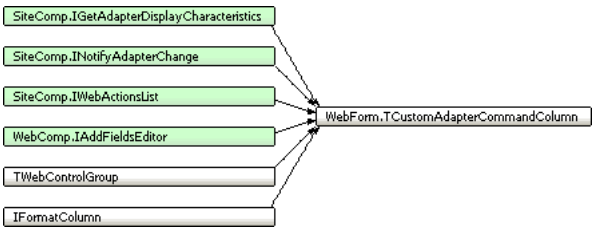
DisplayColumns (see page 6405)

ItemCount (see page 5463)

65.1.20 TCustomAdapterCommandColumn Class

TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.

Class Hierarchy



Pascal

```
TCustomAdapterCommandColumn = class(TWebControlGroup, IWebActionsList, IAddFieldsEditor,
```

```
  IFormatColumn, IGetAdapterDisplayCharacteristics, INotifyAdapterChange);
```

C++

```
  class TCustomAdapterCommandColumn : public TWebControlGroup, public IWebActionsList, public
  IAddFieldsEditor, public IFormatColumn, public IGetAdapterDisplayCharacteristics, public
  INotifyAdapterChange;
```

File

WebForm

Description

Use TCustomAdapterCommandColumn as a base class when creating a specialized display control that organizes the output of several adapter action buttons (TAdapterActionButton (see page 6286) objects) into a column of an adapter grid (TAdapterGrid (see page 6386)). TCustomAdapterCommandColumn generates the HTML to lay out a single column of an adapter grid. Each cell in the column can contain one or more adapter action buttons, each of which executes an adapter action. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TCustomAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 6416), WebFieldControls, and VisibleCommands (see page 6420) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 6417) and CaptionAttributes (see page 6418) properties, which determine the appearance of the label at the top of the column.

The Align (see page 6417), VAlign (see page 6420), and DisplayColumns (see page 6418) properties, which determine the layout of controls inside each cell.

The Style (see page 6419) and StyleRule (see page 6419) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterCommandColumn (see page 6301)

TAdapterGrid (see page 6386)

TAdapterEditColumn (see page 6355)

TAdapterDisplayColumn (see page 6333)

TAdapterActionButton (see page 6286)

65.1.20.1 TCustomAdapterCommandColumn Members

The following tables list the members exposed by TCustomAdapterCommandColumn.

Protected Methods

	Name	Description
	FormatColumnData (see page 6412)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6413)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6413)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6413)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6414)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6414)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6415)	Responds to changes that affect the associated adapter.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 6416)	Identifies the helper object that enables this component to work with the editor at design time.

Public Methods

	Name	Description
V	Create (see page 6411)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6409).
V	Destroy (see page 6412)	Frees an instance of TCustomAdapterCommandColumn (see page 6409).
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6416)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6417)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6417)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6418)	Specifies the attributes of the Caption (see page 6417) property.
	Custom (see page 6418)	Represents a tag option and its value.
	DisplayColumns (see page 6418)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6419)	Adds style attributes to the generated caption.
	StyleRule (see page 6419)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6420)	Specifies the vertical alignment of the action control buttons within the column.
R	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 6420)	Lists the child Web items that generate controls that appear in cells of the column.

65.1.20.2 TCustomAdapterCommandColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6409).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterCommandColumn (see page 6409) at runtime. TCustomAdapterCommandColumn (see page 6409) descendants added at design time are instantiated automatically.

AOwner is the component that becomes the value of the Owner property. Typically, this is the Web module.

See Also

Owner

65.1.20.3 TCustomAdapterCommandColumn.Destroy Destructor

Frees an instance of TCustomAdapterCommandColumn (see page 6409).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandColumn();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterCommandColumn (see page 6409) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.20.4 TCustomAdapterCommandColumn Methods

The methods of the TCustomAdapterCommandColumn class are listed here.

Protected Methods

	Name	Description
	FormatColumnData (see page 6412)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6413)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6413)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6413)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6414)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6414)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6415)	Responds to changes that affect the associated adapter.

65.1.20.4.1 TCustomAdapterCommandColumn.FormatColumnData Method

Generates the HTML for the cells in the column that contain controls to execute actions.

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnData(const UnicodeString Content, TWebContentOptions
Options);
```

Description

The adapter grid calls FormatColumnData to update the HTML for the contents of a cell to include any column attributes that affect the layout of that content. This method adds attributes to the HTML that reflect the Align (see page 6417) and VAlign (see page 6420) properties.

Content is the HTML that represents the content of the cell. It is the value that the Content method returns.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnData returns the resulting HTML.

See Also

Content

TAdapterGrid (see page 6386)

Align (see page 6417)

VAlign (see page 6420)

65.1.20.4.2 TCustomAdapterCommandColumn.FormatColumnHeading Method

Generates the HTML for the table header cell for this column.

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnHeading(TWebContentOptions Options);
```

Description

The adapter grid calls FormatColumnHeading to generate the HTML for the top cell of the column (the one that contains the caption). This method generates the HTML to reflect the Caption (see page 6417) and CaptionAttributes (see page 6418) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnHeading returns the resulting HTML.

See Also

Caption (see page 6417)

TAdapterGrid (see page 6386)

CaptionAttributes (see page 6418)

65.1.20.4.3 TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.

65.1.20.4.4 TCustomAdapterCommandColumn.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

GetLayoutAttributes adds returns a TLayoutAttributes (see page 5985) object that encodes the values of the Style (see page 6419), StyleRule (see page 6419), and Custom (see page 6418) properties.

See Also

Content

Style (see page 6419)

StyleRule (see page 6419)

Custom (see page 6418)

65.1.20.4.5 TCustomAdapterCommandColumn.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TCustomAdapterCommandColumn (see page 6409) descendant can add a component of type AClass to its WebFieldControls property list.

AParent is the TCustomAdapterCommandColumn (see page 6409) instance to which the component will be added.

AClass is the class of the component to be added.

In TCustomAdapterCommandColumn (see page 6409), ImplCanAddClass returns true if AClass is a descendant of TCustomAdapterActionButton (see page 6397) or TCustomLayoutGroup.

See Also

CanAddClass

WebFieldControls

TCustomAdapterActionButton (see page 6397)

TCustomLayoutGroup

65.1.20.4.6 TCustomAdapterCommandColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML for the controls in one cell of the column. ImplContent generates the HTML to display the controls that the VisibleCommands (see page 6420) property lists, reflecting the values of the DisplayColumns (see page 6418), .Style (see page 6419), StyleRule (see page 6419), and Custom (see page 6418) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TCustomAdapterCommandColumn (see page 6409) should format the HTML controls it generates. It implements the ILayoutWebContent (see page 6057) interface.

ImplContent returns the generated HTML.

Note: ImplContent does not include attributes for the alignment of controls inside the cell. The FormatColumnData (see page 6412) method adds this information to the HTML that ImplContent generates.

See Also

Content

FormatColumnData (see page 6412)

ILayoutWebContent (see page 6057)

VisibleCommands (see page 6420)

DisplayColumns (see page 6418)

Style (see page 6419)

StyleRule (see page 6419)

Custom (see page 6418)

65.1.20.4.7 TCustomAdapterCommandColumn.NotifyAdapterChange Method

Responds to changes that affect the associated adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the associated adapter changes. This method updates the set of default actions for that adapter to reflect the current change, and passes on the change notification to the actions that the WebFieldControls property lists.

See Also

GetDefaultWebComponents

WebFieldControls

65.1.20.5 TCustomAdapterCommandColumn Properties

The properties of the TCustomAdapterCommandColumn class are listed here.

Protected Properties

	Name	Description
	AdapterActionsListHelper (see page 6416)	Identifies the helper object that enables this component to work with the editor at design time.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6416)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6417)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6417)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6418)	Specifies the attributes of the Caption (see page 6417) property.
	Custom (see page 6418)	Represents a tag option and its value.
	DisplayColumns (see page 6418)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6419)	Adds style attributes to the generated caption.
	StyleRule (see page 6419)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6420)	Specifies the vertical alignment of the action control buttons within the column.
	VisibleCommands (see page 6420)	Lists the child Web items that generate controls that appear in cells of the column.

65.1.20.5.1 TCustomAdapterCommandColumn.AdapterActionsListHelper Property

Identifies the helper object that enables this component to work with the editor at design time.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component. This helper object implements methods for locating child Web items for specified actions, for listing the child Web items of this component, and for specifying the text of menu commands that operate on this component's list of adapter action buttons.

See Also

WebFieldControls

65.1.20.5.2 TCustomAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5226)

VisibleCommands (see page 6420)

65.1.20.5.3 TCustomAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6420)

DisplayColumns (see page 6418)

65.1.20.5.4 TCustomAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 6418)

65.1.20.5.5 TCustomAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption (see page 6417) property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 6417)

65.1.20.5.6 TCustomAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn (see page 6409) inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 6419)

StyleRule (see page 6419)

65.1.20.5.7 TCustomAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one

adapter action button. TCustomAdapterCommandColumn (see page 6409) places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls

65.1.20.5.8 TCustomAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes (see page 6418) property instead.

See Also

Custom (see page 6418)

StyleRule (see page 6419)

CaptionAttributes (see page 6418)

65.1.20.5.9 TCustomAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 6419) property instead.

See Also

Custom (see page 6418)

Style (see page 6419)

Styles (see page 1097)

StylesFile (see page 1098)

65.1.20.5.10 TCustomAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6417)

DisplayColumns (see page 6418)

65.1.20.5.11 TCustomAdapterCommandColumn.VisibleCommands Property

Lists the child Web items that generate controls that appear in cells of the column.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

Description

VisibleCommands is the list of child Web items that generate HTML controls for the cells of the adapter command column. Each element in this list is an adapter action button (TAdapterActionButton (see page 6286) object).

If you have explicitly added child Web items to this component, VisibleCommands is the same as the value of the WebFieldControls property. Otherwise, if the AddDefaultCommands (see page 6416) property is true, VisibleCommands lists adapter action buttons for the default actions of the associated adapter. If AddDefaultCommands (see page 6416) is false, VisibleCommands is an empty list.

See Also

WebFieldControls

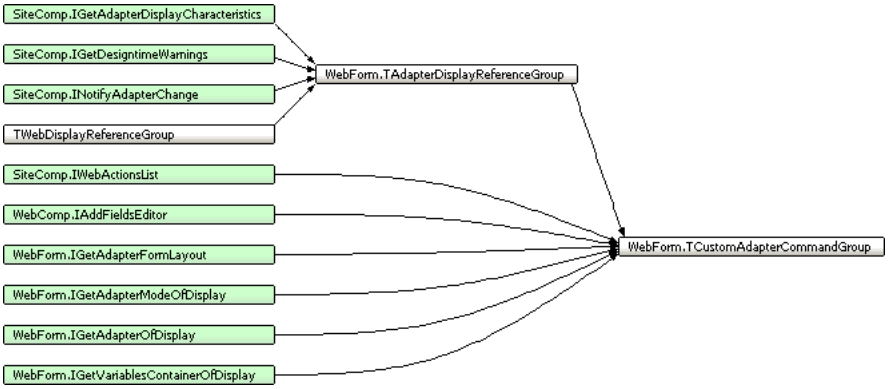
AddDefaultCommands (see page 6416)

TAdapterActionButton (see page 6286)

65.1.21 TCustomAdapterCommandGroup Class

TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6307).

Class Hierarchy



Pascal

```
TCustomAdapterCommandGroup = class(TAdapterDisplayReferenceGroup, IWebActionsList,
IGetAdapterOfDisplay, IGetVariablesContainerOfDisplay, IAddFieldsEditor,
IGetAdapterModeOfDisplay, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterCommandGroup : public TAdapterDisplayReferenceGroup, public
IWebActionsList, public IGetAdapterOfDisplay, public IGetVariablesContainerOfDisplay, public
IAddFieldsEditor, public IGetAdapterModeOfDisplay, public IGetAdapterFormLayout;
```

File

WebForm

Description

Derive from TCustomAdapterCommandGroup to define a component that generates a group of HTML buttons or other HTML elements that execute adapter actions.

See Also

TAdapterDisplayReferenceGroup (see page 6349)

TAdapterCommandGroup (see page 6307)

TAdapterForm (see page 6373)

TAdapterFieldGroup (see page 6367)

TAdapterGrid (see page 6386)

TAdapterActionButton (see page 6286)

65.1.21.1 TCustomAdapterCommandGroup Members

The following tables list the members exposed by TCustomAdapterCommandGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6352)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6352)	Responds when the DisplayComponent (see page 6355) property changes.
	GetAdapter (see page 6424)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 6353)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 6424)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6424)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6424)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetDesigntimeWarnings (see page 6353)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	ImplCanAddClass (see page 6425)	Checks the inheritance of the given component.
	ImplContent (see page 6425)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6376) component.
	NotifyAdapterChange (see page 6353)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6354)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6354)	Sets the value of the DisplayComponent (see page 6355) property.

Protected Properties

	Name	Description
	Adapter (see page 6426)	Provides read-only access to the Adapter that this command group uses.
	AdapterActionsListHelper (see page 6426)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Methods

	Name	Description
	Create (see page 6423)	Creator for TCustomAdapterCommandGroup (see page 6421).
	Destroy (see page 6423)	Frees an instance of TAdapterDisplayReferenceGroup.
	FindAction (see page 1018)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1018)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 961)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 6053)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6053)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6053)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetNewFieldPrefix (see page 6054)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 1019)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1019)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6426)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6426)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DisplayComponent (see page 6355)	Associates the group with a component that generates the display controls.
	Style (see page 6427)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6427)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
	VisibleActions (see page 1019)	This is VisibleActions, a member of class IWebActionsList.

	VisibleCommands (see page 6427)	Provides read-only access to the TWebComponentList (see page 5999) that stores the list of visible components.

65.1.21.2 TCustomAdapterCommandGroup.Create Constructor

Creator for TCustomAdapterCommandGroup (see page 6421).

Pascal

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

C++

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

Description

This constructor calls the base class version, and then creates a TAdapterActionsListHelper (see page 6294) component and sets the AddDefaultActions property to true by default.

See Also

Create_d

AddDefaultActions (see page 6301)

65.1.21.3 TCustomAdapterCommandGroup.Destroy Destructor

Frees an instance of TAdapterDisplayReferenceGroup.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandGroup();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.21.4 TCustomAdapterCommandGroup Methods

The methods of the TCustomAdapterCommandGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 6424)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterFormLayout (see page 6424)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6424)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6424)	Returns the adapter mode defined by the TAdapterPageProducer components.
	ImplCanAddClass (see page 6425)	Checks the inheritance of the given component.

	ImplContent (see page 6425)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6376) component.

65.1.21.4.1 TCustomAdapterCommandGroup.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class TCustomAdapterCommandGroup.

65.1.21.4.2 TCustomAdapterCommandGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.

65.1.21.4.3 TCustomAdapterCommandGroup.GetAdapterModeOfAdapter Method

Returns the adapter mode defined by the adapter component.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

The GetAdapterModeOfAdapter function returns the adapter mode defined by the adapter component. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert (see page 3698), and Delete (see page 3653) modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

65.1.21.4.4 TCustomAdapterCommandGroup.GetAdapterModeOfDisplay Method

Returns the adapter mode defined by the TAdapterPageProducer components.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

This function returns the adapter mode defined by the TAdapterPageProducer components. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert (see page 3698), and Delete (see page 3653) modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

65.1.21.4.5 TCustomAdapterCommandGroup.ImplCanAddClass Method

Checks the inheritance of the given component.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddClass checks the inheritance of the given component. If it is derived from either TCustomAdapterActionButton (see page 6397) or TCustomLayoutGroup, it will return True to indicate that the class can be added to this TCustomAdapterCommandGroup (see page 6421).

65.1.21.4.6 TCustomAdapterCommandGroup.ImplContent Method

Lays out the buttons for the command group using the TAdapterFormLayout (see page 6376) component.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The ImplContent function lays out the buttons for the command group using the TAdapterFormLayout (see page 6376) component. It loops through all of the visible child components, calling them to render their HTML content.

65.1.21.5 TCustomAdapterCommandGroup Properties

The properties of the TCustomAdapterCommandGroup class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 6426)	Provides read-only access to the Adapter that this command group uses.
	AdapterActionsListHelper (see page 6426)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6426)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6426)	Provides read and write access to the custom HTML element attributes, such as Valign="top."

	Style (see page 6427)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6427)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
R	VisibleCommands (see page 6427)	Provides read-only access to the TWebComponentList (see page 5999) that stores the list of visible components.

65.1.21.5.1 TCustomAdapterCommandGroup.Adapter Property

Provides read-only access to the Adapter that this command group uses.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

The Adapter property provides read-only access to the Adapter that this command group uses.

65.1.21.5.2 TCustomAdapterCommandGroup.AdapterActionsListHelper Property

Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

The AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

65.1.21.5.3 TCustomAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

65.1.21.5.4 TCustomAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as Valign="top."

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as Valign="top." The content of this string will be written as it is when generating HTML.

65.1.21.5.5 TCustomAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

65.1.21.5.6 TCustomAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

65.1.21.5.7 TCustomAdapterCommandGroup.VisibleCommands Property

Provides read-only access to the TWebComponentList (see page 5999) that stores the list of visible components.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

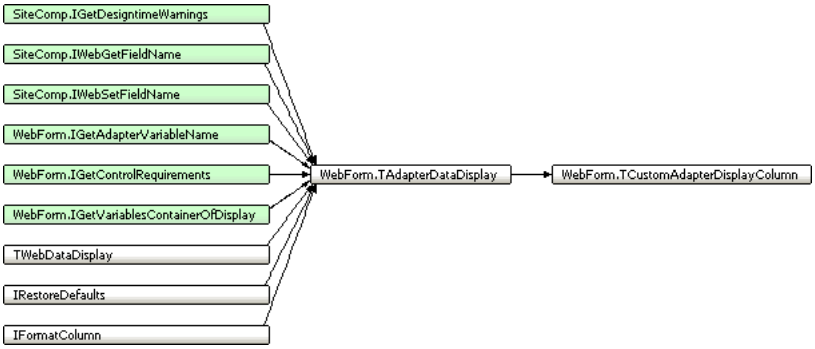
Description

The VisibleCommands property provides read-only access to the TWebComponentList (see page 5999) that stores the list of visible components.

65.1.22 TCustomAdapterDisplayColumn Class

TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.

Class Hierarchy



Pascal

```
TCustomAdapterDisplayColumn = class(TAdapterDataDisplay);
```

C++

```
class TCustomAdapterDisplayColumn : public TAdapterDataDisplay;
```

File

WebForm

Description

TCustomAdapterDisplayColumn is the base class for grid components that display adapter field data. Do not directly instantiate TCustomAdapterDisplayColumn. Use TAdapterDisplayColumn (see page 6333) or create your own concrete descendants of TCustomAdapterDisplayColumn.

See Also

IWebContent (see page 6075)

TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

TAdapterCommandColumn (see page 6301)

TAdapterCommandGroup (see page 6307)

TAdapterDisplayColumn (see page 6333)

65.1.22.1 TCustomAdapterDisplayColumn Members

The following tables list the members exposed by TCustomAdapterDisplayColumn.

Protected Methods

	Name	Description
	CanAddClass (see page 6430)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
	ControlContent (see page 6431)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.

	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 6431)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6431)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6431)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplCanAddClass (see page 6432)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
	ImplContent (see page 6329)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6432)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 6432)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplRestoreDefaults (see page 6432)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6430)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6428).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	IsColumn (see page 6432)	Indicates whether the component generates the controls that appear in the column of a data grid.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6332)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayType (see page 6433)	Sets the display type for the column.
	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6332)	Specifies the name of the associated dataset field.
	HideOptions (see page 6332)	When field should be hidden.
	ImageHeight (see page 6433)	Specifies the height of the image in the column.
	ImageWidth (see page 6434)	Specifies the width of the image in the column.
	VAlign (see page 6332)	Specifies the vertical alignment of the generated display control.

65.1.22.2 TCustomAdapterDisplayColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6428).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterDisplayColumn (see page 6428) at runtime. TCustomAdapterDisplayColumn (see page 6428) objects added at design time using the Web Page editor are instantiated automatically.

65.1.22.3 TCustomAdapterDisplayColumn Methods

The methods of the TCustomAdapterDisplayColumn class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 6430)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
	ControlContent (see page 6431)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.
	FormatImage (see page 6431)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6431)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6431)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	ImplCanAddClass (see page 6432)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
	ImplFormatColumnHeading (see page 6432)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 6432)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplRestoreDefaults (see page 6432)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

Public Methods

	Name	Description
	IsColumn (see page 6432)	Indicates whether the component generates the controls that appear in the column of a data grid.

65.1.22.3.1 TCustomAdapterDisplayColumn.CanAddClass Method

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddClass, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.2 TCustomAdapterDisplayColumn.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.3 TCustomAdapterDisplayColumn.FormatImage Method

Pascal

```
function FormatImage(const VarName: string; const HTMLName: string; Options:
TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatImage(const UnicodeString VarName, const UnicodeString
HTMLName, TWebContentOptions Options);
```

Description

This is FormatImage, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.4 TCustomAdapterDisplayColumn.FormatListDisplay Method

Pascal

```
function FormatListDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall UnicodeString FormatListDisplay(const UnicodeString VarName, const UnicodeString
HTMLName);
```

Description

This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.5 TCustomAdapterDisplayColumn.FormatTextDisplay Method

Pascal

```
function FormatTextDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall UnicodeString FormatTextDisplay(const UnicodeString VarName, const UnicodeString
HTMLName);
```

Description

This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.6 TCustomAdapterDisplayColumn.ImplCanAddClass Method

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.7 TCustomAdapterDisplayColumn.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.8 TCustomAdapterDisplayColumn.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.9 TCustomAdapterDisplayColumn.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

65.1.22.3.10 TCustomAdapterDisplayColumn.IsColumn Method

Indicates whether the component generates the controls that appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a TCustomAdapterDisplayColumn (see page 6428) object generates the controls that appear in a single column of a data grid. As implemented in TCustomAdapterDisplayColumn (see page 6428), IsColumn always returns true.

See Also

IsColumn

65.1.22.4 TCustomAdapterDisplayColumn Properties

The properties of the TCustomAdapterDisplayColumn class are listed here.

Public Properties

	Name	Description
	DisplayType (see page 6433)	Sets the display type for the column.
	ImageHeight (see page 6433)	Specifies the height of the image in the column.
	ImageWidth (see page 6434)	Specifies the width of the image in the column.

65.1.22.4.1 TCustomAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 6433)

ImageWidth (see page 6434)

65.1.22.4.2 TCustomAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 6434)

65.1.22.4.3 TCustomAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 6433) to indicate the width of the image in pixels.

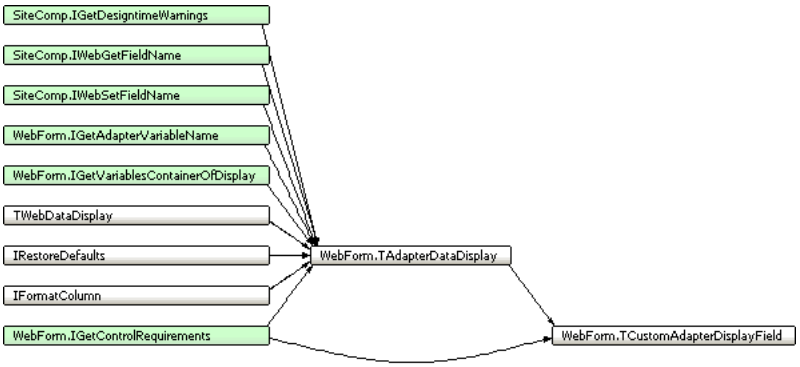
See Also

ImageHeight (see page 6433)

65.1.23 TCustomAdapterDisplayField Class

TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.

Class Hierarchy



Pascal

```
TCustomAdapterDisplayField = class(TAdapterDataDisplay, IGetControlRequirements);
```

C++

```
class TCustomAdapterDisplayField : public TAdapterDataDisplay, public IGetControlRequirements;
```

File

WebForm

Description

TCustomAdapterDisplayField is the base class for free-form components that display and edit adapter field data. Do not directly instantiate TCustomAdapterDisplayField. Use TAdapterDisplayField (see page 6339) or create your own concrete descendants of TCustomAdapterDisplayField.

See Also

TCustomAdapterEditColumn (see page 6442)

TAdapterDisplayField (see page 6339)

IWebContent (see page 6075)

TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

65.1.23.1 TCustomAdapterDisplayField Members

The following tables list the members exposed by TCustomAdapterDisplayField.

Protected Methods

	Name	Description
	AddAttributes (see page 6437)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6437)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
V	GetLayoutAttributes (see page 6437)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplContent (see page 6437)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6438)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6438)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
V	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6436)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6434).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6332)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayColumns (see page 6439)	Specifies the number of columns in a display field.
	DisplayRows (see page 6439)	Specifies the number of rows in a display field.
	DisplayType (see page 6439)	Sets the display type for the column.
	DisplayWidth (see page 6440)	Specifies the width, in characters, of the generated HTML field.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6332)	Specifies the name of the associated dataset field.
	HideOptions (see page 6332)	When field should be hidden.
	ImageHeight (see page 6440)	Specifies the height of the image in the column.
	ImageWidth (see page 6440)	Specifies the width of the image in the column.
	InputType (see page 6441)	Specifies the input type for the display field.
	MaxLength (see page 6441)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6441)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6441)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6332)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6442)	Indicates the viewing state of the adapter field.

65.1.23.2 TCustomAdapterDisplayField.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayField (see page 6434).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterDisplayField (see page 6434) at runtime. TCustomAdapterDisplayField (see page 6434) descendants added at design time using the Web Page editor are instantiated automatically.

65.1.23.3 TCustomAdapterDisplayField Methods

The methods of the TCustomAdapterDisplayField class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 6437)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6437)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	GetLayoutAttributes (see page 6437)	Returns the value of the LayoutAttributes property.
V	ImplContent (see page 6437)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 6438)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6438)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

65.1.23.3.1 TCustomAdapterDisplayField.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string);
```

C++

```
__fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TCustomAdapterDisplayField.

65.1.23.3.2 TCustomAdapterDisplayField.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TCustomAdapterDisplayField.

65.1.23.3.3 TCustomAdapterDisplayField.GetLayoutAttributes Method

Returns the value of the LayoutAttributes property.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

GetLayoutAttributes returns the value of the LayoutAttributes property.

See Also

Content

LayoutAttributes

65.1.23.3.4 TCustomAdapterDisplayField.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

Caption

ILayoutWebContent (see page 6057)

65.1.23.3.5 TCustomAdapterDisplayField.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.

65.1.23.3.6 TCustomAdapterDisplayField.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

65.1.23.4 TCustomAdapterDisplayField Properties

The properties of the TCustomAdapterDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayColumns (see page 6439)	Specifies the number of columns in a display field.
	DisplayRows (see page 6439)	Specifies the number of rows in a display field.
	DisplayType (see page 6439)	Sets the display type for the column.
	DisplayWidth (see page 6440)	Specifies the width, in characters, of the generated HTML field.
	ImageHeight (see page 6440)	Specifies the height of the image in the column.

	ImageWidth (see page 6440)	Specifies the width of the image in the column.
	InputType (see page 6441)	Specifies the input type for the display field.
	MaxLength (see page 6441)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6441)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6441)	Specifies how the display field treats lines that are longer than the display width.
	ViewMode (see page 6442)	Indicates the viewing state of the adapter field.

65.1.23.4.1 TCustomAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6439)

65.1.23.4.2 TCustomAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns (see page 6439) to control the number of rows being displayed.

See Also

DisplayColumns (see page 6439)

65.1.23.4.3 TCustomAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

InputType (see page 6441)

65.1.23.4.4 TCustomAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField (see page 6434) control values are one character (M for modified or I for inserted).

65.1.23.4.5 TCustomAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6440)

ImageWidth (see page 6440)

65.1.23.4.6 TCustomAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 6440) to indicate the width of the image in the column.

See Also

ImageHeight (see page 6440)

65.1.23.4.7 TCustomAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6440)

ImageWidth (see page 6440)

DisplayType (see page 6439)

65.1.23.4.8 TCustomAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

65.1.23.4.9 TCustomAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

65.1.23.4.10 TCustomAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth (see page 6440).

See Also

DisplayWidth (see page 6440)

65.1.23.4.11 TCustomAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

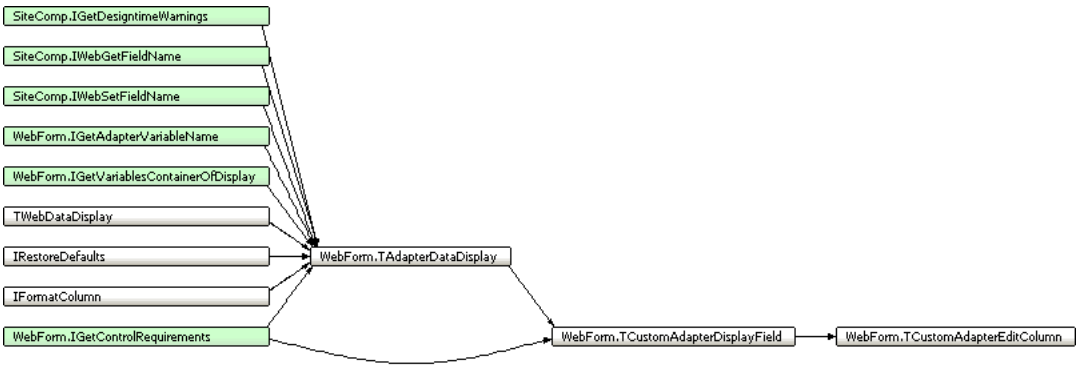
Description

Use ViewMode to specify the viewing state of the field.

65.1.24 TCustomAdapterEditColumn Class

TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.

Class Hierarchy



Pascal

```
TCustomAdapterEditColumn = class(TCustomAdapterDisplayField);
```

C++

```
class TCustomAdapterEditColumn : public TCustomAdapterDisplayField;
```

File

WebForm

Description

TCustomAdapterEditColumn has the same functionality as TCustomAdapterDisplayField (see page 6434), but is designed to work with grids. Do not directly instantiate TCustomAdapterEditColumn. Use TAdapterEditColumn (see page 6355) or create your own concrete descendants of TCustomAdapterEditColumn.

See Also

TAdapterEditColumn (see page 6355)

65.1.24.1 TCustomAdapterEditColumn Members

The following tables list the members exposed by TCustomAdapterEditColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 6437)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6437)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	FieldConditions (see page 6327)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6327)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6327)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 6327)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6327)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6328)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6328)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6328)	Returns the value of the Caption property.
	GetControlRequirements (see page 6328)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6329)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 6329)	This is GetFieldName, a member of class TAdapterDataDisplay.
V	GetLayoutAttributes (see page 6437)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6329)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplContent (see page 6444)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6330)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6330)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6330)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6438)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6330)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 6438)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6331)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
V	SetCaption (see page 6331)	Sets the value of the Caption property.
	SetFieldName (see page 6331)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6326)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 6436)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6434).
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldName (see page 1032)	This is GetFieldName, a member of class IWebGetFieldName.

	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1057)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6332)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayColumns (see page 6439)	Specifies the number of columns in a display field.
	DisplayRows (see page 6439)	Specifies the number of rows in a display field.
	DisplayType (see page 6439)	Sets the display type for the column.
	DisplayWidth (see page 6440)	Specifies the width, in characters, of the generated HTML field.
R	FieldName (see page 1032)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6332)	Specifies the name of the associated dataset field.
	HideOptions (see page 6332)	When field should be hidden.
	ImageHeight (see page 6440)	Specifies the height of the image in the column.
	ImageWidth (see page 6440)	Specifies the width of the image in the column.
	InputType (see page 6441)	Specifies the input type for the display field.
	MaxLength (see page 6441)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6441)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6441)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6332)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6442)	Indicates the viewing state of the adapter field.

65.1.24.2 TCustomAdapterEditColumn Methods

The methods of the TCustomAdapterEditColumn class are listed here.

Protected Methods

	Name	Description
V	ImplContent (see page 6444)	Provides the underlying implementation for the Content method.

65.1.24.2.1 TCustomAdapterEditColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

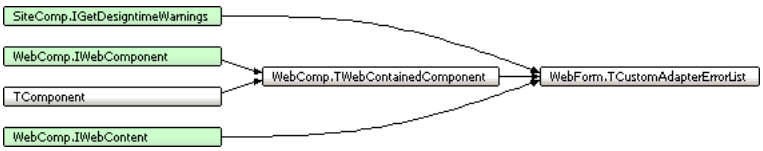
Caption

ILayoutWebContent (see page 6057)

65.1.25 TCustomAdapterErrorList Class

TCustomAdapterErrorList is the base class adapter for components that display error lists.

Class Hierarchy



Pascal

```
TCustomAdapterErrorList = class(TWebContainedComponent, IWebContent, IGetDesigntimeWarnings);
```

C++

```
class TCustomAdapterErrorList : public TWebContainedComponent, public IWebContent, public
IGetDesigntimeWarnings;
```

File

WebForm

Description

TCustomAdapterErrorList is the base class for adapter components that display error lists. Do not instantiate TCustomAdapterErrorList directly. Use TAdapterErrorList (see page 6365), or create your own concrete descendant class.

65.1.25.1 TCustomAdapterErrorList Members

The following tables list the members exposed by TCustomAdapterErrorList.

Protected Methods

	Name	Description
	Content (see page 6447)	Returns the HTML that the Web item generates.
	GetContainerList (see page 6014)	Returns the TWebComponentList (see page 5999) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 6447)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
V	ImplContent (see page 6448)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	IsDefaultField (see page 6015)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Notification (see page 6448)	Forwards notification messages to all owned components.
V	ParentChanged (see page 6016)	Responds when the WebParent (see page 6018) property changes.
V	ReadState (see page 6016)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6017)	Sets the WebParent (see page 6018) property to be the same as this component's parent.
	SetWebParent (see page 6017)	Sets the value of the WebParent (see page 6018) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6018)	Provides access to the helper object that implements the IWebComponent (see page 6065) interface for this contained component.
	WebParent (see page 6018)	Indicates the component that uses this TWebContainedComponent (see page 6012) as one of its Web items.

Public Methods

	Name	Description
	Create (see page 6446)	Creates an instance of TCustomAdapterErrorList (see page 6445).
	Destroy (see page 6446)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6445).
	GetContainer (see page 6067)	Returns the value of the Container (see page 6069) property.
	GetIndex (see page 6067)	Returns the value of the Index (see page 6069) property.
	GetParentComponent (see page 6015)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6015)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6067)	Sets the value of the ComponentList (see page 6068) property.
	SetContainer (see page 6067)	Sets the value of the Container (see page 6069) property.
	SetIndex (see page 6068)	Sets the value of the Index (see page 6069) property.

Public Properties

	Name	Description
	Adapter (see page 6448)	Specifies a related adapter.
	ComponentList (see page 6068)	Indicates the component list that contains the Web item.
	Container (see page 6069)	Specifies the component that uses content generated by this Web item.
	Index (see page 6069)	Indicates where the Web item appears in the Web component list's array of Web items.

65.1.25.2 TCustomAdapterErrorList.Create Constructor

Creates an instance of TCustomAdapterErrorList (see page 6445).

Pascal

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

C++

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

Description

Call Create to create and initialize an instance of TCustomAdapterErrorList (see page 6445).

See Also

Create_d

65.1.25.3 TCustomAdapterErrorList.Destroy Destructor

Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6445).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterErrorList();
```

Description

Call Destroy to destroy an instance of TCustomAdapterErrorList (see page 6445) and free the memory associated with it.

65.1.25.4 TCustomAdapterErrorList Methods

The methods of the TCustomAdapterErrorList class are listed here.

Protected Methods

	Name	Description
	Content (see page 6447)	Returns the HTML that the Web item generates.
	GetDesigntimeWarnings (see page 6447)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
V	ImplContent (see page 6448)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	Notification (see page 6448)	Forwards notification messages to all owned components.

65.1.25.4.1 TCustomAdapterErrorList.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent interface, which the Web item uses to fit its own content into the parent's HTML structures.

Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 6057)

65.1.25.4.2 TCustomAdapterErrorList.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.

65.1.25.4.3 TCustomAdapterErrorList.ImplContent Method

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout Layout);
```

Description

This is ImplContent, a member of class TCustomAdapterErrorList.

65.1.25.4.4 TCustomAdapterErrorList.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

65.1.25.5 TCustomAdapterErrorList Properties

The properties of the TCustomAdapterErrorList class are listed here.

Public Properties

	Name	Description
	Adapter (see page 6448)	Specifies a related adapter.

65.1.25.5.1 TCustomAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

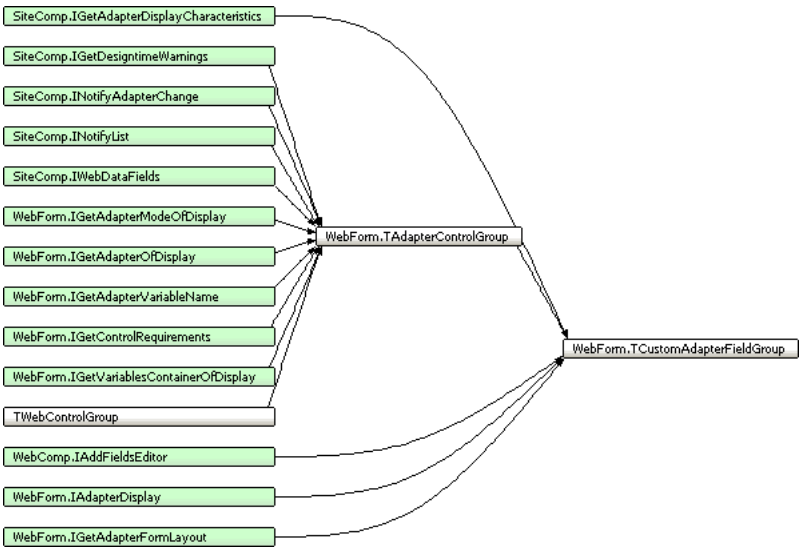
Description

Adapter specifies the adapter component for which this component reports errors.

65.1.26 TCustomAdapterFieldGroup Class

TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.

Class Hierarchy



Pascal

```
TCustomAdapterFieldGroup = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterFieldGroup : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterFieldGroup as a base class for components that generate organized portions of the HTML document containing one or more related HTML controls. TCustomAdapterFieldGroup introduces the properties and methods for managing a collection of Web items that generate the HTML controls that make up the group.

In order to let other Web items use the HTML that defines the organized collection of controls, TCustomAdapterFieldGroup implements the IWebContent (see page 6075) interface.

Do not create instances of TCustomAdapterFieldGroup. TCustomAdapterFieldGroup manages the Web items that generate

HTML defining controls, but does not combine the generated content into any content of its own. Instead, use a descendant of TCustomAdapterFieldGroup such as TAdapterFieldGroup (see page 6367).

See Also

TAdapterGrid (see page 6386)

TAdapterActionButton (see page 6286)

IWebContent (see page 6075)

65.1.26.1 TCustomAdapterFieldGroup Members

The following tables list the members exposed by TCustomAdapterFieldGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterDisplayCharacteristics (see page 6452)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6452)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6453)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6453)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6453)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetLayoutAttributes (see page 6453)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6454)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.
	ImplCanAddClass (see page 6454)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6454)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.

V	Create (see page 6451)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6449).
V	Destroy (see page 6451)	Frees an instance of TCustomAdapterFieldGroup (see page 6449).
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

Public Properties

	Name	Description
	Adapter (see page 6322)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6323)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6323)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6455)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6455)	Adds style attributes to the generated button group.
	StyleRule (see page 6456)	Applies a style in the page producer's style sheet to the button group.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

65.1.26.2 TCustomAdapterFieldGroup.Create Constructor

Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6449).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterFieldGroup (see page 6449) at runtime. TCustomAdapterFieldGroup (see page 6449) objects added at design time using the Web Page editor are instantiated automatically.

65.1.26.3 TCustomAdapterFieldGroup.Destroy Destructor

Frees an instance of TCustomAdapterFieldGroup (see page 6449).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterFieldGroup();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

65.1.26.4 TCustomAdapterFieldGroup Methods

The methods of the TCustomAdapterFieldGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 6452)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6452)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAddAllFieldsItem (see page 6453)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6453)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6453)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
V	GetLayoutAttributes (see page 6453)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6454)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
V	ImplCanAddClass (see page 6454)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6454)	Provides the underlying implementation for the Content method.

65.1.26.4.1 TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.

65.1.26.4.2 TCustomAdapterFieldGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.

65.1.26.4.3 TCustomAdapterFieldGroup.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.

65.1.26.4.4 TCustomAdapterFieldGroup.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.

65.1.26.4.5 TCustomAdapterFieldGroup.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.

65.1.26.4.6 TCustomAdapterFieldGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content

65.1.26.4.7 TCustomAdapterFieldGroup.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.

65.1.26.4.8 TCustomAdapterFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

65.1.26.4.9 TCustomAdapterFieldGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6057)

65.1.26.5 TCustomAdapterFieldGroup Properties

The properties of the TCustomAdapterFieldGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 6455)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6455)	Adds style attributes to the generated button group.
	StyleRule (see page 6456)	Applies a style in the page producer's style sheet to the button group.

65.1.26.5.1 TCustomAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 6455)

65.1.26.5.2 TCustomAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 6455)

StyleRule (see page 6456)

65.1.26.5.3 TCustomAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style (see page 6455) property.

See Also

Style (see page 6455)

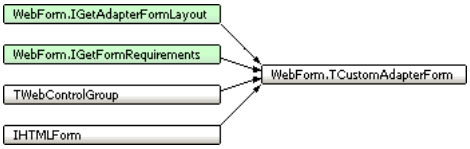
Styles (see page 1097)

StylesFile (see page 1098)

65.1.27 TCustomAdapterForm Class

TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.

Class Hierarchy



Pascal

```
TCustomAdapterForm = class(TWebControlGroup, IHTMLForm, IGetFormRequirements,
IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterForm : public TWebControlGroup, public IHTMLForm, public
IGetFormRequirements, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterForm as a base class when defining a form that can contain adapter fields and actions. TCustomAdapterForm is an abstract class, and should not be instantiated directly. Use TAdapterForm (see page 6373), which is a minimal implementation of TCustomAdapterForm, or create your own implementation.

See Also

TDataNavigator

TQueryButtons

TDataGrid

TFieldGroup

TQueryFieldGroup

IWebContent (see page 6075)

65.1.27.1 TCustomAdapterForm Members

The following tables list the members exposed by TCustomAdapterForm.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6458)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6458)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6459)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6459)	Returns the name of the javascript variable that refers to the generated form.
	GetLayoutAttributes (see page 6459)	Returns an object that specifies how the generated group of controls should look.
	ImplCanAddClass (see page 6460)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6460)	Provides the underlying implementation for the Content method.

Public Methods

	Name	Description
	Create (see page 6457)	Creates and initializes an instance of TCustomAdapterForm (see page 6456).
	Destroy (see page 6458)	Frees an instance of TCustomAdapterForm (see page 6456).

Public Properties

	Name	Description
	Custom (see page 6461)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6461)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6462)	Adds style attributes to the generated form.
	StyleRule (see page 6462)	Applies a style in the page producer's style sheet to the form.

65.1.27.2 TCustomAdapterForm.Create Constructor

Creates and initializes an instance of TCustomAdapterForm (see page 6456).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterForm (see page 6456) at runtime. TCustomAdapterForm (see page 6456) objects added at design time using the Web Page editor are instantiated automatically.

65.1.27.3 TCustomAdapterForm.Destroy Destructor

Frees an instance of TCustomAdapterForm (see page 6456).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterForm();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.27.4 TCustomAdapterForm Methods

The methods of the TCustomAdapterForm class are listed here.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6458)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6458)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6459)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6459)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 6459)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6460)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6460)	Provides the underlying implementation for the Content method.

65.1.27.4.1 TCustomAdapterForm.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterForm.

65.1.27.4.2 TCustomAdapterForm.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class TCustomAdapterForm.

65.1.27.4.3 TCustomAdapterForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormName();
```

Description

GetHTMLFormName is the read access method for the HTMLFormName property.

See Also

HTMLFormName

65.1.27.4.4 TCustomAdapterForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

GetHTMLFormVarName is the read access method for the HTMLFormVarName property.

See Also

HTMLFormVarName

65.1.27.4.5 TCustomAdapterForm.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content

65.1.27.4.6 TCustomAdapterForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

65.1.27.4.7 TCustomAdapterForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6057)

65.1.27.5 TCustomAdapterForm Properties

The properties of the TCustomAdapterForm class are listed here.

Public Properties

	Name	Description
	Custom (see page 6461)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6461)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6462)	Adds style attributes to the generated form.
	StyleRule (see page 6462)	Applies a style in the page producer's style sheet to the form.

65.1.27.5.1 TCustomAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 6462)

65.1.27.5.2 TCustomAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

65.1.27.5.3 TCustomAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 6461)

StyleRule (see page 6462)

65.1.27.5.4 TCustomAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style (see page 6462) property.

See Also

Style (see page 6462)

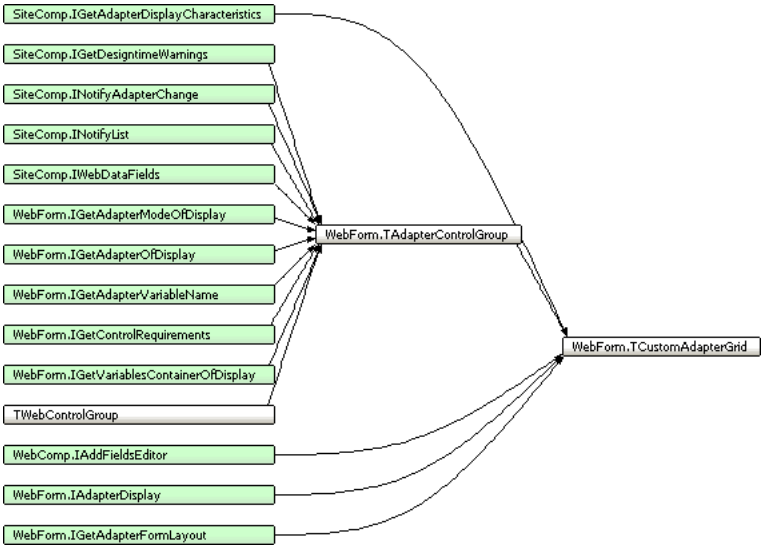
Styles (see page 1097)

StylesFile (see page 1098)

65.1.28 TCustomAdapterGrid Class

TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

Class Hierarchy



Pascal

```
TCustomAdapterGrid = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterGrid : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterGrid as a base class for Web items that generate an HTML table to display multiple records of data supplied by an adapter component. All of the adapter grid's child Web items display field components that use the same adapter, which is specified by a property of the adapter grid.

If no display fields are explicitly added to the adapter grid, it displays all of the adapter's display fields by default. The properties of default display fields cannot be modified.

TCustomAdapterGrid introduces properties to

Specify the associated Adapter (see page 6322)

Specify whether default fields should be added

Customize the appearance of the HTML table.

Do not create instances of TCustomAdapterGrid. Instead, use a descendant of TCustomAdapterGrid such as TAdapterGrid (see page 6386).

See Also

TAdapterCommandColumn (see page 6301)

TAdapterGrid (see page 6386)

TAdapterFieldGroup (see page 6367)

IWebContent (see page 6075)

65.1.28.1 TCustomAdapterGrid Members

The following tables list the members exposed by TCustomAdapterGrid.

Protected Methods

	Name	Description
	AdapterChange (see page 6315)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6315)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6316)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6316)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6317)	Returns the value of the Adapter (see page 6322) property.
	GetAdapterDisplayCharacteristics (see page 6466)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6466)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 6317)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6317)	Returns the value of the AdapterMode (see page 6323) property.
	GetAdapterVariableName (see page 6318)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6466)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6467)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6467)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetControlRequirements (see page 6318)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6318)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6319)	Fills a string list with the names of all of the associated adapter's fields.
	GetNewFieldPrefix (see page 6467)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
	GetVisibleFields (see page 6319)	Returns the value of the VisibleFields (see page 6324) property.
	ImplCanAddClass (see page 6467)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6468)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6320)	Provides the underlying implementation for the GetControlRequirements (see page 6318) method.
	IsFieldInUse (see page 6320)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6320)	Updates the TAdapterControlGroup (see page 6312) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6321)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6321)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6322)	Sets the value of the Adapter (see page 6322) property.

Public Methods

	Name	Description
	AddNotify (see page 1000)	Registers an interest in change notifications.
	Create (see page 6465)	Creates and initializes an instance of TCustomAdapterGrid (see page 6462).
	Destroy (see page 6465)	Frees an instance of TAdapterControlGroup.
	FindField (see page 1023)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 971)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetFieldsList (see page 1024)	This is GetFieldsList, a member of class IWebDataFields.
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1024)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1024)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 999)	Responds when the adapter changes.
	RemoveNotify (see page 1000)	Unregisters interest in change notifications.

Public Properties

	Name	Description
	Adapter (see page 6322)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6323)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6323)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6468)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6469)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6469)	Determines the basic display characteristics of the table.
R	VisibleFields (see page 1025)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6324)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

65.1.28.2 TCustomAdapterGrid.Create Constructor

Creates and initializes an instance of TCustomAdapterGrid (see page 6462).

Pascal

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

C++

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

Description

Call Create to instantiate TCustomAdapterGrid (see page 6462) at runtime. TCustomAdapterGrid (see page 6462) objects added at design time using the Web Page editor are instantiated automatically.

65.1.28.3 TCustomAdapterGrid.Destroy Destructor

Frees an instance of TAdapterControlGroup.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterGrid();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

65.1.28.4 TCustomAdapterGrid Methods

The methods of the TCustomAdapterGrid class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 6466)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6466)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAddAllFieldsItem (see page 6466)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6467)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6467)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetNewFieldPrefix (see page 6467)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
V	ImplCanAddClass (see page 6467)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6468)	Provides the underlying implementation for the Content method.

65.1.28.4.1 TCustomAdapterGrid.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.

65.1.28.4.2 TCustomAdapterGrid.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.

65.1.28.4.3 TCustomAdapterGrid.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.

65.1.28.4.4 TCustomAdapterGrid.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.

65.1.28.4.5 TCustomAdapterGrid.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterGrid.

65.1.28.4.6 TCustomAdapterGrid.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.

65.1.28.4.7 TCustomAdapterGrid.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

65.1.28.4.8 TCustomAdapterGrid.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6057)

65.1.28.5 TCustomAdapterGrid Properties

The properties of the TCustomAdapterGrid class are listed here.

Public Properties

	Name	Description
	HeadingAttributes (see page 6468)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6469)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6469)	Determines the basic display characteristics of the table.

65.1.28.5.1 TCustomAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table.

HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes (see page 6469) or RowAttributes (see page 6469) property instead.

See Also

TableAttributes (see page 6469)

RowAttributes (see page 6469)

65.1.28.5.2 TCustomAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes (see page 6469) or HeadingAttributes (see page 6468) property instead.

See Also

TableAttributes (see page 6469)

HeadingAttributes (see page 6468)

65.1.28.5.3 TCustomAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes (see page 6468) property instead.

See Also

RowAttributes (see page 6469)

HeadingAttributes (see page 6468)

65.1.29 TDesigntimeWarnings Class

Class Hierarchy

Pascal

```
TDesigntimeWarnings = class(TAbstractDesigntimeWarnings);
```

C++

```
class TDesigntimeWarnings : public TAbstractDesigntimeWarnings;
```

File

WebForm

Description

This is class WebForm.TDesigntimeWarnings.

65.1.29.1 TDesigntimeWarnings Members

The following tables list the members exposed by TDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 6471)	Requests any design time warnings from a specified object.
	AddString (see page 6471)	Adds a warning string to the set of warnings that appear at design time.
	Create (see page 6470)	This is Create, a member of class TDesigntimeWarnings.
	Destroy (see page 6470)	This is Destroy, a member of class TDesigntimeWarnings.

Public Properties

	Name	Description
	Strings (see page 6472)	This is Strings, a member of class TDesigntimeWarnings.

65.1.29.2 TDesigntimeWarnings.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDesigntimeWarnings();
```

Description

This is Create, a member of class TDesigntimeWarnings.

65.1.29.3 TDesigntimeWarnings.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDesigntimeWarnings();
```

Description

This is Destroy, a member of class TDesigntimeWarnings.

65.1.29.4 TDesigntimeWarnings Methods

The methods of the TDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	AddObject (see page 6471)	Requests any design time warnings from a specified object.
	AddString (see page 6471)	Adds a warning string to the set of warnings that appear at design time.

65.1.29.4.1 TDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); override;
```

C++

```
virtual __fastcall AddObject(TObject * AObject);
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 6353)

AddString (see page 906)

65.1.29.4.2 TDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); override;
```

C++

```
virtual __fastcall AddString(UnicodeString AWarning);
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 1076)

AddObject (see page 906)

65.1.29.5 TDesigntimeWarnings Properties

The properties of the TDesigntimeWarnings class are listed here.

Public Properties

	Name	Description
	Strings (see page 6472)	This is Strings, a member of class TDesigntimeWarnings.

65.1.29.5.1 TDesigntimeWarnings.Strings Property

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

This is Strings, a member of class TDesigntimeWarnings.

65.1.30 TGeneratedFunctions Class

Class Hierarchy

WebForm.TGeneratedFunctions

Pascal

```
TGeneratedFunctions = class;
```

C++

```
class TGeneratedFunctions;
```

File

WebForm

Description

This is class WebForm.TGeneratedFunctions.

65.1.30.1 TGeneratedFunctions Members

The following tables list the members exposed by TGeneratedFunctions.

Public Methods

	Name	Description
	Add (see page 6473)	This is Add, a member of class TGeneratedFunctions.

	Create (see page 6473)	This is Create, a member of class TGeneratedFunctions.
V	Destroy (see page 6473)	This is Destroy, a member of class TGeneratedFunctions.
	Exists (see page 6474)	This is Exists, a member of class TGeneratedFunctions.

Public Properties

	Name	Description
R	ItemCount (see page 6474)	This is ItemCount, a member of class TGeneratedFunctions.
R	Names (see page 6474)	This is Names, a member of class TGeneratedFunctions.
R	Script (see page 6474)	This is Script, a member of class TGeneratedFunctions.

65.1.30.2 TGeneratedFunctions.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TGeneratedFunctions();
```

Description

This is Create, a member of class TGeneratedFunctions.

65.1.30.3 TGeneratedFunctions.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TGeneratedFunctions();
```

Description

This is Destroy, a member of class TGeneratedFunctions.

65.1.30.4 TGeneratedFunctions Methods

The methods of the TGeneratedFunctions class are listed here.

Public Methods

	Name	Description
	Add (see page 6473)	This is Add, a member of class TGeneratedFunctions.
	Exists (see page 6474)	This is Exists, a member of class TGeneratedFunctions.

65.1.30.4.1 TGeneratedFunctions.Add Method

Pascal

```
procedure Add(const AName: string; const AScript: string);
```

C++

```
__fastcall Add(const UnicodeString AName, const UnicodeString AScript);
```

Description

This is Add, a member of class TGeneratedFunctions.

65.1.30.4.2 TGeneratedFunctions.Exists Method

Pascal

```
function Exists(const AName: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const UnicodeString AName);
```

Description

This is Exists, a member of class TGeneratedFunctions.

65.1.30.5 TGeneratedFunctions Properties

The properties of the TGeneratedFunctions class are listed here.

Public Properties

	Name	Description
	ItemCount (see page 6474)	This is ItemCount, a member of class TGeneratedFunctions.
	Names (see page 6474)	This is Names, a member of class TGeneratedFunctions.
	Script (see page 6474)	This is Script, a member of class TGeneratedFunctions.

65.1.30.5.1 TGeneratedFunctions.ItemCount Property

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

This is ItemCount, a member of class TGeneratedFunctions.

65.1.30.5.2 TGeneratedFunctions.Names Property

Pascal

```
property Names [I: Integer]: string;
```

C++

```
__property UnicodeString Names[int I];
```

Description

This is Names, a member of class TGeneratedFunctions.

65.1.30.5.3 TGeneratedFunctions.Script Property

Pascal

```
property Script [I: Integer]: string;
```

C++

```
__property UnicodeString Script[int I];
```

Description

This is Script, a member of class TGeneratedFunctions.

65.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterDisplay (see page 6476)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 6476)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 6477)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 6478)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 6479)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 6481)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 6482)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 6483)	This is class WebForm.IGetVariablesContainerOfDisplay.

65.2.1 IAdapterDisplay Interface

Class Hierarchy

WebForm.IAdapterDisplay

Pascal

```
IAdapterDisplay = interface;
```

C++

```
__interface IAdapterDisplay;
```

File

WebForm

Description

This is class WebForm.IAdapterDisplay.

65.2.2 IGetAdapterFormLayout Interface

Class Hierarchy

WebForm.IGetAdapterFormLayout

Pascal

```
IGetAdapterFormLayout = interface;
```

C++

```
__interface IGetAdapterFormLayout;
```

File

WebForm

Description

This is class WebForm.IGetAdapterFormLayout.

65.2.2.1 IGetAdapterFormLayout Members

The following tables list the members exposed by IGetAdapterFormLayout.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

65.2.2.2 IGetAdapterFormLayout Methods

The methods of the IGetAdapterFormLayout class are listed here.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 6477)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

65.2.2.2.1 IGetAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

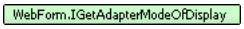
```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

65.2.3 IGetAdapterModeOfDisplay Interface

Class Hierarchy



Pascal

```
IGetAdapterModeOfDisplay = interface;
```

C++

```
__interface IGetAdapterModeOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterModeOfDisplay.

65.2.3.1 IGetAdapterModeOfDisplay Members

The following tables list the members exposed by IGetAdapterModeOfDisplay.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

65.2.3.2 IGetAdapterModeOfDisplay Methods

The methods of the IGetAdapterModeOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 6478)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6478)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

65.2.3.2.1 IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.

65.2.3.2.2 IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay Method

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

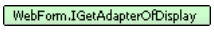
```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

65.2.4 IGetAdapterOfDisplay Interface

Class Hierarchy



Pascal

```
IGetAdapterOfDisplay = interface;
```

C++

```
__interface IGetAdapterOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterOfDisplay.

65.2.4.1 IGetAdapterOfDisplay Members

The following tables list the members exposed by IGetAdapterOfDisplay.

Public Methods

	Name	Description
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

65.2.4.2 IGetAdapterOfDisplay Methods

The methods of the IGetAdapterOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapter (see page 6479)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

65.2.4.2.1 IGetAdapterOfDisplay.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

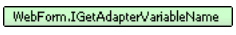
```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class IGetAdapterOfDisplay.

65.2.5 IGetAdapterVariableName Interface

Class Hierarchy



Pascal

```
IGetAdapterVariableName = interface;
```

C++

```
__interface IGetAdapterVariableName;
```

File

WebForm

Description

This is class WebForm.IGetAdapterVariableName.

65.2.5.1 IGetAdapterVariableName Members

The following tables list the members exposed by IGetAdapterVariableName.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

65.2.5.2 IGetAdapterVariableName Methods

The methods of the IGetAdapterVariableName class are listed here.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 6480)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

65.2.5.2.1 IGetAdapterVariableName.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

65.2.5.3 IGetAdapterVariableName Properties

The properties of the IGetAdapterVariableName class are listed here.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6481)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

65.2.5.3.1 IGetAdapterVariableName.AdapterVariableName Property

Pascal

```
property AdapterVariableName: string;
```

C++

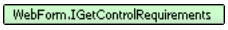
```
__property UnicodeString AdapterVariableName;
```

Description

This is AdapterVariableName, a member of class IGetAdapterVariableName.

65.2.6 IGetControlRequirements Interface

Class Hierarchy



Pascal

```
IGetControlRequirements = interface;
```

C++

```
__interface IGetControlRequirements;
```

File

WebForm

Description

This is class WebForm.IGetControlRequirements.

65.2.6.1 IGetControlRequirements Members

The following tables list the members exposed by IGetControlRequirements.

Public Methods

	Name	Description
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.

65.2.6.2 IGetControlRequirements Methods

The methods of the IGetControlRequirements class are listed here.

Public Methods

	Name	Description
	GetControlRequirements (see page 6482)	This is GetControlRequirements, a member of class IGetControlRequirements.

65.2.6.2.1 IGetControlRequirements.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

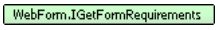
```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class IGetControlRequirements.

65.2.7 IGetFormRequirements Interface

Class Hierarchy



Pascal

```
IGetFormRequirements = interface;
```

C++

```
__interface IGetFormRequirements;
```

File

WebForm

Description

This is class WebForm.IGetFormRequirements.

65.2.7.1 IGetFormRequirements Members

The following tables list the members exposed by IGetFormRequirements.

Public Methods

	Name	Description
	GetFormRequirements (see page 6483)	This is GetFormRequirements, a member of class IGetFormRequirements.

65.2.7.2 IGetFormRequirements Methods

The methods of the IGetFormRequirements class are listed here.

Public Methods

	Name	Description
	GetFormRequirements (see page 6483)	This is GetFormRequirements, a member of class IGetFormRequirements.

65.2.7.2.1 IGetFormRequirements.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

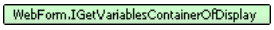
```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class IGetFormRequirements.

65.2.8 IGetVariablesContainerOfDisplay Interface

Class Hierarchy



Pascal

```
IGetVariablesContainerOfDisplay = interface;
```

C++

```
__interface IGetVariablesContainerOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetVariablesContainerOfDisplay.

65.2.8.1 IGetVariablesContainerOfDisplay Members

The following tables list the members exposed by IGetVariablesContainerOfDisplay.

Public Methods

	Name	Description
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

65.2.8.2 IGetVariablesContainerOfDisplay Methods

The methods of the IGetVariablesContainerOfDisplay class are listed here.

Public Methods

	Name	Description
	GetVariablesContainer (see page 6484)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

65.2.8.2.1 IGetVariablesContainerOfDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

65.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 6485)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 6485)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 6486)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 6486)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 6486)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 6487)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 6487)	This is function WebForm.MakeAdapterVariableName.

65.3.1 WebForm.CalcDisplayFieldHTMLElementType Function

Pascal

```
function CalcDisplayFieldHTMLElementType(AField: TComponent; const AAdapterMode: string;
ADisplayType: TDisplayFieldHTMLElementType; var AType: TDisplayFieldHTMLElementType): Boolean;
```

C++

```
Boolean CalcDisplayFieldHTMLElementType(TComponent * AField, const UnicodeString AAdapterMode,
TDisplayFieldHTMLElementType ADisplayType, TDisplayFieldHTMLElementType AType);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldHTMLElementType.

65.3.2 WebForm.CalcDisplayFieldViewMode Function

Pascal

```
function CalcDisplayFieldViewMode(AField: TComponent; const AAdapterMode: string; AViewMode:
TDisplayFieldViewMode; var AMode: TDisplayFieldViewMode): Boolean;
```

C++

```
Boolean CalcDisplayFieldViewMode(TComponent * AField, const UnicodeString AAdapterMode,
TDisplayFieldViewMode AViewMode, TDisplayFieldViewMode AMode);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldViewMode.

65.3.3 WebForm.FindVariablesContainer Function

Pascal

```
function FindVariablesContainer(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainer(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainer.

65.3.4 WebForm.FindVariablesContainerInParent Function

Pascal

```
function FindVariablesContainerInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainerInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainerInParent.

65.3.5 WebForm.GetAdapterModeOfAdapterInParent Function

Pascal

```
function GetAdapterModeOfAdapterInParent(AComponent: TComponent): string;
```

C++

```
UnicodeString GetAdapterModeOfAdapterInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.GetAdapterModeOfAdapterInParent.

65.3.6 WebForm.MakeAdapterHTMLName Function

Pascal

```
function MakeAdapterHTMLName(AVarName: string): string;
```

C++

```
UnicodeString MakeAdapterHTMLName(UnicodeString AVarName);
```

File

WebForm

Description

This is function WebForm.MakeAdapterHTMLName.

65.3.7 WebForm.MakeAdapterVariableName Function

Pascal

```
function MakeAdapterVariableName(AComponent: TComponent): string;
```

C++

```
UnicodeString MakeAdapterVariableName(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.MakeAdapterVariableName.

65.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActionButtonHideOption (see page 6488)	TActionButtonHideOptions (see page 6494) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 6489)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 6489)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 6490)	TDisplayFieldHideOptions (see page 6494) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 6491)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 6491)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 6492)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 6493)	This is record WebForm.TLayoutState.

65.4.1 WebForm.TActionButtonHideOption Enumeration

TActionButtonHideOptions (see page 6494) indicates when a control to represent an adapter action should not be generated.

Pascal

```
TActionButtonHideOption = (
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
);
```

C++

```
enum TActionButtonHideOption {
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
};
```

File

WebForm

Description

TActionButtonHideOptions (see page 6494) is a set of TActionButtonHideOption values. It can include zero or more of the following values:

Value	Meaning
bhoHideOnNoExecuteAccess	Suppress the HTML control when the user does not have rights to execute the associated action, as indicated by the action's HasExecuteAccess method. When this flag is set, server-side script can suppress the control by setting the action's CanExecute property to false.

bhoHideOnActionNotVisible	Suppress the HTML control when the action returns false from its GetVisible method. When this flag is set, server-side script can suppress the control by setting the action's Visible property to false. This flag is set by default.
bhoHideOnDisabledAction	Suppress the HTML control when the action returns false from its WebEnabled method. When this flag is set, server-side script can suppress the control by setting the action's Enabled property to false.
bhoHideAlways	Always suppress the HTML control.

65.4.2 WebForm.TColumnHTMLElementType Enumeration

Pascal

```
TColumnHTMLElementType = (
  coltDefault,
  coltText,
  coltImage,
  coltList
);
```

C++

```
enum TColumnHTMLElementType {
  coltDefault,
  coltText,
  coltImage,
  coltList
};
```

File

WebForm

Description

This is record WebForm.TColumnHTMLElementType.

65.4.3 WebForm.TCommandHTMLElementType Enumeration

TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.

Pascal

```
TCommandHTMLElementType = (
  ctDefault,
  ctButton,
  ctImage,
  ctAnchor,
  ctEventImages
);
```

C++

```
enum TCommandHTMLElementType {
  ctDefault,
  ctButton,
  ctImage,
  ctAnchor,
```

```
  ctEventImages
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TCommandHTMLElementType defines HTML elements that the user can click to execute a command.

The following table lists the possible values:

Value	Meaning
ctDefault	Generate the element type that the adapter action object suggests.
ctButton	Generate a push button with a text label.
ctImage	Generate an image that the user can click.
ctAnchor	Generate a hypertext link.
ctEventImages	Generate an element that displays a different image, depending on the users action. That is, the image changes dynamically depending on user events such as whether the mouse is over the control, whether the user is clicking the control, and so on.

See Also

DisplayType

65.4.4 WebForm.TDisplayFieldHideOption Enumeration

TDisplayFieldHideOptions (see page 6494) indicates when a control to represent a dataset field should not be generated.

Pascal

```
TDisplayFieldHideOption = (
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
);
```

C++

```
enum TDisplayFieldHideOption {
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
};
```

File

WebForm

Description

TDisplayFieldHideOptions (see page 6494) is a set of TDisplayFieldHideOption values. It can include zero or more of the following values:

Value	Meaning
hoHideOnNoInputAccess	Do not generate a control if the user does not have permission to modify this field.
hoHideOnNoDisplayAccess	Do not generate a control if the user does not have permission to view this field.
hoHideOnFieldNotVisable	Do not generate a control if the field is marked as hidden.
hoHideAlways	Never generate a control for the field.

65.4.5 WebForm.TDisplayFieldHTMLElementType Enumeration

TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TDisplayFieldHTMLElementType = (
  dftDefault,
  dftText,
  dftImage,
  dftList
);
```

C++

```
enum TDisplayFieldHTMLElementType {
  dftDefault,
  dftText,
  dftImage,
  dftList
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TDisplayFieldHTMLElementType defines read-only HTML elements.

The following table lists the possible values:

Value	Meaning
dftDefault	Use the default display type of the component that generates the HTML control
dftText	Display as a text string.
dftImage	Display as an image.
dftList	Display as a list of values.

65.4.6 WebForm.TDisplayFieldViewMode Enumeration

TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.

Pascal

```
TDisplayFieldViewMode = (
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
);
```

C++

```
enum TDisplayFieldViewMode {
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
};
```

File

WebForm

Description

The following table lists the possible values for the TDisplayFieldViewMode type and the meaning corresponding to each:

Value	Meaning
vmDefault	Use the default mode associated with the adapter.
vmInput	Generate an input control for the field.
vmDisplay	Generate a read-only control for the field.
vmToggleOnAccess	Generate a control that toggles between read-only and input mode when the user tries to access it.

65.4.7 WebForm.TInputFieldHTMLElementType Enumeration

TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TInputFieldHTMLElementType = (
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
  iftRadio,
  iftCheckBox,
  iftTextArea,
  iftFile
);
```

C++

```
enum TInputFieldHTMLElementType {
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
  iftRadio,
  iftCheckBox,
  iftTextArea,
```

```
  iftFile
};
```

File

WebForm

Description

TInputFieldHTMLElementType defines the set of possible HTML controls that can be used to input an adapter field value. The following table lists the possible values for the TInputFieldHTMLElementType type and the meaning corresponding to each flag:

Flag	Input (see page 3895) field type
iftCheckBox	The input field is a check box.
iftDefault	The input field is the default type.
iftFile	The input field is a file.
iftPasswordInput	The input field is a password prompt.
iftRadio	The input field is a radio button.
iftSelect	The input field is selectable. iftSelect is only used with iftCheckBox and iftRadio.
iftSelectMultiple	More than one input field may be selected at a time. iftSelectMultiple is only used with iftCheckBox and iftRadio.
iftTextArea	The field is a text area.
iftTextInput	The input field is a text input box.

65.4.8 WebForm.TLayoutState Enumeration

Pascal

```
TLayoutState = (
  lsUnknown,
  lsFields,
  lsButtons
);
```

C++

```
enum TLayoutState {
  lsUnknown,
  lsFields,
  lsButtons
};
```

File

WebForm

Description

This is record WebForm.TLayoutState.

65.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActionButtonHideOptions (see page 6494)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 6494)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 6494)	This is type WebForm.TDisplayFieldHideOptions.

65.5.1 WebForm.TActionButtonHideOptions Type

Pascal

```
TActionButtonHideOptions = set of TActionButtonHideOption;
```

C++

```
TActionButtonHideOption TActionButtonHideOptions;
```

File

WebForm

Description

This is type WebForm.TActionButtonHideOptions.

65.5.2 WebForm.TAdapterDataDisplayClass Type

Pascal

```
TAdapterDataDisplayClass = class of TAdapterDataDisplay;
```

C++

```
TAdapterDataDisplay TAdapterDataDisplayClass;
```

File

WebForm

Description

This is type WebForm.TAdapterDataDisplayClass.

65.5.3 WebForm.TDisplayFieldHideOptions Type

Pascal

```
TDisplayFieldHideOptions = set of TDisplayFieldHideOption;
```

C++

```
TDisplayFieldHideOption TDisplayFieldHideOptions;
```

File

WebForm

Description

This is type WebForm.TDisplayFieldHideOptions.

65.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sDisplayText (see page 6496)	This is constant WebForm.sDisplayText.
sFieldText (see page 6496)	This is constant WebForm.sFieldText.
sFormGet (see page 6497)	This is constant WebForm.sFormGet.
sFormPost (see page 6497)	This is constant WebForm.sFormPost.
sMultipartForm (see page 6497)	This is constant WebForm.sMultipartForm.

65.6.1 WebForm.sDisplayText Constant

Pascal

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

C++

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

File

WebForm

Description

This is constant WebForm.sDisplayText.

65.6.2 WebForm.sFieldText Constant

Pascal

```
sFieldText = '<%%= %0:s.EditText %%>';
```

C++

```
sFieldText = '<%%= %0:s.EditText %%>';
```

File

WebForm

Description

This is constant WebForm.sFieldText.

65.6.3 WebForm.sFormGet Constant

Pascal

```
sFormGet = 'get';
```

C++

```
sFormGet = 'get';
```

File

WebForm

Description

This is constant WebForm.sFormGet.

65.6.4 WebForm.sFormPost Constant

Pascal

```
sFormPost = 'post';
```

C++

```
sFormPost = 'post';
```

File

WebForm

Description

This is constant WebForm.sFormPost.

65.6.5 WebForm.sMultipartForm Constant

Pascal

```
sMultipartForm = 'multipart/form-data';
```

C++

```
sMultipartForm = 'multipart/form-data';
```

File

WebForm

Description

This is constant WebForm.sMultipartForm.

66 WebModu Namespace

This is namespace WebModu.

Topics

Name	Description
Classes (see page 6501)	The following table lists classes in this documentation.
Functions (see page 6578)	The following table lists functions in this documentation.
Types (see page 6580)	The following table lists types in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 6501)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 6502)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 6505)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 6510)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 6516)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 6522)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 6531)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 6539)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 6550)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 6554)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 6560)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 6562)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 6565)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 6569)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 6574)	This is class WebModu.TWebPageModuleHelper.

Functions

	Name	Description
	FindNamedModuleVariable (see page 6578)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 6578)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 6578)	This is function WebModu.GetModuleVariableCount.

Types

Name	Description
TGetParamsEvent (see page 6580)	This is type WebModu.TGetParamsEvent.

66.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 6501)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 6502)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 6505)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 6510)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 6516)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 6522)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 6531)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 6539)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 6550)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 6554)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 6560)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 6562)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 6565)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 6569)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 6574)	This is class WebModu.TWebPageModuleHelper.

66.1.1 EWebModuleException Class

EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.

Class Hierarchy

Pascal

```
EWebModuleException = class(EWebBrokerException);
```

C++

```
class EWebModuleException : public EWebBrokerException;
```

File

WebModu

Description

EWebModuleException is the exception that is raised when a server-side script in a WebSnap application tries to do direct a Web module to perform a task it does not know how to handle. For example, Web modules raise this exception when a script

tries to access a variable in a Web module that does not support the use of variables.

See Also

TWebDataModule (see page 6562)

TWebAppDataModule (see page 6550)

TWebPageModule (see page 6569)

TWebAppPageModule (see page 6554)

66.1.2 TAdapterActionName Class

TAdapterActionName describes an adapter action by name.

Class Hierarchy

Pascal

```
TAdapterActionName = class(TPersistent);
```

C++

```
class TAdapterActionName : public TPersistent;
```

File

WebModu

Description

TAdapterAction represents a TAdapterAction object. It is used when the action is referenced by name rather than directly as a component. TAdapterAction publishes properties that specify the name of the action and the parent adapter component to which it belongs. The read-only Action (see page 6504) property provides access to the action component with the specified name that belongs to the specified adapter.

See Also

TAdapter (see page 5179)

TAdapterAction (see page 5190)

66.1.2.1 TAdapterActionName Members

The following tables list the members exposed by TAdapterActionName.

Protected Methods

	Name	Description
	AssignTo (see page 6503)	Copies the properties of an object to a destination object.

Public Properties

	Name	Description
	Action (see page 6504)	Indicates the action that this object specifies.

Published Methods

	Name	Description
	Create (see page 6503)	Creates and initializes an instance of TAdapterActionName (see page 6502).

Published Properties

	Name	Description
	ActionName (see page 6505)	Specifies the name of the action this object represents.
	Adapter (see page 6505)	Specifies the adapter to which the action belongs.

66.1.2.2 TAdapterActionName.Create Constructor

Creates and initializes an instance of TAdapterActionName (see page 6502).

Pascal

```
constructor Create(AModule: TComponent);
```

C++

```
__fastcall TAdapterActionName(TComponent * AModule);
```

Description

Call Create to instantiate TAdapterActionName (see page 6502).

AModule is the Web module instance that should be notified if the adapter specified by the Adapter (see page 6505) property is deleted. The TAdapterActionName (see page 6502) constructor calls this Adapter (see page 6505)'s FreeNotification method so that the adapter will notify the Web module if it (and its actions) are freed.

See Also

FreeNotification

Adapter (see page 6505)

66.1.2.3 TAdapterActionName Methods

The methods of the TAdapterActionName class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 6503)	Copies the properties of an object to a destination object.

66.1.2.3.1 TAdapterActionName.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should

be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3985)

66.1.2.4 TAdapterActionName Properties

The properties of the TAdapterActionName class are listed here.

Public Properties

	Name	Description
R	Action (see page 6504)	Indicates the action that this object specifies.

Published Properties

	Name	Description
	ActionName (see page 6505)	Specifies the name of the action this object represents.
	Adapter (see page 6505)	Specifies the adapter to which the action belongs.

66.1.2.4.1 TAdapterActionName.Action Property

Indicates the action that this object specifies.

Pascal

```
property Action: TComponent;
```

C++

```
__property TComponent * Action;
```

Description

Read (see page 3727) Action to access the adapter action component specified by the Adapter (see page 6505) and ActionName (see page 6505) properties. That is, Action is the action with the name specified by ActionName (see page 6505) that belongs to the adapter specified by Adapter (see page 6505).

See Also

ActionName (see page 6505)

Adapter (see page 6505)

TAdapterAction (see page 5190)

66.1.2.4.2 TAdapterActionName.ActionName Property

Specifies the name of the action this object represents.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

Set ActionName to the name of the desired action. This is the name of one of the actions that belongs to the adapter that the Adapter (see page 6505) property specifies.

See Also

Action (see page 6504)

Adapter (see page 6505)

Actions (see page 5558)

66.1.2.4.3 TAdapterActionName.Adapter Property

Specifies the adapter to which the action belongs.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to the adapter component to which the named action belongs. The TAdapterActionName (see page 6502) object represents the action that this adapter returns when you call its FindAction method, passing in the value of the ActionName (see page 6505) property.

See Also

Action (see page 6504)

ActionName (see page 6505)

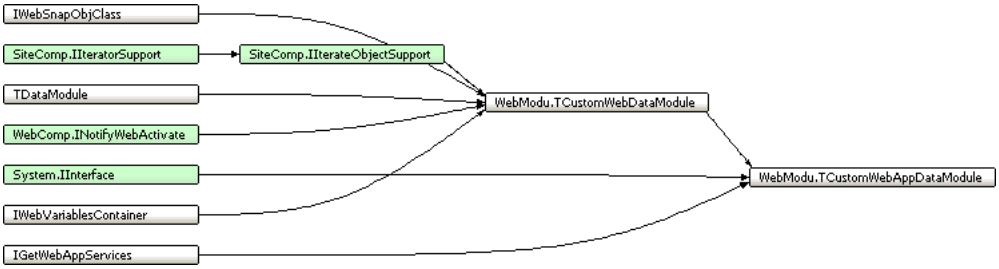
Actions (see page 5558)

FindAction (see page 5530)

66.1.3 TCustomWebAppDataModule Class

TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.

Class Hierarchy



Pascal

```
TCustomWebAppDataModule = class(TCustomWebDataModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppDataModule : public TCustomWebDataModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppDataModule as a base class for defining data modules that function as the main Web module in a WebSnap application. As the application module, TCustomWebAppDataModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling.

If you want an application module that also has an associated HTML page, then derive from TCustomWebAppPageModule (see page 6510) instead.

See Also

TWebAppDataModule (see page 6550)

66.1.3.1 TCustomWebAppDataModule Members

The following tables list the members exposed by TCustomWebAppDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetWebAppServices (see page 6508)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6509)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6509)	Returns a reference to a specified interface if the module supports that interface.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Public Events

	Name	Description
	OnActivate (see page 6522)	Occurs when the Web module is activated.
	OnDeactivate (see page 6522)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	Create (see page 6507)	Creates a new TCustomWebAppDataModule (see page 6505) instance.
V	CreateNew (see page 6507)	Creates a data module without using the associated form file to initialize it.
V	Destroy (see page 6508)	Frees a Web application data module instance.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	AppServices (see page 6510)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6505) object.
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

66.1.3.2 TCustomWebAppDataModule.Create Constructor

Creates a new TCustomWebAppDataModule (see page 6505) instance.

Pascal

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

C++

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

Description

Use this constructor to create a new TCustomWebAppDataModule (see page 6505). WebSnap modules are registered with a factory that is responsible for creating modules as needed. The module's source unit defines a function that returns an instance of the module.

AOwner is the component that becomes the value of the Owner property.

66.1.3.3 TCustomWebAppDataModule.CreateNew Constructor

Creates a data module without using the associated form file to initialize it.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6507) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

66.1.3.4 TCustomWebAppDataModule.Destroy Destructor

Frees a Web application data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppDataModule();
```

Description

```
destructor Destroy; override;
```

Destroy is the destructor for the object instance. It clears the AppServices (see page 6510) property and calls the inherited destructor. The AppServices (see page 6510) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 6510) may also be owned by a Web page module.

66.1.3.5 TCustomWebAppDataModule Methods

The methods of the TCustomWebAppDataModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 6508)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 6509)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 6509)	Returns a reference to a specified interface if the module supports that interface.

66.1.3.5.1 TCustomWebAppDataModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this TCustomWebDataModule (see page 6516). This is the same as the value of the AppServices (see page 6510) property.

See Also

AppServices (see page 6510)

66.1.3.5.2 TCustomWebAppDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppDataModule (see page 6505) uses this method to check whether the component that implements AppServices (see page 6510) is about to be removed, and if so, sets the value of the AppServices (see page 6510) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 6510)

66.1.3.5.3 TCustomWebAppDataModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 3504)

66.1.3.6 TCustomWebAppDataModule Properties

The properties of the TCustomWebAppDataModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 6510)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6505) object.

66.1.3.6.1 TCustomWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6505) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents (see page 6217) object.

See Also

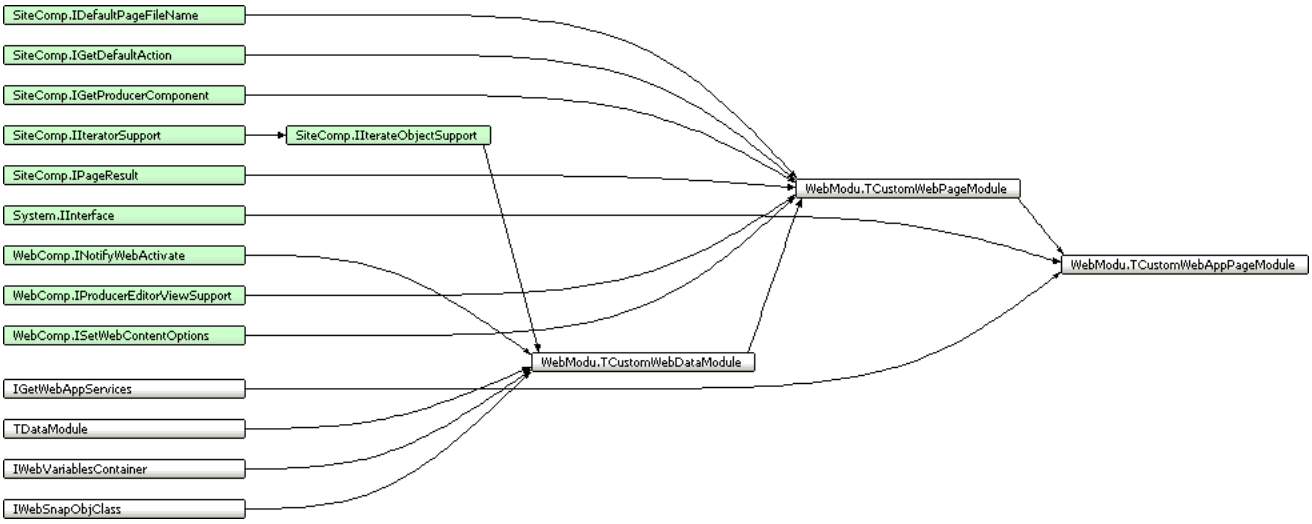
TWebAppComponents (see page 6217)

WebContext (see page 5977)

66.1.4 TCustomWebAppPageModule Class

TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.

Class Hierarchy



Pascal

```
TCustomWebAppPageModule = class(TCustomWebPageModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppPageModule : public TCustomWebPageModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppPageModule as a base class for defining data modules that function as the main Web module in a WebSnap application and that have an associated Web page. As the application module, TCustomWebAppPageModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling. As a Web page module, TCustomWebAppPageModule generates an associated Web page. This Web page is returned by default when incoming request messages do not specify what Web page they want.

See Also

IWebAppServices

TWebAppPageModule (see page 6554)

WebContext

66.1.4.1 TCustomWebAppPageModule Members

The following tables list the members exposed by TCustomWebAppPageModule.

Protected Methods

	Name	Description
V	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
V	GetSiteModuleHelper (see page 6526)	Returns the value of the SiteModuleHelper (see page 6521) property.
V	GetSitePageModuleHelper (see page 6527)	Returns the value of the SitePageModuleHelper (see page 6528) property.
	GetWebAppServices (see page 6514)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
V	Notification (see page 6514)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 6514)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6515)	Sets the value of the AppServices (see page 6515) property.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.
R	SitePageModuleHelper (see page 6528)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 6522)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6529)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6529)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6530)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6530)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6522)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	Create (see page 6512)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6510).
V	CreateNew (see page 6513)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6510) without using an associated form file.
V	Destroy (see page 6513)	Frees an instance of TCustomWebAppPageModule (see page 6510).
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	AppServices (see page 6515)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6510) object.
R	DefaultAction (see page 970)	This is DefaultAction, a member of class IGetDefaultAction.
	DefaultAction (see page 6527)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6528)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

66.1.4.2 TCustomWebAppPageModule.Create Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 6510).

Pascal

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

C++

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

Description

Do not call the TCustomWebAppPageModule (see page 6510) constructor. Instead, use a TWebAppPageModuleFactory

object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6267)

Owner

66.1.4.3 TCustomWebAppPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 6510) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6512) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6267)

Owner

66.1.4.4 TCustomWebAppPageModule.Destroy Destructor

Frees an instance of TCustomWebAppPageModule (see page 6510).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppPageModule();
```

Description

Destroy is the destructor for the object instance. It clears the AppServices (see page 6515) property and calls the inherited destructor. The AppServices (see page 6515) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 6515) may also be owned by a Web page module.

See Also

AppServices (see page 6515)

System::TObject::Free (see page 3382)

66.1.4.5 TCustomWebAppPageModule Methods

The methods of the TCustomWebAppPageModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 6514)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 6514)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 6514)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6515)	Sets the value of the AppServices (see page 6515) property.

66.1.4.5.1 TCustomWebAppPageModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this Web application module. This is the same as the value of the AppServices (see page 6515) property.

See Also

AppServices (see page 6515)

66.1.4.5.2 TCustomWebAppPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppPageModule (see page 6510) uses this method to check whether the component that implements AppServices (see page 6515) is about to be removed, and if so, sets the value of the AppServices (see page 6515) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 6515)

66.1.4.5.3 TCustomWebAppPageModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 3504)

66.1.4.5.4 TCustomWebAppPageModule.SetAppServices Method

Sets the value of the AppServices (see page 6515) property.

Pascal

```
procedure SetAppServices(const Value: IWebAppServices);
```

C++

```
__fastcall SetAppServices(const IWebAppServices Value);
```

Description

SetAppServices is the protected write implemenation of the AppServices (see page 6515) property.

See Also

AppServices (see page 6515)

66.1.4.6 TCustomWebAppPageModule Properties

The properties of the TCustomWebAppPageModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 6515)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6510) object.

66.1.4.6.1 TCustomWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6510) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents (see page 6217) object.

See Also

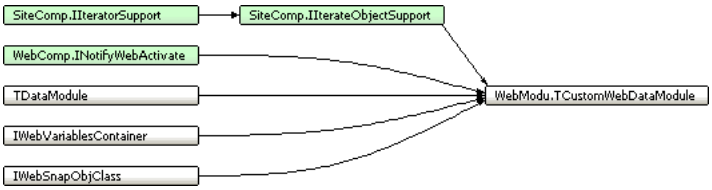
TWebAppComponents (see page 6217)

WebContext (see page 5977)

66.1.5 TCustomWebDataModule Class

TCustomWebDataModule is the base class for all Web modules in a WebSnap application.

Class Hierarchy



Pascal

```
TCustomWebDataModule = class(TDataModule, IWebVariablesContainer, INotifyWebActivate,
IIterateObjectSupport, IWebSnapObjClass);
```

C++

```
class TCustomWebDataModule : public TDataModule, public IWebVariablesContainer, public
INotifyWebActivate, public IIterateObjectSupport, public IWebSnapObjClass;
```

File

WebModu

Description

TCustomWebDataModule is the common base class for Web modules that can be used in a WebSnap application. This class introduces properties for the current HTTP request and response objects, a property to hold the current session object, and events that let you respond when the Web module is activated and deactivated. It also introduces support for interacting with server-side script and managing the WebSnap components that are placed into the Web module.

Do not create instances of TCustomWebDataModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebDataModule descendant to your WebSnap application. Descendant classes include TWebAppDataModule (see page 6550), TWebAppPageModule (see page 6554), TWebPageModule (see page 6569), and TWebDataModule (see page 6562).

See Also

TWebAppDataModule (see page 6550)

TWebAppPageModule (see page 6554)

TWebPageModule (see page 6569)

TWebDataModule (see page 6562)

66.1.5.1 TCustomWebDataModule Members

The following tables list the members exposed by TCustomWebDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Public Events

	Name	Description
	OnActivate (see page 6522)	Occurs when the Web module is activated.
	OnDeactivate (see page 6522)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	Create (see page 6517)	Creates and initializes an instance of TCustomWebDataModule (see page 6516).
	CreateNew (see page 6518)	Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.
	Destroy (see page 6518)	Frees an instance of TCustomWebDataModule (see page 6516).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	Request (see page 6520)	Provides access to the current request object.
	Response (see page 6520)	Provides access to the current response object.
	Session (see page 6521)	Provides access to the current Web session.

66.1.5.2 TCustomWebDataModule.Create Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 6516).

Pascal

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

C++

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

Description

Do not call the TCustomWebDataModule (see page 6516) constructor. Instead, use a Web module factory object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6267)

TWebAppDataModuleFactory (see page 6265)

TWebDataModuleFactory (see page 6269)

TWebPageModuleFactory (see page 6273)

Owner

66.1.5.3 TCustomWebDataModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6517) to create a Web module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6267)

Owner

66.1.5.4 TCustomWebDataModule.Destroy Destructor

Frees an instance of TCustomWebDataModule (see page 6516).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebDataModule();
```

Description

Typically, WebSnap applications do not explicitly free Web module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext

System::TObject::Free (see page 3382)

66.1.5.5 TCustomWebDataModule Methods

The methods of the TCustomWebDataModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.

66.1.5.5.1 TCustomWebDataModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 6521) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 6521) property.

See Also

SiteModuleHelper (see page 6521)

66.1.5.5.2 TCustomWebDataModule.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

66.1.5.5.3 TCustomWebDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebDataModule (

see page 6516) overrides this method to pass the notification on to the object that is the value of the SiteModuleHelper (see page 6521) property.

See Also

SiteModuleHelper (see page 6521)

66.1.5.6 TCustomWebDataModule Properties

The properties of the TCustomWebDataModule class are listed here.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Public Properties

	Name	Description
	Request (see page 6520)	Provides access to the current request object.
	Response (see page 6520)	Provides access to the current response object.
	Session (see page 6521)	Provides access to the current Web session.

66.1.5.6.1 TCustomWebDataModule.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3727) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6520)

66.1.5.6.2 TCustomWebDataModule.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3727) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6520) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6520)

66.1.5.6.3 TCustomWebDataModule.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read (see page 3727) Session to obtain information from or add information to the current Web session. All requests are assigned a Web session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a timeout period elapses).

See Also

TSessionsService (see page 6722)

66.1.5.6.4 TCustomWebDataModule.SiteModuleHelper Property

References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Pascal

```
property SiteModuleHelper: TSiteModuleHelper;
```

C++

```
__property TSiteModuleHelper SiteModuleHelper;
```

Description

Applications can't use the protected SiteModuleHelper property. TCustomWebDataModule (see page 6516) uses this property internally to delegate tasks involved in managing the WebSnap components it contains and to generate the OnActivate (see page 6522) and OnDeactivate (see page 6522) events when the Web module is activated and deactivated.

See Also

OnActivate (see page 6522)

OnDeactivate (see page 6522)

66.1.5.7 TCustomWebDataModule Events

The events of the TCustomWebDataModule class are listed here.

Public Events

	Name	Description
	OnActivate (see page 6522)	Occurs when the Web module is activated.
	OnDeactivate (see page 6522)	Occurs when the Web module is deactivated.

66.1.5.7.1 TCustomWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write (see page 3789) an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6522)

CreateMode

66.1.5.7.2 TCustomWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write (see page 3789) an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

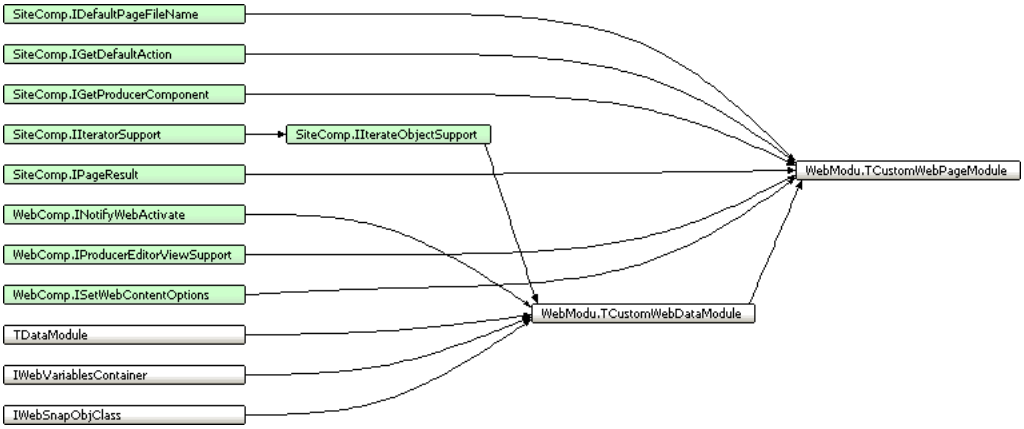
See Also

OnActivate (see page 6522)

66.1.6 TCustomWebPageModule Class

TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.

Class Hierarchy



Pascal

```
TCustomWebPageModule = class(TCustomWebDataModule, IDefaultPageFileName, IPageResult,
ISetWebContentOptions, IGetDefaultAction, IGetProducerComponent, IProducerEditorViewSupport);
```

C++

```
class TCustomWebPageModule : public TCustomWebDataModule, public IDefaultPageFileName, public
IPageResult, public ISetWebContentOptions, public IGetDefaultAction, public
IGetProducerComponent, public IProducerEditorViewSupport;
```

File

WebModu

Description

TCustomWebPageModule is the common base class for Web modules that generate a Web page for a WebSnap application. This class introduces two new properties:

PageProducer (see page 6528), which is a page producer component that generates the Web page associated with this module.

DefaultAction (see page 6527), which is an adapter action associated with the module's Web page. Typically, the action sets up parameters that appear in HTTP requests for the Web page.

TCustomWebPageModule also introduces events that occur when the WebSnap application calls on the Web page module to generate its page or when it generates a redirect message to redirect a request to the associated page.

Do not create instances of TCustomWebPageModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebPageModule descendant to your WebSnap application. Descendant classes include TWebAppPageModule (see page 6554) and TWebPageModule (see page 6569).

See Also

TWebPageModule (see page 6569)

TWebAppPageModule (see page 6554)

66.1.6.1 TCustomWebPageModule Members

The following tables list the members exposed by TCustomWebPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6526)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSitePageModuleHelper (see page 6527)	Returns the value of the SitePageModuleHelper (see page 6528) property.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6527)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.
	SitePageModuleHelper (see page 6528)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 6522)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6529)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6529)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6530)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6530)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6522)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	Create (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522).
	CreateNew (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522) without using an associated form file.
	Destroy (see page 6526)	Frees an instance of TCustomWebPageModule (see page 6522).
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	DefaultAction (see page 6527)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6528)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

66.1.6.2 TCustomWebPageModule.Create Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 6522).

Pascal

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

C++

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

Description

Do not call the TCustomWebPageModule (see page 6522) constructor. Instead, use a Web page module factory object to create Web page module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6267)

TWebPageModuleFactory (see page 6273)

Owner

66.1.6.3 TCustomWebPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 6522) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6525) to create a Web page module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6267)

Owner

66.1.6.4 TCustomWebPageModule.Destroy Destructor

Frees an instance of TCustomWebPageModule (see page 6522).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebPageModule();
```

Description

Typically, WebSnap applications do not explicitly free Web page module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext

System::TObject::Free (see page 3382)

66.1.6.5 TCustomWebPageModule Methods

The methods of the TCustomWebPageModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6526)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSitePageModuleHelper (see page 6527)	Returns the value of the SitePageModuleHelper (see page 6528) property.
	Notification (see page 6527)	Responds to notifications that components are about to be inserted or removed.

66.1.6.5.1 TCustomWebPageModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 6521) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; override;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 6521) property. TCustomWebPageModule (see page 6522) overrides the inherited method to return the value of the SitePageModuleHelper (see page 6528) property.

See Also

SiteModuleHelper (see page 6521)

SitePageModuleHelper (see page 6528)

66.1.6.5.2 TCustomWebPageModule.GetSitePageModuleHelper Method

Returns the value of the SitePageModuleHelper (see page 6528) property.

Pascal

```
function GetSitePageModuleHelper: TSitePageModuleHelper; virtual;
```

C++

```
virtual __fastcall TSitePageModuleHelper GetSitePageModuleHelper();
```

Description

GetSitePageModuleHelper is the protected read implementation of the SitePageModuleHelper (see page 6528) property.

See Also

SitePageModuleHelper (see page 6528)

66.1.6.5.3 TCustomWebPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebPageModule (see page 6522) overrides this method to update the PageProducer (see page 6528) and DefaultAction (see page 6527) properties as appropriate.

See Also

PageProducer (see page 6528)

DefaultAction (see page 6527)

66.1.6.6 TCustomWebPageModule Properties

The properties of the TCustomWebPageModule class are listed here.

Protected Properties

	Name	Description
	SitePageModuleHelper (see page 6528)	References the helper object that assists in assembling the module's Web page.

Public Properties

	Name	Description
	DefaultAction (see page 6527)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6528)	Indicates the interface of the page producer that generates the content of the Web page.

66.1.6.6.1 TCustomWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6528)

66.1.6.6.2 TCustomWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1076)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

66.1.6.6.3 TCustomWebPageModule.SitePageModuleHelper Property

References the helper object that assists in assembling the module's Web page.

Pascal

```
property SitePageModuleHelper: TSitePageModuleHelper;
```

C++

```
__property TSitePageModuleHelper SitePageModuleHelper;
```

Description

Applications can't use the protected SitePageModuleHelper property. TCustomWebPageModule (see page 6522) uses this property internally to delegate tasks involved in assembling the content of its Web page and in interacting with editor views at design time.

66.1.6.7 TCustomWebPageModule Events

The events of the TCustomWebPageModule class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6529)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6529)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6530)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6530)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

66.1.6.7.1 TCustomWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write (see page 3789) an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6530)

PageProducer (see page 6528)

Response (see page 6520)

66.1.6.7.2 TCustomWebPageModule.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write (see page 3789) an OnAfterRedirectToPage event handler to take any final action before returns a response message that redirects the user to this Web page. For example, the OnAfterRedirectToPage event handler can make adjustments to the headers of the response message.

This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost) to this page.

See Also

OnBeforeRedirectToPage (see page 6530)

OnAfterDispatchPage (see page 6529)

RedirectOptions (see page 5464)

Response (see page 6520)

66.1.6.7.3 TCustomWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write (see page 3789) an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request (see page 6520) and Response (see page 6520) properties.

See Also

OnAfterDispatchPage (see page 6529)

Request (see page 6520)

Response (see page 6520)

PageProducer (see page 6528)

66.1.6.7.4 TCustomWebPageModule.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write (see page 3789) an OnBeforeRedirectToPage event handler to perform any tasks necessary for redirecting a request to the associated Web page. For example, the OnBeforeRedirectToPage event handler can add parameters to the Params parameter so that they appear in the request that results from the redirection. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the Web page module assumes it has handled the redirection and performs no further processing. Otherwise, after the event handler exits, the Web page module sets up the response message to encode the redirection parameters and calls SendRedirect.

See Also

OnAfterRedirectToPage (see page 6529)

OnBeforeDispatchPage (see page 6530)

SendRedirect

Request (see page 6520)

Response (see page 6520)

66.1.7 TSiteModuleHelper Class

TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSiteModuleHelper = class(TObject);
```

C++

```
class TSiteModuleHelper : public TObject;
```

File

WebModu

Description

Web modules create and use an instance of TSiteModuleHelper to keep track of WebSnap objects contained in the module and to generate OnActivate (see page 6539) and OnDeactivate (see page 6539) events when the Web module is activated and deactivated.

You should not need to work directly with TSiteModuleHelper instances. The Web module creates and uses this object.

See Also

TCustomWebDataModule (see page 6516)

TCustomWebDataModule (see page 6516)

TCustomWebDataModule (see page 6516)

66.1.7.1 TSiteModuleHelper Members

The following tables list the members exposed by TSiteModuleHelper.

Protected Methods

	Name	Description
	EndIterator (see page 6534)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6534)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6535)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6535)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6535)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 6535)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6535)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 6536)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6536)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6536)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 6537)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 6537)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6539)	This is OnActivate, a member of class TSiteModuleHelper.
	OnDeactivate (see page 6539)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6533)	Creates and initializes an instance of TSiteModuleHelper (see page 6531).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6533)	Frees an instance of TSiteModuleHelper (see page 6531).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.

	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6536)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Request (see page 6537)	Provides access to the current request object.
R	Response (see page 6538)	Provides access to the current response object.
R	Session (see page 6538)	Provides access to the current Web session.
R	WebModuleContext (see page 6538)	Keeps track of name/value pairs that store information for the Web module.

66.1.7.2 TSiteModuleHelper.Create Constructor

Creates and initializes an instance of TSiteModuleHelper (see page 6531).

Pascal

```
constructor Create(AModule: TComponent); virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper(TComponent * AModule);
```

Description

Web modules call the TSiteModuleHelper (see page 6531) constructor from their own constructors.

AModule is the Web module that is using the TSiteModuleHelper (see page 6531) object to keep track of its components and activation events. The TSiteModuleHelper (see page 6531) object becomes the value of the module's SiteModuleHelper property.

See Also

SiteModuleHelper (see page 6521)

66.1.7.3 TSiteModuleHelper.Destroy Destructor

Frees an instance of TSiteModuleHelper (see page 6531).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSiteModuleHelper();
```

Description

Do not free TSiteModuleHelper (see page 6531) instances. This object is freed by the Web module that uses it.

See Also

System::TObject::Free (see page 3382)

66.1.7.4 TSiteModuleHelper Methods

The methods of the TSiteModuleHelper class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 6534)	This is EndIterator, a member of class TSiteModuleHelper.
	FindNamedVariable (see page 6534)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6535)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6535)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6535)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 6535)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6535)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 6536)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6536)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6536)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 6537)	This is StartIterator, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
	Notification (see page 6536)	Responds to notifications that components are about to be inserted or removed.

66.1.7.4.1 TSiteModuleHelper.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TSiteModuleHelper.

66.1.7.4.2 TSiteModuleHelper.FindNamedVariable Method

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

This is FindNamedVariable, a member of class TSiteModuleHelper.

66.1.7.4.3 TSiteModuleHelper.FindVariable Method

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

This is FindVariable, a member of class TSiteModuleHelper.

66.1.7.4.4 TSiteModuleHelper.FindVariableContainer Method

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

This is FindVariableContainer, a member of class TSiteModuleHelper.

66.1.7.4.5 TSiteModuleHelper.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

This is GetCurrentObject, a member of class TSiteModuleHelper.

66.1.7.4.6 TSiteModuleHelper.GetVariable Method

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

This is GetVariable, a member of class TSiteModuleHelper.

66.1.7.4.7 TSiteModuleHelper.GetVariableCount Method

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

This is GetVariableCount, a member of class TSiteModuleHelper.

66.1.7.4.8 TSiteModuleHelper.NextIteration Method

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

This is NextIteration, a member of class TSiteModuleHelper.

66.1.7.4.9 TSiteModuleHelper.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation);
```

C++

```
__fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TSiteModuleHelper (see page 6531) overrides this method to ensure that any WebSnap objects added to the associated module receive notifications when the module is activated or deactivated.

66.1.7.4.10 TSiteModuleHelper.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class TSiteModuleHelper.

66.1.7.4.11 TSiteModuleHelper.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class TSiteModuleHelper.

66.1.7.4.12 TSiteModuleHelper.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TSiteModuleHelper.

66.1.7.5 TSiteModuleHelper Properties

The properties of the TSiteModuleHelper class are listed here.

Protected Properties

	Name	Description
	Module (see page 6537)	This is Module, a member of class TSiteModuleHelper.

Public Properties

	Name	Description
	Request (see page 6537)	Provides access to the current request object.
	Response (see page 6538)	Provides access to the current response object.
	Session (see page 6538)	Provides access to the current Web session.
	WebModuleContext (see page 6538)	Keeps track of name/value pairs that store information for the Web module.

66.1.7.5.1 TSiteModuleHelper.Module Property

Pascal

```
property Module: TComponent;
```

C++

```
__property TComponent * Module;
```

Description

This is Module, a member of class TSiteModuleHelper.

66.1.7.5.2 TSiteModuleHelper.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3727) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6538)

66.1.7.5.3 TSiteModuleHelper.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3727) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6537) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6537)

66.1.7.5.4 TSiteModuleHelper.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read (see page 3727) Session to obtain information from or add information to the current Web session. All requests are assigned a Web session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a time-out period elapses).

See Also

TSessionsService (see page 6722)

66.1.7.5.5 TSiteModuleHelper.WebModuleContext Property

Keeps track of name/value pairs that store information for the Web module.

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

Use WebModuleContext to access the TWebModuleContext object. The TWebModuleContext object maintains a list of named variables that the components in the Web module can use.

See Also

WebModuleContext

66.1.7.6 TSiteModuleHelper Events

The events of the TSiteModuleHelper class are listed here.

Public Events

	Name	Description
	OnActivate (see page 6539)	This is OnActivate, a member of class TSiteModuleHelper.
	OnDeactivate (see page 6539)	This is OnDeactivate, a member of class TSiteModuleHelper.

66.1.7.6.1 TSiteModuleHelper.OnActivate Event

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

This is OnActivate, a member of class TSiteModuleHelper.

66.1.7.6.2 TSiteModuleHelper.OnDeactivate Event

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

This is OnDeactivate, a member of class TSiteModuleHelper.

66.1.8 TSitePageModuleHelper Class

TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSitePageModuleHelper = class(TSiteModuleHelper);
```

C++

```
class TSitePageModuleHelper : public TSiteModuleHelper;
```

File

WebModu

Description

Web page modules create and use an instance of TSitePageModuleHelper to assist in the following tasks:

Assembling the content of the associated Web page.

Interacting with editor views at design time.

Keeping track of WebSnap objects contained in the module.

Generating the events that occur when the Web module is activated or deactivated, and when it dispatches or redirects a request for a Web page.

You should not need to work directly with TSitePageModuleHelper instances. The Web page module creates and uses this object.

See Also

TCustomWebPageModule (see page 6522)

TCustomWebDataModule (see page 6516)

TCustomWebDataModule (see page 6516)

TCustomWebPageModule (see page 6522)

TCustomWebPageModule (see page 6522)

TCustomWebPageModule (see page 6522)

TCustomWebPageModule (see page 6522)

66.1.8.1 TSitePageModuleHelper Members

The following tables list the members exposed by TSitePageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6543)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6534)	This is EndIterator, a member of class TSiteModuleHelper.
	FindNamedVariable (see page 6534)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6535)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6535)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6535)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6543)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
	GetPageProducer (see page 6544)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 6544)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6544)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6544)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6545)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6535)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6535)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6545)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6545)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6545)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6545)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6546)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6546)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6546)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.

	ImplDispatchPage (see page 6546)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
	ImplIncludePage (see page 6547)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
	ImplRedirectToPage (see page 6547)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
	ImplSetWebContentOptions (see page 6547)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6547)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6536)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6536)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6536)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6547)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6548)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6548)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6537)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
	Module (see page 6537)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6539)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 6549)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6549)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6550)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6550)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6539)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6542)	Creates and initializes an instance of TSitePageModuleHelper (see page 6539).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 6542)	Frees an instance of TSitePageModuleHelper (see page 6539).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.

	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6536)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6548)	Represents the adapter action associated with the Web page module.
R	Request (see page 6537)	Provides access to the current request object.
R	Response (see page 6538)	Provides access to the current response object.
R	Session (see page 6538)	Provides access to the current Web session.
R	WebModuleContext (see page 6538)	Keeps track of name/value pairs that store information for the Web module.

66.1.8.2 TSitePageModuleHelper.Create Constructor

Creates and initializes an instance of TSitePageModuleHelper (see page 6539).

Pascal

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

C++

```
virtual __fastcall TSitePageModuleHelper(TComponent * AModule);
```

Description

Web page modules call the TSitePageModuleHelper (see page 6539) constructor from their own constructors.

AModule is the Web page module that is using the TSitePageModuleHelper (see page 6539) object to generate events and help assemble its Web content. The TSitePageModuleHelper (see page 6539) object becomes the value of the module's SitePageModuleHelper property.

See Also

SitePageModuleHelper (see page 6528)

66.1.8.3 TSitePageModuleHelper.Destroy Destructor

Frees an instance of TSitePageModuleHelper (see page 6539).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSitePageModuleHelper();
```

Description

Do not free TSitePageModuleHelper (see page 6539) instances. This object is freed by the Web page module that uses it.

See Also

System::TObject::Free (see page 3382)

66.1.8.4 TSitePageModuleHelper Methods

The methods of the TSitePageModuleHelper class are listed here.

Protected Methods

	Name	Description
	DispatchPage (see page 6543)	This is DispatchPage, a member of class TSitePageModuleHelper.
	GetDefaultAction (see page 6543)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6544)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 6544)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6544)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6544)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6545)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetXMLData (see page 6545)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6545)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6545)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6545)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6546)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6546)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6546)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6546)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6547)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6547)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6547)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6547)	This is IncludePage, a member of class TSitePageModuleHelper.
	RedirectToPage (see page 6547)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6548)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6548)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.

66.1.8.4.1 TSitePageModuleHelper.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const UnicodeString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class TSitePageModuleHelper.

66.1.8.4.2 TSitePageModuleHelper.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class TSitePageModuleHelper.

66.1.8.4.3 TSitePageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; virtual;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TSitePageModuleHelper.

66.1.8.4.4 TSitePageModuleHelper.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class TSitePageModuleHelper.

66.1.8.4.5 TSitePageModuleHelper.GetRedirectParams Method

Pascal

```
procedure GetRedirectParams(const APageName: string; AParams: TStrings);
```

C++

```
__fastcall GetRedirectParams(const UnicodeString APageName, TStrings AParams);
```

Description

This is GetRedirectParams, a member of class TSitePageModuleHelper.

66.1.8.4.6 TSitePageModuleHelper.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TSitePageModuleHelper.

66.1.8.4.7 TSitePageModuleHelper.GetTemplateStream Method

Pascal

```
function GetTemplateStream(const APage: string; out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(const UnicodeString APage, Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TSitePageModuleHelper.

66.1.8.4.8 TSitePageModuleHelper.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TSitePageModuleHelper.

66.1.8.4.9 TSitePageModuleHelper.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TSitePageModuleHelper.

66.1.8.4.10 TSitePageModuleHelper.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.

66.1.8.4.11 TSitePageModuleHelper.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TSitePageModuleHelper.

66.1.8.4.12 TSitePageModuleHelper.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TSitePageModuleHelper.

66.1.8.4.13 TSitePageModuleHelper.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TSitePageModuleHelper.

66.1.8.4.14 TSitePageModuleHelper.HTMLFromTemplate Method

Pascal

```
function HTMLFromTemplate(AStream: TStream): string;
```

C++

```
__fastcall UnicodeString HTMLFromTemplate(TStream AStream);
```

Description

This is HTMLFromTemplate, a member of class TSitePageModuleHelper.

66.1.8.4.15 TSitePageModuleHelper.ImplDispatchPage Method

Pascal

```
function ImplDispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplDispatchPage(const UnicodeString APageName, TWebResponse
AWebResponse);
```

Description

This is ImplDispatchPage, a member of class TSitePageModuleHelper.

66.1.8.4.16 TSitePageModuleHelper.ImplIncludePage Method

Pascal

```
function ImplIncludePage(const APageName: string; var AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplIncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is ImplIncludePage, a member of class TSitePageModuleHelper.

66.1.8.4.17 TSitePageModuleHelper.ImplRedirectToPage Method

Pascal

```
function ImplRedirectToPage(const APageName: string; AParams: TStrings; AResponse:
TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplRedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse);
```

Description

This is ImplRedirectToPage, a member of class TSitePageModuleHelper.

66.1.8.4.18 TSitePageModuleHelper.ImplSetWebContentOptions Method

Pascal

```
procedure ImplSetWebContentOptions(AOptions: TWebContentOptions); virtual;
```

C++

```
virtual __fastcall ImplSetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.

66.1.8.4.19 TSitePageModuleHelper.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class TSitePageModuleHelper.

66.1.8.4.20 TSitePageModuleHelper.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AResponse: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse);
```

Description

This is RedirectToPage, a member of class TSitePageModuleHelper.

66.1.8.4.21 TSitePageModuleHelper.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const AValue: string);
```

C++

```
__fastcall SetDefaultPageFileName(const UnicodeString AValue);
```

Description

This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.

66.1.8.4.22 TSitePageModuleHelper.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class TSitePageModuleHelper.

66.1.8.5 TSitePageModuleHelper Properties

The properties of the TSitePageModuleHelper class are listed here.

Public Properties

	Name	Description
	DefaultAction (see page 6548)	Represents the adapter action associated with the Web page module.

66.1.8.5.1 TSitePageModuleHelper.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the

parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6528)

66.1.8.6 TSitePageModuleHelper Events

The events of the TSitePageModuleHelper class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6549)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6549)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6550)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6550)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

66.1.8.6.1 TSitePageModuleHelper.OnAfterDispatchPage Event

Occurs after the Web page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

OnAfterDispatchPage uses the same event handler as that of the Web page module's OnAfterDispatchPage event. The TSitePageModuleHelper (see page 6539) object generates this event after the Web page module writes the content for its Web page to the response object.

See Also

ondeactivate (see page 6539)

OnAfterDispatchPage (see page 6529)

66.1.8.6.2 TSitePageModuleHelper.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

OnAfterRedirectToPage uses the same event handler as that of the Web page module's OnAfterRedirectToPage event. The TSitePageModuleHelper (see page 6539) object generates this event after the Web page module sends a response redirecting the caller to its Web page.

See Also

OnBeforeRedirectToPage (see page 6550)

OnAfterRedirectToPage (see page 6529)

66.1.8.6.3 TSitePageModuleHelper.OnBeforeDispatchPage Event

Occurs before the Web module generates the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

OnBeforeDispatchPage uses the same event handler as that of the Web page module's OnBeforeDispatchPage event. The TSitePageModuleHelper (see page 6539) object generates this event before the Web page module generates the content for its Web page.

See Also

OnAfterDispatchPage (see page 6549)

OnBeforeDispatchPage (see page 6530)

66.1.8.6.4 TSitePageModuleHelper.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

OnBeforeRedirectToPage uses the same event handler as that of the Web page module's OnBeforeRedirectToPage event. The TSitePageModuleHelper (see page 6539) object generates this event before the Web page module assembles and sends a response redirecting the caller to its Web page.

See Also

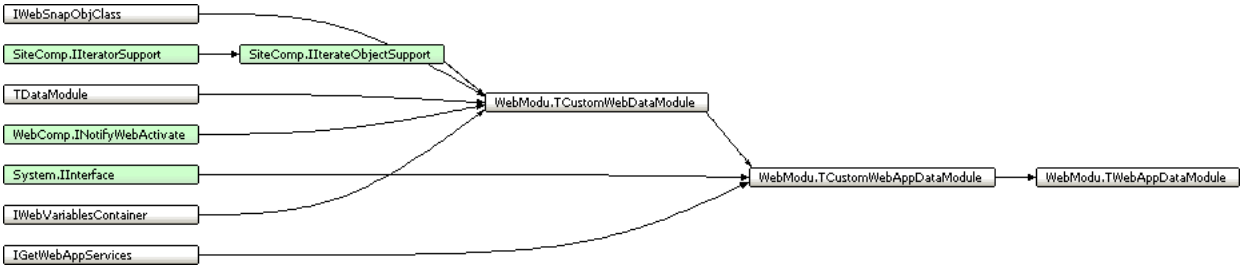
OnAfterRedirectToPage (see page 6549)

OnBeforeRedirectToPage (see page 6530)

66.1.9 TWebAppDataModule Class

TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.

Class Hierarchy



Pascal

```
TWebAppDataModule = class(TCustomWebAppDataModule);
```

C++

```
class TWebAppDataModule : public TCustomWebAppDataModule;
```

File

WebModu

Description

TWebAppDataModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppDataModule when it creates a TWebAppDataModule. System (see page 3165) components may be added or removed later. TWebAppComponents (see page 6217) is the component in a TWebAppDataModule that references all of the system components in use by the WebSnap application.

See Also

Web Modules

66.1.9.1 TWebAppDataModule Members

The following tables list the members exposed by TWebAppDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetWebAppServices (see page 6508)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6509)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6509)	Returns a reference to a specified interface if the module supports that interface.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.

	_Release (see page 3504)	Decrements the reference count for this interface.
V	Create (see page 6507)	Creates a new TCustomWebAppDataModule (see page 6505) instance.
V	Create (see page 6517)	Creates and initializes an instance of TCustomWebDataModule (see page 6516).
V	CreateNew (see page 6507)	Creates a data module without using the associated form file to initialize it.
V	CreateNew (see page 6518)	Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.
V	Destroy (see page 6508)	Frees a Web application data module instance.
V	Destroy (see page 6518)	Frees an instance of TCustomWebDataModule (see page 6516).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6553)	Occurs when the Web module is activated.
	OnDeactivate (see page 6553)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	AppServices (see page 6552)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

66.1.9.2 TWebAppDataModule Properties

The properties of the TWebAppDataModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 6552)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

66.1.9.2.1 TWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects

on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 6217)

WebContext (see page 5977)

66.1.9.3 TWebAppDataModule Events

The events of the TWebAppDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6553)	Occurs when the Web module is activated.
	OnDeactivate (see page 6553)	Occurs when the Web module is deactivated.

66.1.9.3.1 TWebAppDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6522)

CreateMode

66.1.9.3.2 TWebAppDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

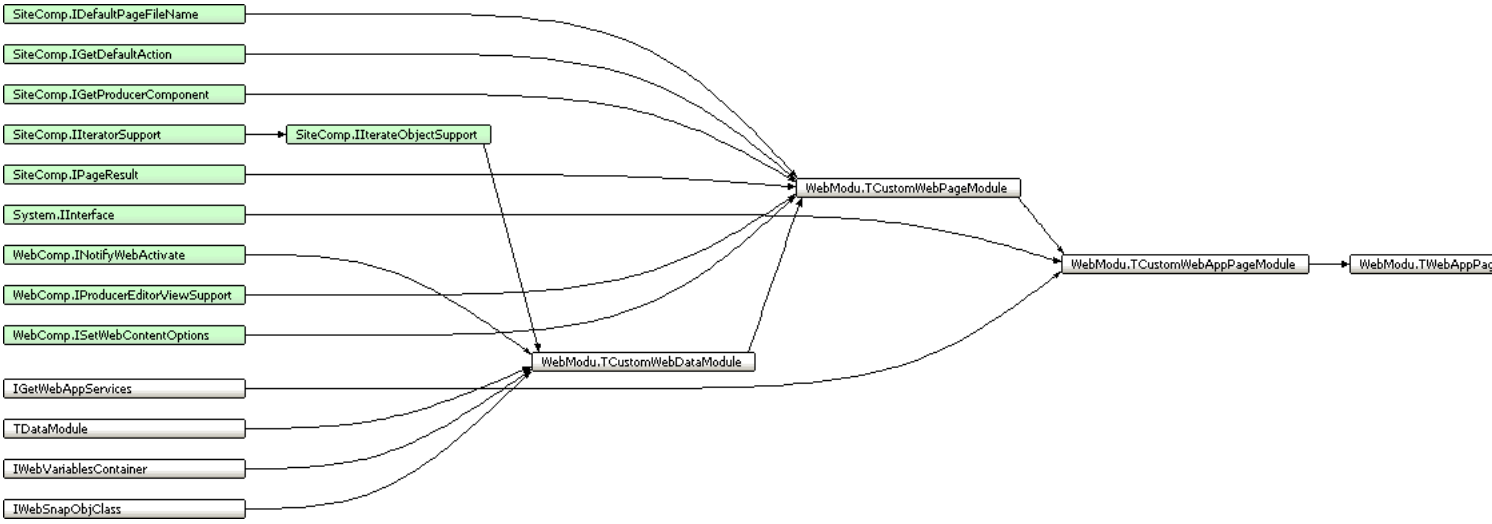
See Also

OnActivate (see page 6522)

66.1.10 TWebAppPageModule Class

TWebAppPageModule is used as the WebSnap application module when the application module generates a page.

Class Hierarchy



Pascal

```
TWebAppPageModule = class(TCustomWebAppPageModule);
```

C++

```
class TWebAppPageModule : public TCustomWebAppPageModule;
```

File

WebModu

Description

Like TWebAppDataModule (see page 6550), TWebAppPageModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppPageModule when it creates a TWebAppPageModule. System (see page 3165) components may be added or removed later. TWebAppComponents (see page 6217) is the component in a TWebAppPageModule that references all of the system components in use by the WebSnap application.

The WebAppPageModule is different from TWebAppDataModule (see page 6550) because it generates a page. The PageProducer (see page 6557) property references the producer component responsible for generating content. This property must not be nil (Delphi) or NULL (C++) or no content will be generated. The WebSnap Application Wizard automatically adds a page producer component when it creates a new TWebAppPageModule.

See Also

Web Modules

66.1.10.1 TWebAppPageModule Members

The following tables list the members exposed by TWebAppPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSiteModuleHelper (see page 6526)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSitePageModuleHelper (see page 6527)	Returns the value of the SitePageModuleHelper (see page 6528) property.
	GetWebAppServices (see page 6514)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6514)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6527)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6514)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6515)	Sets the value of the AppServices (see page 6515) property.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.
	SitePageModuleHelper (see page 6528)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6529)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeRedirectToPage (see page 6530)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	Create (see page 6512)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6510).
	Create (see page 6517)	Creates and initializes an instance of TCustomWebDataModule (see page 6516).
	Create (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522).
	CreateNew (see page 6513)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6510) without using an associated form file.
	CreateNew (see page 6518)	Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.
	CreateNew (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522) without using an associated form file.
	Destroy (see page 6513)	Frees an instance of TCustomWebAppPageModule (see page 6510).
	Destroy (see page 6518)	Frees an instance of TCustomWebDataModule (see page 6516).
	Destroy (see page 6526)	Frees an instance of TCustomWebPageModule (see page 6522).
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.

	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6558)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6558)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6559)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6559)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	AppServices (see page 6556)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 6557)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6557)	Indicates the interface of the page producer that generates the content of the Web page.

66.1.10.2 TWebAppPageModule Properties

The properties of the TWebAppPageModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 6556)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 6557)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6557)	Indicates the interface of the page producer that generates the content of the Web page.

66.1.10.2.1 TWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 6217)

WebContext (see page 5977)

66.1.10.2.2 TWebAppPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6528)

66.1.10.2.3 TWebAppPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1076)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

66.1.10.3 TWebAppPageModule Events

The events of the TWebAppPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6558)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6558)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6559)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6559)	Occurs when the Web module is deactivated.

66.1.10.3.1 TWebAppPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6522)

CreateMode

66.1.10.3.2 TWebAppPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6530)

PageProducer (see page 6528)

Response (see page 6520)

66.1.10.3.3 TWebAppPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 6529)

Request (see page 6520)

Response (see page 6520)

PageProducer (see page 6528)

66.1.10.3.4 TWebAppPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6522)

66.1.11 TWebAppPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebAppPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebAppPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebAppPageModuleHelper.

66.1.11.1 TWebAppPageModuleHelper Members

The following tables list the members exposed by TWebAppPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6543)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6534)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6534)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6535)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6535)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6535)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6543)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6562)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.
	GetProducerComponent (see page 6544)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6544)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6544)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6545)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6535)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6535)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6545)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6545)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6545)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6545)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6546)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.

	HasXSLBrowserView (see page 6546)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6546)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6546)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6547)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6547)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6547)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6547)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6536)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6536)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6536)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6547)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6548)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6548)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6537)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 6537)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6539)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 6549)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6549)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6550)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6550)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6539)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 6533)	Creates and initializes an instance of TSiteModuleHelper (see page 6531).
V	Create (see page 6542)	Creates and initializes an instance of TSitePageModuleHelper (see page 6539).
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 6533)	Frees an instance of TSiteModuleHelper (see page 6531).
V	Destroy (see page 6542)	Frees an instance of TSitePageModuleHelper (see page 6539).
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.

	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6536)	Responds to notifications that components are about to be inserted or removed.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6548)	Represents the adapter action associated with the Web page module.
	Request (see page 6537)	Provides access to the current request object.
	Response (see page 6538)	Provides access to the current response object.
	Session (see page 6538)	Provides access to the current Web session.
	WebModuleContext (see page 6538)	Keeps track of name/value pairs that store information for the Web module.

66.1.11.2 TWebAppPageModuleHelper Methods

The methods of the TWebAppPageModuleHelper class are listed here.

Protected Methods

	Name	Description
	GetPageProducer (see page 6562)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.

66.1.11.2.1 TWebAppPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

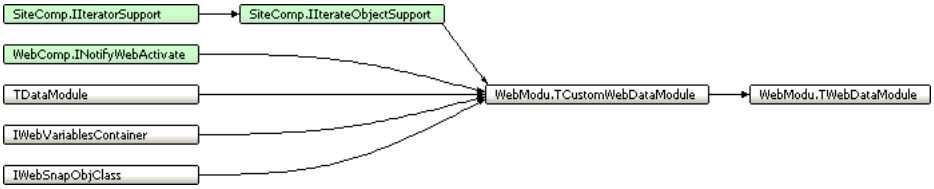
Description

This is GetPageProducer, a member of class TWebAppPageModuleHelper.

66.1.12 TWebDataModule Class

TWebDataModule is a container for components in a WebSnap application.

Class Hierarchy



Pascal

```
TWebDataModule = class(TCustomWebDataModule);
```

C++

```
class TWebDataModule : public TCustomWebDataModule;
```

File

WebModu

Description

Use TWebDataModule as a container for components that may be used by other modules in your WebSnap application. TWebDataModule is compatible with WebSnap applications. The initialization section of TWebDataModule registers a factory that is used by the WebSnap application to create the data module.

See Also

Web Modules

66.1.12.1 TWebDataModule Members

The following tables list the members exposed by TWebDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.

Public Methods

	Name	Description
	Create (see page 6517)	Creates and initializes an instance of TCustomWebDataModule (see page 6516).
	CreateNew (see page 6518)	Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.
	Destroy (see page 6518)	Frees an instance of TCustomWebDataModule (see page 6516).
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.

	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6564)	Occurs when the Web module is activated.
	OnDeactivate (see page 6564)	Occurs when the Web module is deactivated.

66.1.12.2 TWebDataModule Events

The events of the TWebDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6564)	Occurs when the Web module is activated.
	OnDeactivate (see page 6564)	Occurs when the Web module is deactivated.

66.1.12.2.1 TWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6522)

CreateMode

66.1.12.2.2 TWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are

deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6522)

66.1.13 TWebNotifyContainer Class

Class Hierarchy

Pascal

```
TWebNotifyContainer = class(TObject);
```

C++

```
class TWebNotifyContainer : public TObject;
```

File

WebModu

Description

This is class WebModu.TWebNotifyContainer.

66.1.13.1 TWebNotifyContainer Members

The following tables list the members exposed by TWebNotifyContainer.

Protected Fields

	Name	Description
	FList (see page 6567)	This is FList, a member of class TWebNotifyContainer.

Protected Methods

	Name	Description
	GetCount (see page 6568)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 6568)	This is Add, a member of class TWebNotifyContainer.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6566)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 6566)	Disposes of an object instance.

	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyWebActivate (see page 6568)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 6568)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

66.1.13.2 TWebNotifyContainer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebNotifyContainer();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

66.1.13.3 TWebNotifyContainer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebNotifyContainer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

66.1.13.4 TWebNotifyContainer Fields

The fields of the TWebNotifyContainer class are listed here.

Protected Fields

	Name	Description
	FList (see page 6567)	This is FList, a member of class TWebNotifyContainer.

66.1.13.4.1 TWebNotifyContainer.FList Field

Pascal

```
FList: TComponentList;
```

C++

```
TComponentList FList;
```

Description

This is FList, a member of class TWebNotifyContainer.

66.1.13.5 TWebNotifyContainer Methods

The methods of the TWebNotifyContainer class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6568)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 6568)	This is Add, a member of class TWebNotifyContainer.
	NotifyWebActivate (see page 6568)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 6568)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

66.1.13.5.1 TWebNotifyContainer.Add Method

Pascal

```
procedure Add(AComponent: TComponent);
```

C++

```
__fastcall Add(TComponent * AComponent);
```

Description

This is Add, a member of class TWebNotifyContainer.

66.1.13.5.2 TWebNotifyContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebNotifyContainer.

66.1.13.5.3 TWebNotifyContainer.NotifyWebActivate Method

Pascal

```
procedure NotifyWebActivate;
```

C++

```
__fastcall NotifyWebActivate();
```

Description

This is NotifyWebActivate, a member of class TWebNotifyContainer.

66.1.13.5.4 TWebNotifyContainer.NotifyWebDeactivate Method

Pascal

```
procedure NotifyWebDeactivate;
```

C++

```
__fastcall NotifyWebDeactivate();
```

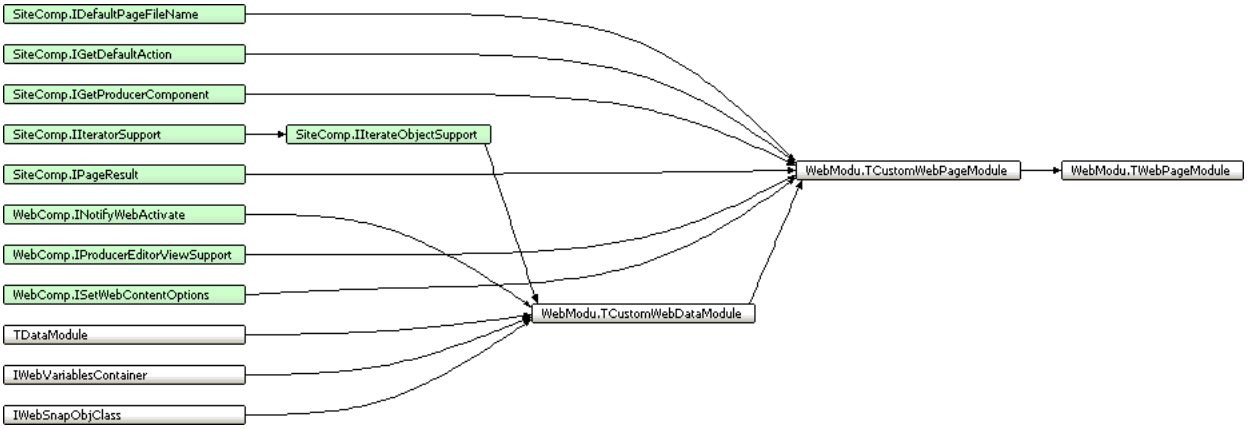
Description

This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

66.1.14 TWebPageModule Class

TWebPageModule is a WebSnap module that generates a page.

Class Hierarchy



Pascal

```
TWebPageModule = class(TCustomWebPageModule);
```

C++

```
class TWebPageModule : public TCustomWebPageModule;
```

File

WebModu

Description

Use TWebPageModule in a WebSnap application to hold all the components needed for generating a specific Web page. When you use the WebSnap page module wizard to create a TWebPageModule descendant, the wizard adds a page producer to the module. The page producer generates the content of the Web page and adds code to the unit it creates (in Delphi, the code is added to the initialization section; in C++, it is added to a #pragma startup). The added code registers the TWebPageModule descendant with a factory object that the WebSnap application can use to create the Web page module.

The page producer that the wizard adds to the Web page module becomes the value of the PageProducer (see page 6572) property. If the PageProducer (see page 6572) property is nil (Delphi) or NULL (C++), the Web page module can't generate the content of a Web page. Add additional components to the Web page module that assist the page producer in creating its content. Other modules in the WebSnap application can access any components you add to the Web page module, but typically Web page module components are intended only for internal use.

See Also

TWebAppDataModule (see page 6550)

TWebAppPageModule (see page 6554)

TWebDataModule (see page 6562)

TWebPageModuleFactory (see page 6273)

Web Modules

66.1.14.1 TWebPageModule Members

The following tables list the members exposed by TWebPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6519)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSiteModuleHelper (see page 6526)	Returns the value of the SiteModuleHelper (see page 6521) property.
	GetSitePageModuleHelper (see page 6527)	Returns the value of the SitePageModuleHelper (see page 6528) property.
	GetWebSnapObjClass (see page 6519)	Returns the class type for this adapter field.
	Notification (see page 6519)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6527)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6521)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6522) and OnDeactivate (see page 6522) events.
	SitePageModuleHelper (see page 6528)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6529)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeRedirectToPage (see page 6530)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	Create (see page 6517)	Creates and initializes an instance of TCustomWebDataModule (see page 6516).
	Create (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522).
	CreateNew (see page 6518)	Creates and initializes an instance of TCustomWebDataModule (see page 6516) without using an associated form file.
	CreateNew (see page 6525)	Creates and initializes an instance of TCustomWebPageModule (see page 6522) without using an associated form file.
	Destroy (see page 6518)	Frees an instance of TCustomWebDataModule (see page 6516).
	Destroy (see page 6526)	Frees an instance of TCustomWebPageModule (see page 6522).
	DispatchPage (see page 1006)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 997)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 993)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 970)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 976)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1007)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 997)	Advances to the objects next item.
	NotifyActivate (see page 6061)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6061)	This is NotifyDeactivate, a member of class INotifyWebActivate.

	RedirectToPage (see page 1007)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 957)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6065)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 998)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
R	Request (see page 6520)	Provides access to the current request object.
R	Response (see page 6520)	Provides access to the current response object.
R	Session (see page 6521)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6572)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6573)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6573)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6574)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	DefaultAction (see page 6571)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6572)	Indicates the interface of the page producer that generates the content of the Web page.

66.1.14.2 TWebPageModule Properties

The properties of the TWebPageModule class are listed here.

Published Properties

	Name	Description
	DefaultAction (see page 6571)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6572)	Indicates the interface of the page producer that generates the content of the Web page.

66.1.14.2.1 TWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6528)

66.1.14.2.2 TWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1076)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

66.1.14.3 TWebPageModule Events

The events of the TWebPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6572)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6573)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6573)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6574)	Occurs when the Web module is deactivated.

66.1.14.3.1 TWebPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the

CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6522)

CreateMode

66.1.14.3.2 TWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6530)

PageProducer (see page 6528)

Response (see page 6520)

66.1.14.3.3 TWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 6529)

Request (see page 6520)

Response (see page 6520)

PageProducer (see page 6528)

66.1.14.3.4 TWebPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6522)

66.1.15 TWebPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebPageModuleHelper.

66.1.15.1 TWebPageModuleHelper Members

The following tables list the members exposed by TWebPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6543)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6534)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6534)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6535)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6535)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6535)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6543)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6577)	This is GetPageProducer, a member of class TWebPageModuleHelper.
	GetProducerComponent (see page 6544)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6544)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6544)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6545)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6535)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6535)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6545)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6545)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6545)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6545)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6546)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6546)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6546)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6546)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6547)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6547)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6547)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6547)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6536)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6536)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6536)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6547)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6548)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6548)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6537)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 6537)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6539)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 6549)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6549)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6550)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6550)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6539)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.

	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 6533)	Creates and initializes an instance of TSiteModuleHelper (see page 6531).
	Create (see page 6542)	Creates and initializes an instance of TSitePageModuleHelper (see page 6539).
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 6533)	Frees an instance of TSiteModuleHelper (see page 6531).
	Destroy (see page 6542)	Frees an instance of TSitePageModuleHelper (see page 6539).
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6536)	Responds to notifications that components are about to be inserted or removed.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6548)	Represents the adapter action associated with the Web page module.
R	Request (see page 6537)	Provides access to the current request object.
R	Response (see page 6538)	Provides access to the current response object.
R	Session (see page 6538)	Provides access to the current Web session.
R	WebModuleContext (see page 6538)	Keeps track of name/value pairs that store information for the Web module.

66.1.15.2 TWebPageModuleHelper Methods

The methods of the TWebPageModuleHelper class are listed here.

Protected Methods

	Name	Description
	GetPageProducer (see page 6577)	This is GetPageProducer, a member of class TWebPageModuleHelper.

66.1.15.2.1 TWebPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TWebPageModuleHelper.

66.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FindNamedModuleVariable (see page 6578)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 6578)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 6578)	This is function WebModu.GetModuleVariableCount.

66.2.1 WebModu.FindNamedModuleVariable Function

Pascal

```
function FindNamedModuleVariable(AModule: TComponent; const AName: string): TComponent;
```

C++

```
TComponent * FindNamedModuleVariable(TComponent * AModule, const UnicodeString AName);
```

File

WebModu

Description

This is function WebModu.FindNamedModuleVariable.

66.2.2 WebModu.GetModuleVariable Function

Pascal

```
function GetModuleVariable(AModule: TComponent; Index: Integer): TComponent;
```

C++

```
TComponent * GetModuleVariable(TComponent * AModule, int Index);
```

File

WebModu

Description

This is function WebModu.GetModuleVariable.

66.2.3 WebModu.GetModuleVariableCount Function

Pascal

```
function GetModuleVariableCount(AModule: TComponent): Integer;
```

C++

```
int GetModuleVariableCount(TComponent * AModule);
```

File

WebModu

Description

This is function WebModu.GetModuleVariableCount.

66.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetParamsEvent (see page 6580)	This is type WebModu.TGetParamsEvent.

66.3.1 WebModu.TGetParamsEvent Type

Pascal

```
TGetParamsEvent = procedure (Sender: TObject; Params: TStrings);
```

C++

```
(Sender: TObject; Params: TStrings) ( TGetParamsEvent)();
```

File

WebModu

Description

This is type WebModu.TGetParamsEvent.

67 WebNode Namespace

This is namespace WebNode.

Topics

Name	Description
Interfaces (see page 6582)	The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebNode (see page 6582)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

67.1 Interfaces

The following table lists interfaces in this documentation.

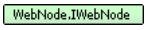
Interfaces

	Name	Description
	IWebNode (see page 6582)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

67.1.1 IWebNode Interface

IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

Class Hierarchy



Pascal

```
IWebNode = interface;
```

C++

```
__interface IWebNode;
```

File

WebNode

Description

TRIO objects use IWebNode to execute method calls on an invokable interface. IWebNode takes an encoded method call, executes it, and returns the encoded results. Typically, objects that implement IWebNode use a specific transport protocol to forward the encoded method call to a remote server. For example, the THTTPReqResp class implements this interface for making interface calls to a remote Web Service application using HTTP.

When implementing the IWebNode interface, you can use another IWebNode implementation to delegate part of the method execution. For example, you can write a class that compresses or encrypts the encoded method call, passes it to another IWebNode implementation that expects the compressed or encrypted string, and then decompresses or decrypts the results returned by the other IWebNode implementation.

Note: C++ method declarations that use IWebNode use the _di_IWebNode type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< >  _di_IWebNode;
```

See Also

THTTPReqResp (see page 1214)

TRio

67.1.1.1 IWebNode Members

The following tables list the members exposed by IWebNode.

Public Methods

	Name	Description
	BeforeExecute (see page 6583)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6584)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6585)	Returns the value of the MimeBoundary (see page 6585) property.
	SetMimeBoundary (see page 6585)	Sets the value of the MimeBoundary (see page 6585) property.

Public Properties

	Name	Description
	MimeBoundary (see page 6585)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

67.1.1.2 IWebNode Methods

The methods of the IWebNode class are listed here.

Public Methods

	Name	Description
	BeforeExecute (see page 6583)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6584)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6585)	Returns the value of the MimeBoundary (see page 6585) property.
	SetMimeBoundary (see page 6585)	Sets the value of the MimeBoundary (see page 6585) property.

67.1.1.2.1 IWebNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute (see page 6584) method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute (see page 6584) method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 6584)

OnBeforeExecute

67.1.1.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	IWebNode.Execute (String, TStream) (see page 6584)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream) (see page 6584)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream, TStream) (see page 6584)	Executes an encoded method call and returns the encoded results in a stream.

67.1.1.2.2.1 IWebNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Response: TStream); overload; deprecated;
procedure Execute(const Request: TStream; Response: TStream); overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall Execute(const UnicodeString DataMsg, TStream Response);
__fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert

BeforeExecute (see page 6583)

67.1.1.2.3 IWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 6585) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary (see page 6585) property.

See Also

MimeBoundary (see page 6585)

67.1.1.2.4 IWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 6585) property.

Pascal

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

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 6585) property.

Value is the new value for MimeBoundary (see page 6585).

See Also

MimeBoundary (see page 6585)

67.1.1.3 IWebNode Properties

The properties of the IWebNode class are listed here.

Public Properties

	Name	Description
	MimeBoundary (see page 6585)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

67.1.1.3.1 IWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property UnicodeString MimeBoundary;
```

Description

When the method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

Execute (see page 6584)

68 WebReq Namespace

This is namespace WebReq.

Topics

Name	Description
Classes (see page 6588)	The following table lists classes in this documentation.
Functions (see page 6608)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6609)	The following table lists structs, records, enums in this documentation.
Variables (see page 6610)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 6588)	
	TWebModuleList (see page 6592)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 6600)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

Functions

	Name	Description
	WebRequestHandler (see page 6608)	Returns a TWebRequestHandler (see page 6600) object.

Structs, Records, Enums

	Name	Description
	TRequestNotification (see page 6609)	This is record WebReq.TRequestNotification.

Variables

Name	Description
WebRequestHandlerProc (see page 6610)	This is variable WebReq.WebRequestHandlerProc.

68.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 6588)	
	TWebModuleList (see page 6592)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 6600)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

68.1.1 TWebModuleFactoryList Class

Class Hierarchy

Pascal

```
TWebModuleFactoryList = class(TObject);
```

C++

```
class TWebModuleFactoryList : public TObject;
```

File

WebReq

See Also

TWebModuleFactory

68.1.1.1 TWebModuleFactoryList Members

The following tables list the members exposed by TWebModuleFactoryList.

Protected Methods

	Name	Description
	GetItem (see page 6591)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 6590)	Adds a factory to the web module factory list.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6589)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6590)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AppModuleFactory (see page 6591)	
R	ItemCount (see page 6591)	Lists the number of items in the web module factory list.
R	Items (see page 6592)	

68.1.1.2 TWebModuleFactoryList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebModuleFactoryList();
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

new

68.1.1.3 TWebModuleFactoryList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleFactoryList();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release (see page 3732) method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

68.1.1.4 TWebModuleFactoryList Methods

The methods of the TWebModuleFactoryList class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 6591)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 6590)	Adds a factory to the web module factory list.

68.1.1.4.1 TWebModuleFactoryList.AddFactory Method

Adds a factory to the web module factory list.

Pascal

```
procedure AddFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddFactory(TAbstractWebModuleFactory AFactory);
```

Description

Use the AddFactory method to add a factory to the web module factory list (more here, obviously).

See Also

AppModuleFactory (see page 6591)

68.1.1.4.2 TWebModuleFactoryList.GetItem Method

Pascal

```
function GetItem(I: Integer): TAbstractWebModuleFactory;
```

C++

```
__fastcall TAbstractWebModuleFactory GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleFactoryList.

68.1.1.5 TWebModuleFactoryList Properties

The properties of the TWebModuleFactoryList class are listed here.

Public Properties

	Name	Description
	AppModuleFactory (see page 6591)	
	ItemCount (see page 6591)	Lists the number of items in the web module factory list.
	Items (see page 6592)	

68.1.1.5.1 TWebModuleFactoryList.AppModuleFactory Property

Pascal

```
property AppModuleFactory: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory AppModuleFactory;
```

See Also

AddFactory (see page 6590)

68.1.1.5.2 TWebModuleFactoryList.ItemCount Property

Lists the number of items in the web module factory list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Use the ItemCount property to list the number of items in the web module factory list.

See Also

Items (see page 6592)

68.1.1.5.3 TWebModuleFactoryList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory Items[int I];
```

See Also

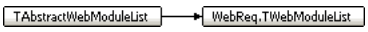
AddFactory (see page 6590)

ItemCount (see page 6591)

68.1.2 TWebModuleList Class

TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.

Class Hierarchy



Pascal

```
TWebModuleList = class(TAbstractWebModuleList);
```

C++

```
class TWebModuleList : public TAbstractWebModuleList;
```

File

WebReq

Description

TWebModuleList contains every Web module that has been instantiated in order to serve the current request. In addition, it lists the factory objects for all Web modules in the application, including any that are not instantiated.

When a WebSnap application handles multiple requests concurrently, it includes multiple Web modules lists, each on a separate thread and each associated with a different request.

The Web context, which can be accessed using the global WebContext (see page 5977) function, provides access to the TWebModuleList object that is associated with the current request. Use the Web context's WebModules property to obtain a reference to the Web module list.

See Also

TWebModuleFactoryList (see page 6588)

TAbstractWebModuleFactory

WebContext

TAbstractWebContext (see page 5935)

68.1.2.1 TWebModuleList Members

The following tables list the members exposed by TWebModuleList.

Protected Methods

	Name	Description
	GetFactory (see page 6598)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 6598)	
	GetItem (see page 6598)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 6599)	
	GetOnModuleAdded (see page 6599)	
	SetOnModuleAdded (see page 6599)	

Protected Properties

	Name	Description
	Factories (see page 6599)	This is Factories, a member of class TWebModuleList.

Public Methods

	Name	Description
	AddModule (see page 6594)	Adds a module to the Items property array.
	AddModuleClass (see page 6595)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 6595)	Creates a new Web module, given its name, and adds it to the Items property array.
	AutoCreateModules (see page 6596)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 6596)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	Create (see page 6593)	Creates and initializes a TWebModuleList (see page 6592) object.
	Destroy (see page 6594)	Frees a TWebModuleList (see page 6592) instance.
	FindModuleClass (see page 6597)	Locates a Web module given its class reference.
	FindModuleName (see page 6597)	Locates a Web module given its name.

68.1.2.2 TWebModuleList.Create Constructor

Creates and initializes a TWebModuleList (see page 6592) object.

Pascal

```
constructor Create(const AFactories: TWebModuleFactoryList);
```

C++

```
__fastcall TWebModuleList(const TWebModuleFactoryList AFactories);
```

Description

There is no need to call the TWebModuleList (see page 6592) constructor. Web module lists are created automatically for each incoming HTTP request.

AFactories is the list of Web module factories in the WebSnap application. TWebModuleList (see page 6592) uses this list to implement the Factory property.

See Also

Factory

68.1.2.3 TWebModuleList.Destroy Destructor

Frees a TWebModuleList (see page 6592) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleList();
```

Description

Do not free TWebModuleList (see page 6592) objects. Web module lists are freed automatically when the WebSnap application finishes processing the HTTP request.

68.1.2.4 TWebModuleList Methods

The methods of the TWebModuleList class are listed here.

Protected Methods

	Name	Description
	GetFactory (see page 6598)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 6598)	
	GetItem (see page 6598)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 6599)	
	GetOnModuleAdded (see page 6599)	
	SetOnModuleAdded (see page 6599)	

Public Methods

	Name	Description
	AddModule (see page 6594)	Adds a module to the Items property array.
	AddModuleClass (see page 6595)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 6595)	Creates a new Web module, given its name, and adds it to the Items property array.
	AutoCreateModules (see page 6596)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 6596)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	FindModuleClass (see page 6597)	Locates a Web module given its class reference.
	FindModuleName (see page 6597)	Locates a Web module given its name.

68.1.2.4.1 TWebModuleList.AddModule Method

Adds a module to the Items property array.

Pascal

```
procedure AddModule(AComponent: TComponent);
```

C++

```
__fastcall AddModule(TComponent * AComponent);
```

Description

Call AddModule to add a newly instantiated Web module to the Items property array.

AComponent is a Web module instance that is not already listed in the Items property array. It must be one of the objects created by a Web module factory from the Factory property array.

See Also

AddModuleName (see page 6595)

AddModuleClass (see page 6595)

Items

Factory

68.1.2.4.2 TWebModuleList.AddModuleClass Method

Creates a new Web module, given its class, and adds it to the Items property array.

Pascal

```
function AddModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleClass(TComponentClass AClass);
```

Description

AddModuleClass searches for a Web module factory that can create a Web module of the specified class. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AClass is a class reference to the Web module you want to instantiate. It should not match the class type of any Web module already listed in the Items property array.

AddModuleClass returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified class, AddModuleClass returns nil (Delphi) or NULL (C++).

See Also

AddModuleName (see page 6595)

AddModule (see page 6594)

Items

Factory

FindModuleClass (see page 6597)

68.1.2.4.3 TWebModuleList.AddModuleName Method

Creates a new Web module, given its name, and adds it to the Items property array.

Pascal

```
function AddModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleName(const UnicodeString AName);
```

Description

AddModuleName searches for a Web module factory that can create a Web module of the specified name. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AName is the name of the Web module you want to instantiate. It should not match the name of any Web module already listed in the Items property array.

AddModuleClass (see page 6595) returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified name, AddModuleClass (see page 6595) returns nil (Delphi) or NULL (C++).

See Also

AddModuleClass (see page 6595)

AddModule (see page 6594)

Items

Factory

FindModuleName (see page 6597)

68.1.2.4.4 TWebModuleList.AutoCreateModules Method

Instantiates the Web modules that are flagged to handle all incoming requests.

Pascal

```
procedure AutoCreateModules;
```

C++

```
__fastcall AutoCreateModules();
```

Description

TWebRequestHandler (see page 6600) calls AutoCreateModules to instantiate all of the Web modules that should be used to handle every incoming request object.

AutoCreateModules searches the module factory list for factory objects that have a CreateMode property with the crAlways flag. For each such factory object, AutoCreateModules creates the corresponding Web module if it does not already exist and adds it to the Items property array.

See Also

CreateMode

AutoDestroyModules (see page 6596)

Items

Factory

TWebRequestHandler (see page 6600)

68.1.2.4.5 TWebModuleList.AutoDestroyModules Method

Frees all Web modules that are flagged as inappropriate for caching between requests.

Pascal

```
procedure AutoDestroyModules;
```

C++

```
__fastcall AutoDestroyModules();
```

Description

TWebRequestHandler (see page 6600) calls AutoDestroyModules when it finishes processing an HTTP request message. This allows the Web module list to free those Web modules that should not be cached for reuse with another request message.

AutoDestroyModules searches the module factory list for factory objects that have a CacheMode property with the caDestroy flag. For each such factory object, AutoDestroyModules frees the Web module if it exists and removes it from Items property array.

See Also

CacheMode

AutoCreateModules (see page 6596)

Items

TWebRequestHandler (see page 6600)

Factory

68.1.2.4.6 TWebModuleList.FindModuleClass Method

Locates a Web module given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to locate the Web module instance of a given class type. FindModuleClass searches the Items property array for a Web module instance of the specified class.

AClass is a class reference to the class type of the desired Web module.

FindModuleClass returns the requested Web module instance if it exists. If there is no Web module of the specified class, FindModuleClass returns nil (Delphi) or NULL (C++).

See Also

FindModuleName (see page 6597)

Items

68.1.2.4.7 TWebModuleList.FindModuleName Method

Locates a Web module given its name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AName);
```

Description

Call FindModuleName to locate the Web module instance of a given name. FindModuleClass (see page 6597) searches the Items property array for a Web module instance of the specified name.

AName is the name of the desired Web module.

FindModuleName returns the requested Web module instance if it exists. If there is no Web module with the specified name, FindModuleClass (see page 6597) returns nil (Delphi) or NULL (C++).

See Also

FindModuleClass (see page 6597)

Items

68.1.2.4.8 TWebModuleList.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractWebModuleFactory; override;
```

C++

```
virtual __fastcall TAbstractWebModuleFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TWebModuleList.

68.1.2.4.9 TWebModuleList.GetFactoryCount Method

Pascal

```
function GetFactoryCount: Integer; override;
```

C++

```
virtual __fastcall int GetFactoryCount();
```

See Also

GetFactory

FactoryCount

68.1.2.4.10 TWebModuleList.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleList.

68.1.2.4.11 TWebModuleList.GetItemCount Method

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

See Also

GetItem

ItemCount

68.1.2.4.12 TWebModuleList.GetOnModuleAdded Method

Pascal

```
function GetOnModuleAdded: TModuleAddedProc; override;
```

C++

```
virtual __fastcall TModuleAddedProc GetOnModuleAdded();
```

See Also

OnModuleAdded

SetOnModuleAdded

68.1.2.4.13 TWebModuleList.SetOnModuleAdded Method

Pascal

```
procedure SetOnModuleAdded(AProc: TModuleAddedProc); override;
```

C++

```
virtual __fastcall SetOnModuleAdded(TModuleAddedProc AProc);
```

See Also

GetOnModuleAdded

OnModuleAdded

68.1.2.5 TWebModuleList Properties

The properties of the TWebModuleList class are listed here.

Protected Properties

	Name	Description
R	Factories (see page 6599)	This is Factories, a member of class TWebModuleList.

68.1.2.5.1 TWebModuleList.Factories Property

Pascal

```
property Factories: TWebModuleFactoryList;
```

C++

```
__property TWebModuleFactoryList Factories;
```

Description

This is Factories, a member of class TWebModuleList.

68.1.3 TWebRequestHandler Class

TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

Class Hierarchy

Pascal

```
TWebRequestHandler = class(TComponent);
```

C++

```
class TWebRequestHandler : public TComponent;
```

File

WebReq

Description

TWebRequestHandler keeps a pool of active Web modules. In response to a request from the application, TWebRequestHandler creates a request object and assigns it to one of the active Web modules.

Do not create instances of TWebRequestHandler. Use TWebRequestHandler as a base class when declaring nonvisual components that can appear on the component palette and be used in the form designer. Properties and methods of TWebRequestHandler provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TWebApplication (see page 5912)

TWebModule

68.1.3.1 TWebRequestHandler Members

The following tables list the members exposed by TWebRequestHandler.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6602)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6603)	Deactivates the Web module of the application.
	HandleRequest (see page 6603)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6602)	Adds a Web module factory to the request handler.

	Create (see page 6601)	Creates an instance of TWebRequestHandler (see page 6600).
	Destroy (see page 6601)	Destroys the TWebRequestHandler (see page 6600) instance and frees its memory.
	HandleException (see page 6603)	Responds to exceptions related to TWebRequestHandler (see page 6600) objects.

Public Properties

	Name	Description
	ActiveCount (see page 6604)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6605)	Determines whether the Web modules used by the application are cached for re-use.
	InactiveCount (see page 6605)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6606)	Sets or returns the MaxConnections used by the socket.
	WebModuleClass (see page 6606)	Contains a reference to a Web module.
	WebModuleFactory (see page 6607)	A list of Web module factory objects.
	WebModuleFactoryCount (see page 6607)	The number of Web module factories in use by the request handler.

68.1.3.2 TWebRequestHandler.Create Constructor

Creates an instance of TWebRequestHandler (see page 6600).

Pascal

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

C++

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

Description

Call Create to instantiate a TWebRequestHandler (see page 6600) object.

See Also

Destroy (see page 6601)

68.1.3.3 TWebRequestHandler.Destroy Destructor

Destroys the TWebRequestHandler (see page 6600) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestHandler();
```

Description

Destroy cleans up references in the global fix-up list for form linking. It then calls the inherited destructor.

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

System::TObject::Destroy (see page 3366)

68.1.3.4 TWebRequestHandler Methods

The methods of the TWebRequestHandler class are listed here.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6602)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6603)	Deactivates the Web module of the application.
	HandleRequest (see page 6603)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6602)	Adds a Web module factory to the request handler.
	HandleException (see page 6603)	Responds to exceptions related to TWebRequestHandler (see page 6600) objects.

68.1.3.4.1 TWebRequestHandler.ActivateWebModules Method

Activates a Web module for the application and returns it.

Pascal

```
function ActivateWebModules: TWebModuleList;
```

C++

```
__fastcall TWebModuleList ActivateWebModules();
```

Description

ActivateWebModules is called internally when the application receives an HTTP request message. If the InactiveCount (see page 6605) property is greater than zero, a cached Web module is reactivated. Otherwise, a new instance of the Web module is created.

Before ActivateWebModules is called, the CreateForm method must be called to register the data module.

Note: In cross-platform applications, a Web application object contains only a single Web module. Usually, that Web module is a TWebModule object, but it may also be an ordinary data module that contains a TWebDispatcher object. If no Web module is associated with the Web application, ActivateWebModule raises an exception. If the ActiveCount (see page 6604) property is already at MaxConnections (see page 6606) when ActivateWebModules is called, ActivateWebModules raises an exception.

See Also

CreateForm (see page 5914)

ActiveCount (see page 6604)

InactiveCount (see page 6605)

MaxConnections (see page 6606)

68.1.3.4.2 TWebRequestHandler.AddWebModuleFactory Method

Adds a Web module factory to the request handler.

Pascal

```
procedure AddWebModuleFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddWebModuleFactory(TAbstractWebModuleFactory AFactory);
```

Description

AddWebModuleFactory is used to add a Web module factory to the Web request handler. Web module factories are for internal use, and should not require any attention from developers.

68.1.3.4.3 TWebRequestHandler.DeactivateWebModules Method

Deactivates the Web module of the application.

Pascal

```
procedure DeactivateWebModules(WebModules: TWebModuleList);
```

C++

```
__fastcall DeactivateWebModules(TWebModuleList WebModules);
```

Description

Do not call DeactivateWebModules directly. The Web application calls DeactivateWebModules internally after handling an HTTP request message. When CacheConnections (see page 6605) is true, the Web module remains inactive until the next request message is received. When CacheConnections (see page 6605) is false, the runtime instance of the Web module is destroyed.

See Also

ActivateWebModules (see page 6602)

CacheConnections (see page 6605)

HandleRequest (see page 6603)

InactiveCount (see page 6605)

68.1.3.4.4 TWebRequestHandler.HandleException Method

Responds to exceptions related to TWebRequestHandler (see page 6600) objects.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

When exceptions occur in TWebRequestHandler (see page 6600) and descendant objects, HandleException will attempt to deal with it. If it cannot do so, it displays an error message to the user.

68.1.3.4.5 TWebRequestHandler.HandleRequest Method

Responds to an HTTP request message received by the server.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

When a descendant of TWebApplication receives an HTTP request message, it creates a TWebRequest object to represent the message, and a TWebResponse message to match the request. It then calls HandleRequest, passing those objects as the Request and Response parameters.

HandleRequest activates the Web module associated with the Web application. It then passes the request and response objects to the dispatcher, so that it can dispatch the request to the appropriate action items. When the dispatcher returns, HandleRequest checks whether the response was handled but not sent, and if not, sends it. Finally, HandleRequest deactivates the data module.

HandleRequest returns a Boolean value indicating whether the request message was handled and sent.

See Also

ActivateWebModules (see page 6602)

DeactivateWebModules (see page 6603)

68.1.3.5 TWebRequestHandler Properties

The properties of the TWebRequestHandler class are listed here.

Public Properties

	Name	Description
	ActiveCount (see page 6604)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6605)	Determines whether the Web modules used by the application are cached for re-use.
	InactiveCount (see page 6605)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6606)	Sets or returns the MaxConnections used by the socket.
	WebModuleClass (see page 6606)	Contains a reference to a Web module.
	WebModuleFactory (see page 6607)	A list of Web module factory objects.
	WebModuleFactoryCount (see page 6607)	The number of Web module factories in use by the request handler.

68.1.3.5.1 TWebRequestHandler.ActiveCount Property

Indicates the number of active connections currently being serviced by the application.

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

Use ActiveCount to monitor the number of HTTP request messages the application is servicing at any one time. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 6606) property.

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it generates a new instance of the Web module. After the response is returned to the Web client, the Web module is either destroyed or deactivated and cached for later reuse. Each active Web module resides in a separate execution thread.

ActiveCount is the number of Web modules currently in use. InactiveCount (see page 6605) is the number of deactivated Web modules cached for later reuse. MaxConnections (see page 6606) is the maximum number of Web modules that can be active at any time. ActiveCount can never exceed MaxConnections (see page 6606).

Note: ActiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) applications. Each CGI application handles only a single request message.

See Also

InactiveCount (see page 6605)

68.1.3.5.2 TWebRequestHandler.CacheConnections Property

Determines whether the Web modules used by the application are cached for re-use.

Pascal

```
property CacheConnections: Boolean;
```

C++

```
__property Boolean CacheConnections;
```

Description

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it spawns a thread to handle the request and creates a new Web module that contains instances of all the objects used in the Web application. To improve performance, these Web modules can be saved in a cache when the thread associated with the HTTP request message finishes executing.

Set CacheConnections to true to have the Web application save inactive Web modules in a cache. Caching the Web modules speeds performance as new instances of the Web module and its contents do not need to be created every time a request message is received. Set CacheConnections to false to have the Web application create a new Web module for every request thread. Recreating the Web module ensures that changes made to the module or its contents when servicing the last request do not cause unexpected behavior the next time the module is activated.

CacheConnections is only relevant to Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) Web applications.

See Also

MaxConnections (see page 6606)

68.1.3.5.3 TWebRequestHandler.InactiveCount Property

Indicates the number of Web modules cached for later reuse.

Pascal

```
property InactiveCount: Integer;
```

C++

```
__property int InactiveCount;
```

Description

Use InactiveCount to monitor the number of unused Web modules. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 6606) property.

When CacheConnections (see page 6605) is true, each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it reactivates one of the inactive Web modules. The application generates a new instance of the Web module if InactiveCount is 0. After the response is returned to the Web client, the Web module is deactivated again, and counted by the InactiveCount property.

When CacheConnections (see page 6605) is false, a new Web module is created for each HTTP request message and

destroyed when the response is returned. InactiveCount is always 0 when CacheConnections (see page 6605) is false.

Note: InactiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

ActiveCount (see page 6604)

68.1.3.5.4 TWebRequestHandler.MaxConnections Property

Sets or returns the MaxConnections used by the socket.

Pascal

```
property MaxConnections: Integer;
```

C++

```
__property int MaxConnections;
```

Description

Set MaxConnections to adjust the number of HTTP request messages the application can handle simultaneously. Use the ActiveCount (see page 6604) and InactiveCount (see page 6605) properties to get a sense of the frequency and duration of request messages. Then set MaxConnections so that it is high enough to avoid exceptions when the application runs out of Web modules, but low enough not to degrade performance too severely by running too many simultaneous threads.

If a request message is received when ActiveCount (see page 6604) is already at MaxConnections, the application raises an exception.

Note: InactiveCount (see page 6605) is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

CacheConnections (see page 6605)

68.1.3.5.5 TWebRequestHandler.WebModuleClass Property

Contains a reference to a Web module.

Pascal

```
property WebModuleClass: TComponentClass;
```

C++

```
__property TComponentClass WebModuleClass;
```

Description

WebModuleClass is a reference to a Web module, and is for internal use only. When ActivateWebModules (see page 6602) is called, it is used in that method to help generate infrastructure for a Web module.

See Also

ActivateWebModules (see page 6602)

68.1.3.5.6 TWebRequestHandler.WebModuleFactory Property

A list of Web module factory objects.

Pascal

```
property WebModuleFactory [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory WebModuleFactory[int I];
```

Description

This is a list of Web module factory objects, which are intended for internal use.

See Also

WebModuleFactoryCount (see page 6607)

68.1.3.5.7 TWebRequestHandler.WebModuleFactoryCount Property

The number of Web module factories in use by the request handler.

Pascal

```
property WebModuleFactoryCount: Integer;
```

C++

```
__property int WebModuleFactoryCount;
```

Description

WebModuleFactoryCount is the number of Web module factories in use by the request handler. This property is intended for internal use.

See Also

WebModuleFactory (see page 6607)

68.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebRequestHandler (see page 6608)	Returns a TWebRequestHandler (see page 6600) object.

68.2.1 WebReq.WebRequestHandler Function

Returns a TWebRequestHandler (see page 6600) object.

Pascal

```
function WebRequestHandler: TWebRequestHandler;
```

C++

```
TWebRequestHandler WebRequestHandler();
```

File

WebReq

Description

Use the WebRequestHandler function to obtain a reference to the global TWebRequestHandler (see page 6600) object. This object manages all the Web modules in the application, and creates Web request and response objects when the application receives HTTP request messages.

See Also

TWebApplication

TWebModule

68.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRequestNotification (see page 6609)	This is record WebReq.TRequestNotification.

68.3.1 WebReq.TRequestNotification Enumeration

Pascal

```
TRequestNotification = (
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
);
```

C++

```
enum TRequestNotification {
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
};
```

File

WebReq

Description

This is record WebReq.TRequestNotification.

68.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WebRequestHandlerProc (see page 6610)	This is variable WebReq.WebRequestHandlerProc.

68.4.1 WebReq.WebRequestHandlerProc Variable

Pascal

```
WebRequestHandlerProc: function: TWebRequestHandler = nil;
```

C++

```
function: TWebRequestHandler WebRequestHandlerProc = nil;
```

File

WebReq

Description

This is variable WebReq.WebRequestHandlerProc.

69 WebScript Namespace

This is namespace WebScript.

Topics

Name	Description
Classes (see page 6613)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6686)	The following table lists structs, records, enums in this documentation.
Types (see page 6688)	The following table lists types in this documentation.
Constants (see page 6689)	The following table lists constants in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 6613)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 6614)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 6614)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 6618)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 6621)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 6623)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 6628)	This is class WebScript.THTMLItem.
	THTMLItems (see page 6631)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 6634)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 6637)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 6639)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 6643)	This is class WebScript.TScriptError.
	TScriptErrors (see page 6647)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 6649)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 6653)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 6658)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 6659)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 6662)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 6664)	This is class WebScript.TScriptItem.
	TScriptItems (see page 6666)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 6668)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 6673)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 6682)	This is class WebScript.TVariableLookup.

Constants

Name	Description
GlobalScriptVariableNames (see page 6689)	This is constant WebScript.GlobalScriptVariableNames.

Structs, Records, Enums

	Name	Description
	TGlobalScriptVariables (see page 6686)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 6687)	This is record WebScript.TIncludeType.
	TLineNumbers (see page 6687)	This is record WebScript.TLineNumbers.

Types

Name	Description
CreateComponentStubFactory (see page 6688)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 6688)	This is type WebScript.PLineNumbers.

69.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 6613)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 6614)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 6614)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 6618)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 6621)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 6623)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 6628)	This is class WebScript.THTMLItem.
	THTMLItems (see page 6631)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 6634)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 6637)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 6639)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 6643)	This is class WebScript.TScriptError.
	TScriptErrors (see page 6647)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 6649)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 6653)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 6658)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 6659)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 6662)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 6664)	This is class WebScript.TScriptItem.
	TScriptItems (see page 6666)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 6668)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 6673)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 6682)	This is class WebScript.TVariableLookup.

69.1.1 ERecursiveIncludeFile Class

Class Hierarchy

Pascal

```
ERecursiveIncludeFile = class(EScriptError);
```

C++

```
class ERecursiveIncludeFile : public EScriptError;
```

File

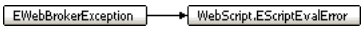
WebScript

Description

This is class WebScript.ERecursiveIncludeFile.

69.1.2 EScriptEvalError Class

Class Hierarchy



Pascal

```
EScriptEvalError = class(EWebBrokerException);
```

C++

```
class EScriptEvalError : public EWebBrokerException;
```

File

WebScript

Description

This is class WebScript.EScriptEvalError.

69.1.3 TAbstractScriptEngine Class

Class Hierarchy

Pascal

```
TAbstractScriptEngine = class(TObject);
```

C++

```
class TAbstractScriptEngine : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptEngine.

69.1.3.1 TAbstractScriptEngine Members

The following tables list the members exposed by TAbstractScriptEngine.

Public Methods

	Name	Description
	AddObjectFactory (see page 6616)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6615)	This is Create, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 6617)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6615)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
A	ListLanguageNames (see page 6617)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
A	SupportsLanguage (see page 6617)	This is SupportsLanguage, a member of class TAbstractScriptEngine.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Name (see page 6617)	This is Name, a member of class TAbstractScriptEngine.
R	ScriptObjectFactories (see page 6618)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 6618)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

69.1.3.2 TAbstractScriptEngine.Create Constructor

Pascal

```
constructor Create(const AName: string);
```

C++

```
__fastcall TAbstractScriptEngine(const UnicodeString AName);
```

Description

This is Create, a member of class TAbstractScriptEngine.

69.1.3.3 TAbstractScriptEngine.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractScriptEngine();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.3.4 TAbstractScriptEngine Methods

The methods of the TAbstractScriptEngine class are listed here.

Public Methods

	Name	Description
	AddObjectFactory (see page 6616)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 6617)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
	ListLanguageNames (see page 6617)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	SupportsLanguage (see page 6617)	This is SupportsLanguage, a member of class TAbstractScriptEngine.

69.1.3.4.1 TAbstractScriptEngine.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory);
```

Description

This is AddObjectFactory, a member of class TAbstractScriptEngine.

69.1.3.4.2 TAbstractScriptEngine.CreateComponentStub Method

Pascal

```
function CreateComponentStub(AObject: TObject): IInterface;
```

C++

```
__fastcall IInterface CreateComponentStub(TObject * AObject);
```

Description

This is CreateComponentStub, a member of class TAbstractScriptEngine.

69.1.3.4.3 TAbstractScriptEngine.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ListLanguageNames(TStrings AList) = 0;
```

Description

This is ListLanguageNames, a member of class TAbstractScriptEngine.

69.1.3.4.4 TAbstractScriptEngine.SupportsLanguage Method

Pascal

```
function SupportsLanguage(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean SupportsLanguage(const UnicodeString AName) = 0;
```

Description

This is SupportsLanguage, a member of class TAbstractScriptEngine.

69.1.3.5 TAbstractScriptEngine Properties

The properties of the TAbstractScriptEngine class are listed here.

Public Properties

	Name	Description
	Name (see page 6617)	This is Name, a member of class TAbstractScriptEngine.
	ScriptObjectFactories (see page 6618)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 6618)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

69.1.3.5.1 TAbstractScriptEngine.Name Property

Pascal

```
property Name: string;
```

C++

```
__property UnicodeString Name;
```

Description

This is Name, a member of class TAbstractScriptEngine.

69.1.3.5.2 TAbstractScriptEngine.ScriptObjectFactories Property

Pascal

```
property ScriptObjectFactories: TAbstractScriptObjectFactories;
```

C++

```
__property TAbstractScriptObjectFactories ScriptObjectFactories;
```

Description

This is ScriptObjectFactories, a member of class TAbstractScriptEngine.

69.1.3.5.3 TAbstractScriptEngine.ScriptProducerClass Property

Pascal

```
property ScriptProducerClass: TScriptProducerClass;
```

C++

```
__property TScriptProducerClass ScriptProducerClass;
```

Description

This is ScriptProducerClass, a member of class TAbstractScriptEngine.

69.1.4 TAbstractScriptObjectFactories Class

Class Hierarchy

Pascal

```
TAbstractScriptObjectFactories = class(TObject);
```

C++

```
class TAbstractScriptObjectFactories : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactories.

69.1.4.1 TAbstractScriptObjectFactories Members

The following tables list the members exposed by TAbstractScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 6620)	This is GetCount, a member of class TAbstractScriptObjectFactories.
	GetFactory (see page 6620)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
	GetObjectList (see page 6620)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6621)	This is Count, a member of class TAbstractScriptObjectFactories.
	Factories (see page 6621)	This is Factories, a member of class TAbstractScriptObjectFactories.
	ObjectList (see page 6621)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

69.1.4.2 TAbstractScriptObjectFactories Methods

The methods of the TAbstractScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6620)	This is GetCount, a member of class TAbstractScriptObjectFactories.
	GetFactory (see page 6620)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
	GetObjectList (see page 6620)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

69.1.4.2.1 TAbstractScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

This is GetCount, a member of class TAbstractScriptObjectFactories.

69.1.4.2.2 TAbstractScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I) = 0;
```

Description

This is GetFactory, a member of class TAbstractScriptObjectFactories.

69.1.4.2.3 TAbstractScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; virtual; abstract;
```

C++

```
virtual __fastcall TObjectList GetObjectList() = 0;
```

Description

This is GetObjectList, a member of class TAbstractScriptObjectFactories.

69.1.4.3 TAbstractScriptObjectFactories Properties

The properties of the TAbstractScriptObjectFactories class are listed here.

Public Properties

	Name	Description
	Count (see page 6621)	This is Count, a member of class TAbstractScriptObjectFactories.
	Factories (see page 6621)	This is Factories, a member of class TAbstractScriptObjectFactories.
	ObjectList (see page 6621)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

69.1.4.3.1 TAbstractScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAbstractScriptObjectFactories.

69.1.4.3.2 TAbstractScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TAbstractScriptObjectFactories.

69.1.4.3.3 TAbstractScriptObjectFactories.ObjectList Property

Pascal

```
property ObjectList: TObjectList;
```

C++

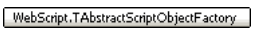
```
__property TObjectList ObjectList;
```

Description

This is ObjectList, a member of class TAbstractScriptObjectFactories.

69.1.5 TAbstractScriptObjectFactory Class

Class Hierarchy



Pascal

```
TAbstractScriptObjectFactory = class;
```

C++

```
class TAbstractScriptObjectFactory;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactory.

69.1.5.1 TAbstractScriptObjectFactory Members

The following tables list the members exposed by TAbstractScriptObjectFactory.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 6622)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 6622)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 6623)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 6623)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6623)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

69.1.5.2 TAbstractScriptObjectFactory Methods

The methods of the TAbstractScriptObjectFactory class are listed here.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 6622)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 6622)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 6623)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 6623)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6623)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

69.1.5.2.1 TAbstractScriptObjectFactory.AddGlobalObjects Method

Pascal

```
procedure AddGlobalObjects(AScriptGlobalObjects: TScriptGlobalObjects); virtual;
```

C++

```
virtual __fastcall AddGlobalObjects(TScriptGlobalObjects AScriptGlobalObjects);
```

Description

This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.

69.1.5.2.2 TAbstractScriptObjectFactory.CreateComponentObject Method

Pascal

```
function CreateComponentObject(AComponent: TObject; AClassKey: TClass): IInterface; virtual;
abstract;
```

C++

```
virtual __fastcall IInterface CreateComponentObject(TObject * AComponent, TClass AClassKey) =
0;
```

Description

This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.

69.1.5.2.3 TAbstractScriptObjectFactory.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable,
IScriptContext AContext) = 0;
```

Description

This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

69.1.5.2.4 TAbstractScriptObjectFactory.CreateProducerObject Method

Pascal

```
function CreateProducerObject(AVariable: TGlobalScriptVariables; AProducer: IScriptProducer):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateProducerObject(TGlobalScriptVariables AVariable,
IScriptProducer AProducer) = 0;
```

Description

This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.

69.1.5.2.5 TAbstractScriptObjectFactory.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

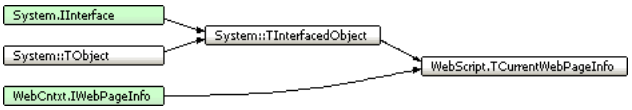
```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

69.1.6 TCurrentWebPageInfo Class

Class Hierarchy



Pascal

```
TCurrentWebPageInfo = class(TInterfacedObject, IWebPageInfo);
```

C++

```
class TCurrentWebPageInfo : public TInterfacedObject, public IWebPageInfo;
```

File

WebScript

Description

This is class WebScript.TCurrentWebPageInfo.

69.1.6.1 TCurrentWebPageInfo Members

The following tables list the members exposed by TCurrentWebPageInfo.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetHasViewAccess (see page 6626)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 6626)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 6626)	Returns the value of the PageDescription property.
	GetPageHREF (see page 6627)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 6627)	Returns the value of the PageIsPublished property.
	GetPageName (see page 6627)	Returns the value of the PageName property.
	GetPageTitle (see page 6628)	Returns the value of the PageTitle property.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	WebPageInfo (see page 6628)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6625)	This is Create, a member of class TCurrentWebPageInfo.

	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	HasViewAccess (see page 5974)	Indicates whether the current user has access rights to view the Web page.
	LoginRequired (see page 5974)	Indicates whether callers must log in before accessing this Web page.
	PageDescription (see page 5974)	Describes the Web page.
	PageHREF (see page 5975)	Identifies the HREF to use for references to the associated Web page.
	PageIsPublished (see page 5975)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 5976).
	PageName (see page 5976)	Denotes the logical name for the Web page.
	PageTitle (see page 5976)	Indicates the title that appears on the Web page.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

69.1.6.2 TCurrentWebPageInfo.Create Constructor

Pascal

```
constructor Create(AContext: IScriptContext);
```

C++

```
__fastcall TCurrentWebPageInfo(IScriptContext AContext);
```

Description

This is Create, a member of class TCurrentWebPageInfo.

69.1.6.3 TCurrentWebPageInfo Methods

The methods of the TCurrentWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetHasViewAccess (see page 6626)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 6626)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 6626)	Returns the value of the PageDescription property.
	GetPageHREF (see page 6627)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 6627)	Returns the value of the PageIsPublished property.
	GetPageName (see page 6627)	Returns the value of the PageName property.
	GetPageTitle (see page 6628)	Returns the value of the PageTitle property.
	WebPageInfo (see page 6628)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

69.1.6.3.1 TCurrentWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess property.

See Also

HasViewAccess (see page 5974)

69.1.6.3.2 TCurrentWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired property.

See Also

LoginRequired (see page 5974)

69.1.6.3.3 TCurrentWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription property.

See Also

PageDescription (see page 5974)

69.1.6.3.4 TCurrentWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF property.

See Also

PageHREF (see page 5975)

69.1.6.3.5 TCurrentWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished property.

See Also

PageIsPublished (see page 5975)

69.1.6.3.6 TCurrentWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall UnicodeString GetPageName();
```

Description

GetPageName is the read implementation of the PageName property.

See Also

PageName (see page 5976)

69.1.6.3.7 TCurrentWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle property.

See Also

PageTitle (see page 5976)

69.1.6.3.8 TCurrentWebPageInfo.WebPageInfo Method

Pascal

```
function WebPageInfo: IWebPageInfo;
```

C++

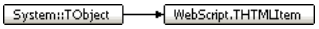
```
__fastcall IWebPageInfo WebPageInfo();
```

Description

This is WebPageInfo, a member of class TCurrentWebPageInfo.

69.1.7 THTMLItem Class

Class Hierarchy



Pascal

```
THTMLItem = class(TObject);
```

C++

```
class THTMLItem : public TObject;
```

File

WebScript

Description

This is class WebScript.THTMLItem.

69.1.7.1 THTMLItem Members

The following tables list the members exposed by THTMLItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	AssignString (see page 6630)	This is AssignString, a member of class THTMLItem.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6630)	This is Create, a member of class THTMLItem.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	HTML (see page 6630)	This is HTML, a member of class THTMLItem.
	Ident (see page 6631)	This is Ident, a member of class THTMLItem.
	ItemLength (see page 6631)	This is ItemLength, a member of class THTMLItem.
	TagToken (see page 6631)	This is TagToken, a member of class THTMLItem.

69.1.7.2 THTMLItem.Create Constructor

Pascal

```
constructor Create(AItems: THTMLItems);
```

C++

```
__fastcall THTMLItem(THTMLItems AItems);
```

Description

This is Create, a member of class THTMLItem.

69.1.7.3 THTMLItem Methods

The methods of the THTMLItem class are listed here.

Public Methods

	Name	Description
	AssignString (see page 6630)	This is AssignString, a member of class THTMLItem.

69.1.7.3.1 THTMLItem.AssignString Method

Pascal

```
procedure AssignString(AOffset: Integer; AHTML: PAnsiChar; ALength: Integer; AEncoding:
TEncoding);
```

C++

```
__fastcall AssignString(int AOffset, PAnsiChar AHTML, int ALength, TEncoding AEncoding);
```

Description

This is AssignString, a member of class THTMLItem.

69.1.7.4 THTMLItem Properties

The properties of the THTMLItem class are listed here.

Public Properties

	Name	Description
R	HTML (see page 6630)	This is HTML, a member of class THTMLItem.
R	Ident (see page 6631)	This is Ident, a member of class THTMLItem.
R	ItemLength (see page 6631)	This is ItemLength, a member of class THTMLItem.
	TagToken (see page 6631)	This is TagToken, a member of class THTMLItem.

69.1.7.4.1 THTMLItem.HTML Property

Pascal

```
property HTML: string;
```

C++

```
__property UnicodeString HTML;
```

Description

This is HTML, a member of class THTMLItem.

69.1.7.4.2 THTMLItem.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property UnicodeString Ident;
```

Description

This is Ident, a member of class THTMLItem.

69.1.7.4.3 THTMLItem.ItemLength Property

Pascal

```
property ItemLength: Integer;
```

C++

```
__property int ItemLength;
```

Description

This is ItemLength, a member of class THTMLItem.

69.1.7.4.4 THTMLItem.TagToken Property

Pascal

```
property TagToken: Boolean;
```

C++

```
__property Boolean TagToken;
```

Description

This is TagToken, a member of class THTMLItem.

69.1.8 THTMLItems Class

Class Hierarchy

Pascal

```
THTMLItems = class(TObjectList);
```

C++

```
class THTMLItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.THTMLItems.

69.1.8.1 THTMLItems Members

The following tables list the members exposed by THTMLItems.

Public Methods

	Name	Description
	Create (see page 6632)	This is Create, a member of class THTMLItems.
V	Destroy (see page 6632)	Destroys an instance of TList.
	GetHTMLItem (see page 6633)	This is GetHTMLItem, a member of class THTMLItems.
	MakeIdentifier (see page 6633)	This is MakeIdentifier, a member of class THTMLItems.

Public Properties

	Name	Description
R	Items (see page 6633)	Lists the object references.

69.1.8.2 THTMLItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall THTMLItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class THTMLItems.

69.1.8.3 THTMLItems.Destroy Destructor

Destroys an instance of TList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTMLItems();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TList reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of items.

Note: Destroy does not free the memory pointed to by the elements of the list.

See Also

System::TObject::Free (see page 3382)

69.1.8.4 THTMLItems Methods

The methods of the THTMLItems class are listed here.

Public Methods

	Name	Description
	GetHTMLItem (see page 6633)	This is GetHTMLItem, a member of class THTMLItems.
	MakeIdentifier (see page 6633)	This is MakeIdentifier, a member of class THTMLItems.

69.1.8.4.1 THTMLItems.GetHTMLItem Method

Pascal

```
function GetHTMLItem(I: Integer): THTMLItem;
```

C++

```
__fastcall THTMLItem GetHTMLItem(int I);
```

Description

This is GetHTMLItem, a member of class THTMLItems.

69.1.8.4.2 THTMLItems.MakeIdentifier Method

Pascal

```
function MakeIdentifier(AItem: THTMLItem): string;
```

C++

```
__fastcall UnicodeString MakeIdentifier(THTMLItem AItem);
```

Description

This is MakeIdentifier, a member of class THTMLItems.

69.1.8.5 THTMLItems Properties

The properties of the THTMLItems class are listed here.

Public Properties

	Name	Description
	Items (see page 6633)	Lists the object references.

69.1.8.5.1 THTMLItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: THTMLItem;
```

C++

```
__property THTMLItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

Count

Clear

Pack

operator_sb

69.1.9 TLineNumberMap Class

Class Hierarchy

Pascal

```
TLineNumberMap = class(TObject);
```

C++

```
class TLineNumberMap : public TObject;
```

File

WebScript

Description

This is class WebScript.TLineNumberMap.

69.1.9.1 TLineNumberMap Members

The following tables list the members exposed by TLineNumberMap.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	DestLineToSourceLine (see page 6636)	This is DestLineToSourceLine, a member of class TLineNumberMap.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	New (see page 6636)	This is New, a member of class TLineNumberMap.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6636)	This is Count, a member of class TLineNumberMap.
	Items (see page 6637)	This is Items, a member of class TLineNumberMap.

69.1.9.2 TLineNumberMap Methods

The methods of the TLineNumberMap class are listed here.

Public Methods

	Name	Description
	DestLineToSourceLine (see page 6636)	This is DestLineToSourceLine, a member of class TLineNumberMap.
	New (see page 6636)	This is New, a member of class TLineNumberMap.

69.1.9.2.1 TLineNumberMap.DestLineToSourceLine Method

Pascal

```
function DestLineToSourceLine(ALine: Integer; var AFileName: string): Integer;
```

C++

```
__fastcall int DestLineToSourceLine(int ALine, UnicodeString AFileName);
```

Description

This is DestLineToSourceLine, a member of class TLineNumberMap.

69.1.9.2.2 TLineNumberMap.New Method

Pascal

```
function New: PLineNumbers;
```

C++

```
__fastcall PLineNumbers New();
```

Description

This is New, a member of class TLineNumberMap.

69.1.9.3 TLineNumberMap Properties

The properties of the TLineNumberMap class are listed here.

Public Properties

	Name	Description
R	Count (see page 6636)	This is Count, a member of class TLineNumberMap.
R	Items (see page 6637)	This is Items, a member of class TLineNumberMap.

69.1.9.3.1 TLineNumberMap.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TLineNumberMap.

69.1.9.3.2 TLineNumberMap.Items Property

Pascal

```
property Items [I: Integer]: PLineNumbers;
```

C++

```
__property PLineNumbers Items[int I];
```

Description

This is Items, a member of class TLineNumberMap.

69.1.10 TScriptCodeItem Class

Class Hierarchy

Pascal

```
TScriptCodeItem = class(TScriptItem);
```

C++

```
class TScriptCodeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptCodeItem.

69.1.10.1 TScriptCodeItem Members

The following tables list the members exposed by TScriptCodeItem.

Protected Methods

	Name	Description
	GetScript (see page 6639)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	AssignString (see page 6638)	This is AssignString, a member of class TScriptCodeItem.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.

	Create (see page 6665)	This is Create, a member of class TScriptItem.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6666)	This is Script, a member of class TScriptItem.

69.1.10.2 TScriptCodeItem Methods

The methods of the TScriptCodeItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6639)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AssignString (see page 6638)	This is AssignString, a member of class TScriptCodeItem.

69.1.10.2.1 TScriptCodeItem.AssignString Method

Pascal

```
procedure AssignString(const AValue: string);
```

C++

```
__fastcall AssignString(const UnicodeString AValue);
```

Description

This is AssignString, a member of class TScriptCodeItem.

69.1.10.2.2 TScriptCodeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

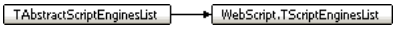
```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptCodeItem.

69.1.11 TScriptEnginesList Class

Class Hierarchy



Pascal

```
TScriptEnginesList = class(TAbstractScriptEnginesList);
```

C++

```
class TScriptEnginesList : public TAbstractScriptEnginesList;
```

File

WebScript

Description

This is class WebScript.TScriptEnginesList.

69.1.11.1 TScriptEnginesList Members

The following tables list the members exposed by TScriptEnginesList.

Public Methods

	Name	Description
	AddEngine (see page 6641)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 6641)	This is AddObjectFactory, a member of class TScriptEnginesList.
	Create (see page 6640)	Constructs an object and initializes its data before the object is first used.
V	Destroy (see page 6640)	Disposes of an object instance.
	Find (see page 6641)	This is Find, a member of class TScriptEnginesList.
V	FindScriptProducerClass (see page 6642)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	ListLanguageNames (see page 6642)	This is ListLanguageNames, a member of class TScriptEnginesList.
	RemoveEngine (see page 6642)	This is RemoveEngine, a member of class TScriptEnginesList.

Public Properties

	Name	Description
R	Count (see page 6642)	This is Count, a member of class TScriptEnginesList.
R	Items (see page 6643)	This is Items, a member of class TScriptEnginesList.

69.1.11.2 TScriptEnginesList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptEnginesList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

69.1.11.3 TScriptEnginesList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptEnginesList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.11.4 TScriptEnginesList Methods

The methods of the TScriptEnginesList class are listed here.

Public Methods

	Name	Description
	AddEngine (see page 6641)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 6641)	This is AddObjectFactory, a member of class TScriptEnginesList.
	Find (see page 6641)	This is Find, a member of class TScriptEnginesList.
	FindScriptProducerClass (see page 6642)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	ListLanguageNames (see page 6642)	This is ListLanguageNames, a member of class TScriptEnginesList.
	RemoveEngine (see page 6642)	This is RemoveEngine, a member of class TScriptEnginesList.

69.1.11.4.1 TScriptEnginesList.AddEngine Method

Pascal

```
procedure AddEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall AddEngine(TAbstractScriptEngine AEngine);
```

Description

This is AddEngine, a member of class TScriptEnginesList.

69.1.11.4.2 TScriptEnginesList.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory; const AEngineName: string);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory, const UnicodeString
AEngineName);
```

Description

This is AddObjectFactory, a member of class TScriptEnginesList.

69.1.11.4.3 TScriptEnginesList.Find Method

Pascal

```
function Find(const AName: string): TAbstractScriptEngine;
```

C++

```
__fastcall TAbstractScriptEngine Find(const UnicodeString AName);
```

Description

This is Find, a member of class TScriptEnginesList.

69.1.11.4.4 TScriptEnginesList.FindScriptProducerClass Method

Pascal

```
function FindScriptProducerClass(const ALanguageName: string): TScriptProducerClass; override;
```

C++

```
virtual __fastcall TScriptProducerClass FindScriptProducerClass(const UnicodeString
ALanguageName);
```

Description

This is FindScriptProducerClass, a member of class TScriptEnginesList.

69.1.11.4.5 TScriptEnginesList.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings);
```

C++

```
__fastcall ListLanguageNames(TStrings AList);
```

Description

This is ListLanguageNames, a member of class TScriptEnginesList.

69.1.11.4.6 TScriptEnginesList.RemoveEngine Method

Pascal

```
procedure RemoveEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall RemoveEngine(TAbstractScriptEngine AEngine);
```

Description

This is RemoveEngine, a member of class TScriptEnginesList.

69.1.11.5 TScriptEnginesList Properties

The properties of the TScriptEnginesList class are listed here.

Public Properties

	Name	Description
	Count (see page 6642)	This is Count, a member of class TScriptEnginesList.
	Items (see page 6643)	This is Items, a member of class TScriptEnginesList.

69.1.11.5.1 TScriptEnginesList.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptEnginesList.

69.1.11.5.2 TScriptEnginesList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractScriptEngine;
```

C++

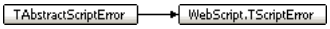
```
__property TAbstractScriptEngine Items[int I];
```

Description

This is Items, a member of class TScriptEnginesList.

69.1.12 TScriptError Class

Class Hierarchy



Pascal

```
TScriptError = class(TAbstractScriptError);
```

C++

```
class TScriptError : public TAbstractScriptError;
```

File

WebScript

Description

This is class WebScript.TScriptError.

69.1.12.1 TScriptError Members

The following tables list the members exposed by TScriptError.

Protected Fields

	Name	Description
	FCharPos (see page 6644)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 6644)	This is FDescription, a member of class TScriptError.
	FFileName (see page 6645)	This is FFileName, a member of class TScriptError.
	FLine (see page 6645)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 6645)	This is FSourceLine, a member of class TScriptError.

Protected Methods

	Name	Description
	GetCharPos (see page 6645)	This is GetCharPos, a member of class TScriptError.
	GetDescription (see page 6646)	This is GetDescription, a member of class TScriptError.
	GetFileName (see page 6646)	This is GetFileName, a member of class TScriptError.
	GetLine (see page 6646)	This is GetLine, a member of class TScriptError.

	GetSourceLine (see page 6646)	This is GetSourceLine, a member of class TScriptError.
	SetDescription (see page 6647)	This is SetDescription, a member of class TScriptError.

Public Methods

	Name	Description
	Create (see page 6644)	This is Create, a member of class TScriptError.

69.1.12.2 TScriptError.Create Constructor

Pascal

```
constructor Create(E: Exception); overload;
```

C++

```
__fastcall TScriptError(Exception E);
```

Description

This is Create, a member of class TScriptError.

69.1.12.3 TScriptError Fields

The fields of the TScriptError class are listed here.

Protected Fields

	Name	Description
	FCharPos (see page 6644)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 6644)	This is FDescription, a member of class TScriptError.
	FFileName (see page 6645)	This is FFileName, a member of class TScriptError.
	FLine (see page 6645)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 6645)	This is FSourceLine, a member of class TScriptError.

69.1.12.3.1 TScriptError.FCharPos Field

Pascal

```
FCharPos: Integer;
```

C++

```
int FCharPos;
```

Description

This is FCharPos, a member of class TScriptError.

69.1.12.3.2 TScriptError.FDescription Field

Pascal

```
FDescription: string;
```

C++

```
UnicodeString FDescription;
```

Description

This is FDescription, a member of class TScriptError.

69.1.12.3.3 TScriptError.FFileName Field

Pascal

```
FFileName: string;
```

C++

```
UnicodeString FFileName;
```

Description

This is FFileName, a member of class TScriptError.

69.1.12.3.4 TScriptError.FLine Field

Pascal

```
FLine: Integer;
```

C++

```
int FLine;
```

Description

This is FLine, a member of class TScriptError.

69.1.12.3.5 TScriptError.FSourceLine Field

Pascal

```
FSourceLine: string;
```

C++

```
UnicodeString FSourceLine;
```

Description

This is FSourceLine, a member of class TScriptError.

69.1.12.4 TScriptError Methods

The methods of the TScriptError class are listed here.

Protected Methods

	Name	Description
	GetCharPos (see page 6645)	This is GetCharPos, a member of class TScriptError.
	GetDescription (see page 6646)	This is GetDescription, a member of class TScriptError.
	GetFileName (see page 6646)	This is GetFileName, a member of class TScriptError.
	GetLine (see page 6646)	This is GetLine, a member of class TScriptError.
	GetSourceLine (see page 6646)	This is GetSourceLine, a member of class TScriptError.
	SetDescription (see page 6647)	This is SetDescription, a member of class TScriptError.

69.1.12.4.1 TScriptError.GetCharPos Method

Pascal

```
function GetCharPos: Integer; override;
```

C++

```
virtual __fastcall int GetCharPos();
```

Description

This is GetCharPos, a member of class TScriptError.

69.1.12.4.2 TScriptError.GetDescription Method

Pascal

```
function GetDescription: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDescription();
```

Description

This is GetDescription, a member of class TScriptError.

69.1.12.4.3 TScriptError.GetFileName Method

Pascal

```
function GetFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetFileName();
```

Description

This is GetFileName, a member of class TScriptError.

69.1.12.4.4 TScriptError.GetLine Method

Pascal

```
function GetLine: Integer; override;
```

C++

```
virtual __fastcall int GetLine();
```

Description

This is GetLine, a member of class TScriptError.

69.1.12.4.5 TScriptError.GetSourceLine Method

Pascal

```
function GetSourceLine: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSourceLine();
```

Description

This is GetSourceLine, a member of class TScriptError.

69.1.12.4.6 TScriptError.SetDescription Method

Pascal

```
procedure SetDescription(const AValue: string); override;
```

C++

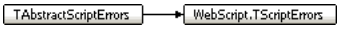
```
virtual __fastcall SetDescription(const UnicodeString AValue);
```

Description

This is SetDescription, a member of class TScriptError.

69.1.13 TScriptErrors Class

Class Hierarchy



Pascal

```
TScriptErrors = class(TAbstractScriptErrors);
```

C++

```
class TScriptErrors : public TAbstractScriptErrors;
```

File

WebScript

Description

This is class WebScript.TScriptErrors.

69.1.13.1 TScriptErrors Members

The following tables list the members exposed by TScriptErrors.

Protected Methods

	Name	Description
	GetError (see page 6649)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 6649)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 6649)	This is Add, a member of class TScriptErrors.
	Create (see page 6647)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 6648)	Disposes of an object instance.

69.1.13.2 TScriptErrors.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptErrors();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

69.1.13.3 TScriptErrors.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptErrors();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.13.4 TScriptErrors Methods

The methods of the TScriptErrors class are listed here.

Protected Methods

	Name	Description
	GetError (see page 6649)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 6649)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 6649)	This is Add, a member of class TScriptErrors.

69.1.13.4.1 TScriptErrors.Add Method

Pascal

```
procedure Add(const AError: TAbstractScriptError); override;
```

C++

```
virtual __fastcall Add(const TAbstractScriptError AError);
```

Description

This is Add, a member of class TScriptErrors.

69.1.13.4.2 TScriptErrors.GetError Method

Pascal

```
function GetError(I: Integer): TAbstractScriptError; override;
```

C++

```
virtual __fastcall TAbstractScriptError GetError(int I);
```

Description

This is GetError, a member of class TScriptErrors.

69.1.13.4.3 TScriptErrors.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int GetErrorCount();
```

Description

This is GetErrorCount, a member of class TScriptErrors.

69.1.14 TScriptFile Class

Class Hierarchy

Pascal

```
TScriptFile = class;
```

C++

```
class TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptFile.

69.1.14.1 TScriptFile Members

The following tables list the members exposed by TScriptFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 6651)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 6650)	This is Create, a member of class TScriptFile.
V	Destroy (see page 6650)	This is Destroy, a member of class TScriptFile.
	ParseStream (see page 6651)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6651)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 6652)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 6652)	This is LongName, a member of class TScriptFile.
R	Script (see page 6652)	This is Script, a member of class TScriptFile.
R	ShortName (see page 6653)	This is ShortName, a member of class TScriptFile.
R	Source (see page 6653)	This is Source, a member of class TScriptFile.

69.1.14.2 TScriptFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptFile(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptFile.

69.1.14.3 TScriptFile.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptFile();
```

Description

This is Destroy, a member of class TScriptFile.

69.1.14.4 TScriptFile Fields

The fields of the TScriptFile class are listed here.

Public Fields

	Name	Description
	FLineNumberMap (see page 6651)	This is FLineNumberMap, a member of class TScriptFile.

69.1.14.4.1 TScriptFile.FLineNumberMap Field

Pascal

```
FLineNumberMap: TLineNumberMap;
```

C++

```
TLineNumberMap FLineNumberMap;
```

Description

This is FLineNumberMap, a member of class TScriptFile.

69.1.14.5 TScriptFile Methods

The methods of the TScriptFile class are listed here.

Public Methods

	Name	Description
	ParseStream (see page 6651)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6651)	This is ParseString, a member of class TScriptFile.

69.1.14.5.1 TScriptFile.ParseStream Method

Pascal

```
procedure ParseStream(ASource: TStream; AOwned: Boolean);
```

C++

```
__fastcall ParseStream(TStream ASource, Boolean AOwned);
```

Description

This is ParseStream, a member of class TScriptFile.

69.1.14.5.2 TScriptFile.ParseString Method

Pascal

```
procedure ParseString(const ASource: string);
```

C++

```
__fastcall ParseString(const UnicodeString ASource);
```

Description

This is ParseString, a member of class TScriptFile.

69.1.14.6 TScriptFile Properties

The properties of the TScriptFile class are listed here.

Public Properties

	Name	Description
	ApplicationName (see page 6652)	This is ApplicationName, a member of class TScriptFile.
	LongName (see page 6652)	This is LongName, a member of class TScriptFile.
	Script (see page 6652)	This is Script, a member of class TScriptFile.
	ShortName (see page 6653)	This is ShortName, a member of class TScriptFile.
	Source (see page 6653)	This is Source, a member of class TScriptFile.

69.1.14.6.1 TScriptFile.ApplicationName Property

Pascal

```
property ApplicationName: string;
```

C++

```
__property UnicodeString ApplicationName;
```

Description

This is ApplicationName, a member of class TScriptFile.

69.1.14.6.2 TScriptFile.LongName Property

Pascal

```
property LongName: string;
```

C++

```
__property UnicodeString LongName;
```

Description

This is LongName, a member of class TScriptFile.

69.1.14.6.3 TScriptFile.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptFile.

69.1.14.6.4 TScriptFile.ShortName Property

Pascal

```
property ShortName: string;
```

C++

```
__property UnicodeString ShortName;
```

Description

This is ShortName, a member of class TScriptFile.

69.1.14.6.5 TScriptFile.Source Property

Pascal

```
property Source: string;
```

C++

```
__property UnicodeString Source;
```

Description

This is Source, a member of class TScriptFile.

69.1.15 TScriptGlobalObjects Class

Class Hierarchy

Pascal

```
TScriptGlobalObjects = class(TObject);
```

C++

```
class TScriptGlobalObjects : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptGlobalObjects.

69.1.15.1 TScriptGlobalObjects Members

The following tables list the members exposed by TScriptGlobalObjects.

Protected Methods

	Name	Description
	GetScriptEngine (see page 6656)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 6656)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6654)	This is Create, a member of class TScriptGlobalObjects.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6655)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindNamedItemIntf (see page 6656)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamedItemCount (see page 6656)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 6656)	This is GetNamedItemName, a member of class TScriptGlobalObjects.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	NamedItemCount (see page 6657)	This is NamedItemCount, a member of class TScriptGlobalObjects.
R	NamedItemName (see page 6657)	This is NamedItemName, a member of class TScriptGlobalObjects.
R	Producer (see page 6657)	This is Producer, a member of class TScriptGlobalObjects.
R	ScriptEngine (see page 6658)	This is ScriptEngine, a member of class TScriptGlobalObjects.

69.1.15.2 TScriptGlobalObjects.Create Constructor

Pascal

```
constructor Create(AProducer: IScriptProducer);
```

C++

```
__fastcall TScriptGlobalObjects(IScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptGlobalObjects.

69.1.15.3 TScriptGlobalObjects.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptGlobalObjects();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.15.4 TScriptGlobalObjects Methods

The methods of the TScriptGlobalObjects class are listed here.

Protected Methods

	Name	Description
	GetScriptEngine (see page 6656)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 6656)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
	FindNamedItemIntf (see page 6656)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	GetNamedItemCount (see page 6656)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 6656)	This is GetNamedItemName, a member of class TScriptGlobalObjects.

69.1.15.4.1 TScriptGlobalObjects.AddNamedIntf Method

Pascal

```
procedure AddNamedIntf(const AName: string; AIntf: IUnknown);
```

C++

```
__fastcall AddNamedIntf(const UnicodeString AName, IUnknown AIntf);
```

Description

This is AddNamedIntf, a member of class TScriptGlobalObjects.

69.1.15.4.2 TScriptGlobalObjects.FindNamedItemIntf Method

Pascal

```
function FindNamedItemIntf(const AName: string): IUnknown;
```

C++

```
__fastcall IUnknown FindNamedItemIntf(const UnicodeString AName);
```

Description

This is FindNamedItemIntf, a member of class TScriptGlobalObjects.

69.1.15.4.3 TScriptGlobalObjects.GetNamedItemCount Method

Pascal

```
function GetNamedItemCount: Integer;
```

C++

```
__fastcall int GetNamedItemCount();
```

Description

This is GetNamedItemCount, a member of class TScriptGlobalObjects.

69.1.15.4.4 TScriptGlobalObjects.GetNamedItemName Method

Pascal

```
function GetNamedItemName(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetNamedItemName(int I);
```

Description

This is GetNamedItemName, a member of class TScriptGlobalObjects.

69.1.15.4.5 TScriptGlobalObjects.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TScriptGlobalObjects.

69.1.15.5 TScriptGlobalObjects Properties

The properties of the TScriptGlobalObjects class are listed here.

Public Properties

	Name	Description
	NamedItemCount (see page 6657)	This is NamedItemCount, a member of class TScriptGlobalObjects.
	NamedItemName (see page 6657)	This is NamedItemName, a member of class TScriptGlobalObjects.
	Producer (see page 6657)	This is Producer, a member of class TScriptGlobalObjects.
	ScriptEngine (see page 6658)	This is ScriptEngine, a member of class TScriptGlobalObjects.

69.1.15.5.1 TScriptGlobalObjects.NamedItemCount Property

Pascal

```
property NamedItemCount: Integer;
```

C++

```
__property int NamedItemCount;
```

Description

This is NamedItemCount, a member of class TScriptGlobalObjects.

69.1.15.5.2 TScriptGlobalObjects.NamedItemName Property

Pascal

```
property NamedItemName [I: Integer]: string;
```

C++

```
__property UnicodeString NamedItemName[int I];
```

Description

This is NamedItemName, a member of class TScriptGlobalObjects.

69.1.15.5.3 TScriptGlobalObjects.Producer Property

Pascal

```
property Producer: IScriptProducer;
```

C++

```
__property IScriptProducer Producer;
```

Description

This is Producer, a member of class TScriptGlobalObjects.

69.1.15.5.4 TScriptGlobalObjects.ScriptEngine Property

Pascal

```
property ScriptEngine: TAbstractScriptEngine;
```

C++

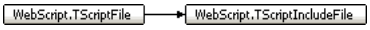
```
__property TAbstractScriptEngine ScriptEngine;
```

Description

This is ScriptEngine, a member of class TScriptGlobalObjects.

69.1.16 TScriptIncludeFile Class

Class Hierarchy



Pascal

```
TScriptIncludeFile = class(TScriptFile);
```

C++

```
class TScriptIncludeFile : public TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFile.

69.1.16.1 TScriptIncludeFile Members

The following tables list the members exposed by TScriptIncludeFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 6651)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 6659)	This is Create, a member of class TScriptIncludeFile.
V	Destroy (see page 6650)	This is Destroy, a member of class TScriptFile.
	LoadFile (see page 6659)	This is LoadFile, a member of class TScriptIncludeFile.
	ParseStream (see page 6651)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6651)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 6652)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 6652)	This is LongName, a member of class TScriptFile.
R	Script (see page 6652)	This is Script, a member of class TScriptFile.

	ShortName (see page 6653)	This is ShortName, a member of class TScriptFile.
	Source (see page 6653)	This is Source, a member of class TScriptFile.

69.1.16.2 TScriptIncludeFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer; AIncludeType: TIncludeType; const AFileName:
string);
```

C++

```
__fastcall TScriptIncludeFile(TScriptProducer AProducer, TIncludeType AIncludeType, const
UnicodeString AFileName);
```

Description

This is Create, a member of class TScriptIncludeFile.

69.1.16.3 TScriptIncludeFile Methods

The methods of the TScriptIncludeFile class are listed here.

Public Methods

	Name	Description
	LoadFile (see page 6659)	This is LoadFile, a member of class TScriptIncludeFile.

69.1.16.3.1 TScriptIncludeFile.LoadFile Method

Pascal

```
procedure LoadFile;
```

C++

```
__fastcall LoadFile();
```

Description

This is LoadFile, a member of class TScriptIncludeFile.

69.1.17 TScriptIncludeFiles Class

Class Hierarchy

Pascal

```
TScriptIncludeFiles = class(TObjectList);
```

C++

```
class TScriptIncludeFiles : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFiles.

69.1.17.1 TScriptIncludeFiles Members

The following tables list the members exposed by TScriptIncludeFiles.

Public Methods

	Name	Description
	Create (see page 6660)	This is Create, a member of class TScriptIncludeFiles.
	GetScriptIncludeFile (see page 6660)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

Public Properties

	Name	Description
	Items (see page 6661)	Lists the object references.

69.1.17.2 TScriptIncludeFiles.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptIncludeFiles(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptIncludeFiles.

69.1.17.3 TScriptIncludeFiles Methods

The methods of the TScriptIncludeFiles class are listed here.

Public Methods

	Name	Description
	GetScriptIncludeFile (see page 6660)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

69.1.17.3.1 TScriptIncludeFiles.GetScriptIncludeFile Method

Pascal

```
function GetScriptIncludeFile(I: Integer): TScriptIncludeFile;
```

C++

```
__fastcall TScriptIncludeFile GetScriptIncludeFile(int I);
```

Description

This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

69.1.17.4 TScriptIncludeFiles Properties

The properties of the TScriptIncludeFiles class are listed here.

Public Properties

	Name	Description
	Items (see page 6661)	Lists the object references.

69.1.17.4.1 TScriptIncludeFiles.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptIncludeFile;
```

C++

```
__property TScriptIncludeFile Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

Count

Clear

Pack

operator_sb

69.1.18 TScriptIncludeItem Class

Class Hierarchy

Pascal

```
TScriptIncludeItem = class(TScriptItem);
```

C++

```
class TScriptIncludeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeItem.

69.1.18.1 TScriptIncludeItem Members

The following tables list the members exposed by TScriptIncludeItem.

Protected Methods

	Name	Description
	GetScript (see page 6663)	This is GetScript, a member of class TScriptIncludeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6663)	This is Create, a member of class TScriptIncludeItem.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.

	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6666)	This is Script, a member of class TScriptItem.

69.1.18.2 TScriptIncludeItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems; AIncludeFile: TScriptIncludeFile);
```

C++

```
__fastcall TScriptIncludeItem(TScriptItems AItems, TScriptIncludeFile AIncludeFile);
```

Description

This is Create, a member of class TScriptIncludeItem.

69.1.18.3 TScriptIncludeItem Methods

The methods of the TScriptIncludeItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6663)	This is GetScript, a member of class TScriptIncludeItem.

69.1.18.3.1 TScriptIncludeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptIncludeItem.

69.1.19 TScriptItem Class

Class Hierarchy

Pascal

```
TScriptItem = class(TObject);
```

C++

```
class TScriptItem : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptItem.

69.1.19.1 TScriptItem Members

The following tables list the members exposed by TScriptItem.

Protected Methods

	Name	Description
	GetScript (see page 6665)	This is GetScript, a member of class TScriptItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6665)	This is Create, a member of class TScriptItem.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6666)	This is Script, a member of class TScriptItem.

69.1.19.2 TScriptItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems);
```

C++

```
__fastcall TScriptItem(TScriptItems AItems);
```

Description

This is Create, a member of class TScriptItem.

69.1.19.3 TScriptItem Methods

The methods of the TScriptItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6665)	This is GetScript, a member of class TScriptItem.

69.1.19.3.1 TScriptItem.GetScript Method

Pascal

```
function GetScript: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptItem.

69.1.19.4 TScriptItem Properties

The properties of the TScriptItem class are listed here.

Public Properties

	Name	Description
	Script (see page 6666)	This is Script, a member of class TScriptItem.

69.1.19.4.1 TScriptItem.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptItem.

69.1.20 TScriptItems Class

Class Hierarchy

Pascal

```
TScriptItems = class(TObjectList);
```

C++

```
class TScriptItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptItems.

69.1.20.1 TScriptItems Members

The following tables list the members exposed by TScriptItems.

Public Methods

	Name	Description
	Create (see page 6667)	This is Create, a member of class TScriptItems.
	GetScriptItem (see page 6667)	This is GetScriptItem, a member of class TScriptItems.

Public Properties

	Name	Description
	Items (see page 6667)	Lists the object references.

69.1.20.2 TScriptItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptItems.

69.1.20.3 TScriptItems Methods

The methods of the TScriptItems class are listed here.

Public Methods

	Name	Description
	GetScriptItem (see page 6667)	This is GetScriptItem, a member of class TScriptItems.

69.1.20.3.1 TScriptItems.GetScriptItem Method

Pascal

```
function GetScriptItem(I: Integer): TScriptItem;
```

C++

```
__fastcall TScriptItem GetScriptItem(int I);
```

Description

This is GetScriptItem, a member of class TScriptItems.

69.1.20.4 TScriptItems Properties

The properties of the TScriptItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 6667)	Lists the object references.

69.1.20.4.1 TScriptItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptItem;
```

C++

```
__property TScriptItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed

as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

Count

Clear

Pack

operator_sb

69.1.21 TScriptObjectFactories Class

Class Hierarchy

Pascal

```
TScriptObjectFactories = class(TAbstractScriptObjectFactories);
```

C++

```
class TScriptObjectFactories : public TAbstractScriptObjectFactories;
```

File

WebScript

Description

This is class WebScript.TScriptObjectFactories.

69.1.21.1 TScriptObjectFactories Members

The following tables list the members exposed by TScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 6671)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 6671)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 6672)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6670)	Constructs an object and initializes its data before the object is first used.
	CreateGlobalObject (see page 6671)	This is CreateGlobalObject, a member of class TScriptObjectFactories.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 6670)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6672)	This is Count, a member of class TScriptObjectFactories.
	Factories (see page 6672)	This is Factories, a member of class TScriptObjectFactories.
	ObjectList (see page 6621)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

69.1.21.2 TScriptObjectFactories.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptObjectFactories();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

69.1.21.3 TScriptObjectFactories.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptObjectFactories();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.21.4 TScriptObjectFactories Methods

The methods of the TScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6671)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 6671)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 6672)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6671)	This is CreateGlobalObject, a member of class TScriptObjectFactories.

69.1.21.4.1 TScriptObjectFactories.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface;
```

C++

```
__fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable, IScriptContext
AContext);
```

Description

This is CreateGlobalObject, a member of class TScriptObjectFactories.

69.1.21.4.2 TScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TScriptObjectFactories.

69.1.21.4.3 TScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; override;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TScriptObjectFactories.

69.1.21.4.4 TScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; override;
```

C++

```
virtual __fastcall TObjectList GetObjectList();
```

Description

This is GetObjectList, a member of class TScriptObjectFactories.

69.1.21.5 TScriptObjectFactories Properties

The properties of the TScriptObjectFactories class are listed here.

Public Properties

	Name	Description
	Count (see page 6672)	This is Count, a member of class TScriptObjectFactories.
	Factories (see page 6672)	This is Factories, a member of class TScriptObjectFactories.

69.1.21.5.1 TScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptObjectFactories.

69.1.21.5.2 TScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

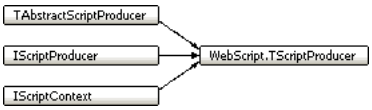
```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TScriptObjectFactories.

69.1.22 TScriptProducer Class

Class Hierarchy



Pascal

```
TScriptProducer = class(TAbstractScriptProducer, IScriptProducer, IScriptContext);
```

C++

```
class TScriptProducer : public TAbstractScriptProducer, public IScriptProducer, public
IScriptContext;
```

File

WebScript

Description

This is class WebScript.TScriptProducer.

69.1.22.1 TScriptProducer Members

The following tables list the members exposed by TScriptProducer.

Protected Fields

	Name	Description
	FErrors (see page 6675)	This is FErrors, a member of class TScriptProducer.
	FScriptEngine (see page 6675)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 6676)	This is FScriptFile, a member of class TScriptProducer.

Protected Methods

	Name	Description
	Evaluate (see page 6676)	This is Evaluate, a member of class TScriptProducer.
A	EvaluateExpression (see page 6676)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 6677)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 6677)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 6677)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 6677)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 6678)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 6678)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 6678)	This is HandleScriptError, a member of class TScriptProducer.
A	ImplHandleScriptError (see page 6678)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 6678)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 6679)	This is ParseString, a member of class TScriptProducer.
	ReplaceTags (see page 6679)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 6679)	This is SetContent, a member of class TScriptProducer.
	Write (see page 6679)	This is the overview for the Write method overload.
	WriteItem (see page 6680)	This is WriteItem, a member of class TScriptProducer.

Protected Properties

	Name	Description
	Component (see page 6681)	This is Component, a member of class TScriptProducer.
	Content (see page 6681)	This is Content, a member of class TScriptProducer.
	Designing (see page 6681)	This is Designing, a member of class TScriptProducer.
	Errors (see page 6681)	This is Errors, a member of class TScriptProducer.
	Script (see page 6682)	This is Script, a member of class TScriptProducer.

Public Methods

	Name	Description
	Create (see page 6674)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 6674)	Disposes of an object instance.

69.1.22.2 TScriptProducer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebModuleContext: TWebModuleContext; AStripParamQuotes: Boolean;
AHandleTag: THandleTagProc; const AScriptEngine: string; ALocateFileService:
ILocateFileService); override;
```

C++

```
virtual __fastcall TScriptProducer(TWebModuleContext AWebModuleContext, Boolean
AStripParamQuotes, THandleTagProc AHandleTag, const UnicodeString AScriptEngine,
ILocateFileService ALocateFileService);
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

new

69.1.22.3 TScriptProducer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptProducer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil

before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

69.1.22.4 TScriptProducer Fields

The fields of the TScriptProducer class are listed here.

Protected Fields

	Name	Description
	FErrors (see page 6675)	This is FErrors, a member of class TScriptProducer.
	FScriptEngine (see page 6675)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 6676)	This is FScriptFile, a member of class TScriptProducer.

69.1.22.4.1 TScriptProducer.FErrors Field

Pascal

```
FErrors: TAbstractScriptErrors;
```

C++

```
TAbstractScriptErrors FErrors;
```

Description

This is FErrors, a member of class TScriptProducer.

69.1.22.4.2 TScriptProducer.FScriptEngine Field

Pascal

```
FScriptEngine: string;
```

C++

```
UnicodeString FScriptEngine;
```

Description

This is FScriptEngine, a member of class TScriptProducer.

69.1.22.4.3 TScriptProducer.FScriptFile Field

Pascal

```
FScriptFile: TScriptFile;
```

C++

```
TScriptFile FScriptFile;
```

Description

This is FScriptFile, a member of class TScriptProducer.

69.1.22.5 TScriptProducer Methods

The methods of the TScriptProducer class are listed here.

Protected Methods

	Name	Description
	Evaluate (see page 6676)	This is Evaluate, a member of class TScriptProducer.
A	EvaluateExpression (see page 6676)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 6677)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 6677)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 6677)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 6677)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 6678)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 6678)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 6678)	This is HandleScriptError, a member of class TScriptProducer.
A	ImplHandleScriptError (see page 6678)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 6678)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 6679)	This is ParseString, a member of class TScriptProducer.
	ReplaceTags (see page 6679)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 6679)	This is SetContent, a member of class TScriptProducer.
	Write (see page 6679)	This is the overview for the Write method overload.
	WriteItem (see page 6680)	This is WriteItem, a member of class TScriptProducer.

69.1.22.5.1 TScriptProducer.Evaluate Method

Pascal

```
function Evaluate: string;
```

C++

```
__fastcall UnicodeString Evaluate();
```

Description

This is Evaluate, a member of class TScriptProducer.

69.1.22.5.2 TScriptProducer.EvaluateExpression Method

Pascal

```
function EvaluateExpression(AScriptFile: TScriptFile): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString EvaluateExpression(TScriptFile AScriptFile) = 0;
```

Description

This is EvaluateExpression, a member of class TScriptProducer.

69.1.22.5.3 TScriptProducer.GetContent Method

Pascal

```
function GetContent: string;
```

C++

```
__fastcall UnicodeString GetContent();
```

Description

This is GetContent, a member of class TScriptProducer.

69.1.22.5.4 TScriptProducer.GetErrors Method

Pascal

```
function GetErrors: TAbstractScriptErrors;
```

C++

```
__fastcall TAbstractScriptErrors GetErrors();
```

Description

This is GetErrors, a member of class TScriptProducer.

69.1.22.5.5 TScriptProducer.GetHTMLBlock Method

Pascal

```
function GetHTMLBlock(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetHTMLBlock(int I);
```

Description

This is GetHTMLBlock, a member of class TScriptProducer.

69.1.22.5.6 TScriptProducer.GetHTMLBlockCount Method

Pascal

```
function GetHTMLBlockCount: Integer;
```

C++

```
__fastcall int GetHTMLBlockCount();
```

Description

This is GetHTMLBlockCount, a member of class TScriptProducer.

69.1.22.5.7 TScriptProducer.GetWebModuleContext Method

Pascal

```
function GetWebModuleContext: TWebModuleContext;
```

C++

```
__fastcall TWebModuleContext GetWebModuleContext();
```

Description

This is GetWebModuleContext, a member of class TScriptProducer.

69.1.22.5.8 TScriptProducer.HandleParserError Method

Pascal

```
procedure HandleParserError(E: Exception);
```

C++

```
__fastcall HandleParserError(Exception E);
```

Description

This is HandleParserError, a member of class TScriptProducer.

69.1.22.5.9 TScriptProducer.HandleScriptError Method

Pascal

```
function HandleScriptError(const ScriptError: IUnknown): HRESULT;
```

C++

```
__fastcall HRESULT HandleScriptError(const IUnknown ScriptError);
```

Description

This is HandleScriptError, a member of class TScriptProducer.

69.1.22.5.10 TScriptProducer.ImplHandleScriptError Method

Pascal

```
function ImplHandleScriptError(const ScriptError: IUnknown): HRESULT; virtual; abstract;
```

C++

```
virtual __fastcall HRESULT ImplHandleScriptError(const IUnknown ScriptError) = 0;
```

Description

This is ImplHandleScriptError, a member of class TScriptProducer.

69.1.22.5.11 TScriptProducer.ParseStream Method

Pascal

```
procedure ParseStream(Stream: TStream; Owned: Boolean);
```

C++

```
__fastcall ParseStream(TStream Stream, Boolean Owned);
```

Description

This is ParseStream, a member of class TScriptProducer.

69.1.22.5.12 TScriptProducer.ParseString Method

Pascal

```
procedure ParseString(const S: string);
```

C++

```
__fastcall ParseString(const UnicodeString S);
```

Description

This is ParseString, a member of class TScriptProducer.

69.1.22.5.13 TScriptProducer.ReplaceTags Method

Pascal

```
function ReplaceTags(const AValue: string): string;
```

C++

```
__fastcall UnicodeString ReplaceTags(const UnicodeString AValue);
```

Description

This is ReplaceTags, a member of class TScriptProducer.

69.1.22.5.14 TScriptProducer.SetContent Method

Pascal

```
procedure SetContent(const Value: string);
```

C++

```
__fastcall SetContent(const UnicodeString Value);
```

Description

This is SetContent, a member of class TScriptProducer.

69.1.22.5.15 Write Method

This is the overview for the Write method overload.

Overload List

	Name	Description
	TScriptProducer.Write (PAnsiChar) (see page 6680)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (PWideChar, Integer) (see page 6680)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (string) (see page 6680)	This is Write, a member of class TScriptProducer.

69.1.22.5.15.1 TScriptProducer.Write Method (PAnsiChar)

Pascal

```
procedure Write(Value: PAnsiChar); overload;
```

C++

```
__fastcall Write(PAnsiChar Value);
```

Description

This is Write, a member of class TScriptProducer.

69.1.22.5.15.2 TScriptProducer.Write Method (PWideChar, Integer)

Pascal

```
procedure Write(const AValue: PWideChar; ALength: Integer); overload;
```

C++

```
__fastcall Write(const PWideChar AValue, int ALength);
```

Description

This is Write, a member of class TScriptProducer.

69.1.22.5.15.3 TScriptProducer.Write Method (string)

Pascal

```
procedure Write(const Value: string); overload;
```

C++

```
__fastcall Write(const UnicodeString Value);
```

Description

This is Write, a member of class TScriptProducer.

69.1.22.5.16 TScriptProducer.WriteItem Method

Pascal

```
procedure WriteItem(Index: Integer);
```

C++

```
__fastcall WriteItem(int Index);
```

Description

This is WriteItem, a member of class TScriptProducer.

69.1.22.6 TScriptProducer Properties

The properties of the TScriptProducer class are listed here.

Protected Properties

	Name	Description
	Component (see page 6681)	This is Component, a member of class TScriptProducer.
	Content (see page 6681)	This is Content, a member of class TScriptProducer.

	Designing (see page 6681)	This is Designing, a member of class TScriptProducer.
	Errors (see page 6681)	This is Errors, a member of class TScriptProducer.
	Script (see page 6682)	This is Script, a member of class TScriptProducer.

69.1.22.6.1 TScriptProducer.Component Property

Pascal

```
property Component: TComponent;
```

C++

```
__property TComponent * Component;
```

Description

This is Component, a member of class TScriptProducer.

69.1.22.6.2 TScriptProducer.Content Property

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

This is Content, a member of class TScriptProducer.

69.1.22.6.3 TScriptProducer.Designing Property

Pascal

```
property Designing: Boolean;
```

C++

```
__property Boolean Designing;
```

Description

This is Designing, a member of class TScriptProducer.

69.1.22.6.4 TScriptProducer.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class TScriptProducer.

69.1.22.6.5 TScriptProducer.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptProducer.

69.1.23 TVariableLookup Class

Class Hierarchy

Pascal

```
TVariableLookup = class(TObject);
```

C++

```
class TVariableLookup : public TObject;
```

File

WebScript

Description

This is class WebScript.TVariableLookup.

69.1.23.1 TVariableLookup Members

The following tables list the members exposed by TVariableLookup.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6683)	This is Create, a member of class TVariableLookup.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.

	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindGlobalVariable (see page 6684)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 6684)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	LookupObject (see page 6684)	This is LookupObject, a member of class TVariableLookup.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Ident (see page 6684)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 6685)	This is Obj, a member of class TVariableLookup.

69.1.23.2 TVariableLookup.Create Constructor

Pascal

```
constructor Create(const AIdent: string; AWebModuleContext: TWebModuleContext);
```

C++

```
__fastcall TVariableLookup(const UnicodeString AIdent, TWebModuleContext AWebModuleContext);
```

Description

This is Create, a member of class TVariableLookup.

69.1.23.3 TVariableLookup Methods

The methods of the TVariableLookup class are listed here.

Public Methods

	Name	Description
	FindGlobalVariable (see page 6684)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 6684)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	LookupObject (see page 6684)	This is LookupObject, a member of class TVariableLookup.

69.1.23.3.1 TVariableLookup.FindGlobalVariable Method

Pascal

```
function FindGlobalVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariable(const UnicodeString AName);
```

Description

This is FindGlobalVariable, a member of class TVariableLookup.

69.1.23.3.2 TVariableLookup.FindGlobalVariableContainer Method

Pascal

```
function FindGlobalVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariableContainer(const UnicodeString AName);
```

Description

This is FindGlobalVariableContainer, a member of class TVariableLookup.

69.1.23.3.3 TVariableLookup.LookupObject Method

Pascal

```
function LookupObject: Boolean;
```

C++

```
__fastcall Boolean LookupObject();
```

Description

This is LookupObject, a member of class TVariableLookup.

69.1.23.4 TVariableLookup Properties

The properties of the TVariableLookup class are listed here.

Public Properties

	Name	Description
	Ident (see page 6684)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 6685)	This is Obj, a member of class TVariableLookup.

69.1.23.4.1 TVariableLookup.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property UnicodeString Ident;
```

Description

This is Ident, a member of class TVariableLookup.

69.1.23.4.2 TVariableLookup.Obj Property

Pascal

```
property Obj: TObject;
```

C++

```
__property TObject * Obj;
```

Description

This is Obj, a member of class TVariableLookup.

69.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TGlobalScriptVariables (see page 6686)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 6687)	This is record WebScript.TIncludeType.

Records

	Name	Description
	TLineNumbers (see page 6687)	This is record WebScript.TLineNumbers.

69.2.1 WebScript.TGlobalScriptVariables Enumeration

Pascal

```
TGlobalScriptVariables = (
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
);
```

C++

```
enum TGlobalScriptVariables {
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
};
```

File

WebScript

Description

This is record WebScript.TGlobalScriptVariables.

69.2.2 WebScript.TIncludeType Enumeration

Pascal

```
TIncludeType = (
  incFile,
  incPage
);
```

C++

```
enum TIncludeType {
  incFile,
  incPage
};
```

File

WebScript

Description

This is record WebScript.TIncludeType.

69.2.3 WebScript.TLineNumbers Record

Pascal

```
TLineNumbers = record
  SourceStartLine: Integer;
  SourceLineCount: Integer;
  DestStartLine: Integer;
  DestLineCount: Integer;
  ScriptIncludeFile: TScriptIncludeFile;
end;
```

C++

```
struct TLineNumbers {
  int SourceStartLine;
  int SourceLineCount;
  int DestStartLine;
  int DestLineCount;
  TScriptIncludeFile ScriptIncludeFile;
};
```

File

WebScript

Description

This is record WebScript.TLineNumbers.

69.3 Types

The following table lists types in this documentation.

Types

Name	Description
CreateComponentStubFactory (see page 6688)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 6688)	This is type WebScript.PLineNumbers.

69.3.1 WebScript.CreateComponentStubFactory Type

Pascal

```
CreateComponentStubFactory = function (AComponent: TObject; AComponentClass: TClass):
IInterface;
```

C++

```
(AComponent: TObject; AComponentClass: TClass): IInterface ( CreateComponentStubFactory)();
```

File

WebScript

Description

This is type WebScript.CreateComponentStubFactory.

69.3.2 WebScript.PLineNumbers Type

Pascal

```
PLineNumbers = ^TLineNumbers;
```

C++

```
^TLineNumbers PLineNumbers;
```

File

WebScript

Description

This is type WebScript.PLineNumbers.

69.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
GlobalScriptVariableNames (see page 6689)	This is constant WebScript.GlobalScriptVariableNames.

69.4.1 WebScript.GlobalScriptVariableNames Constant

Pascal

```
GlobalScriptVariableNames: array[TGlobalScriptVariables] of PChar = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

C++

```
array[TGlobalScriptVariables] of PChar GlobalScriptVariableNames = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

File

WebScript

Description

This is constant WebScript.GlobalScriptVariableNames.

70 WebServExp Namespace

This is namespace WebServExp.

Topics

Name	Description
Classes (see page 6693)	The following table lists classes in this documentation.
Interfaces (see page 6701)	The following table lists interfaces in this documentation.
Functions (see page 6704)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6706)	The following table lists structs, records, enums in this documentation.
Types (see page 6708)	The following table lists types in this documentation.

Classes

	Name	Description
	TWebServExp (see page 6693)	This is class WebServExp.TWebServExp.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 6704)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 6704)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 6705)	This is function WebServExp.IsBaseClassTypeInfo.

Interfaces

	Name	Description
	IWebServExp (see page 6701)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 6703)	This is class WebServExp.IWebServExpAccess.

Structs, Records, Enums

	Name	Description
	ArgumentType (see page 6706)	This is record WebServExp.ArgumentType.
	MessageType (see page 6706)	This is record WebServExp.MessageType.
	TSchemaType (see page 6707)	This is record WebServExp.TSchemaType.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 6708)	This is type WebServExp.TAfterPublishingWSDLEvent.

TBeforePublishingTypesEvent (see page 6708)	This is type WebServExp.TBeforePublishingTypesEvent.
TPublishingTypeEvent (see page 6709)	This is type WebServExp.TPublishingTypeEvent.
TSchemaTypeArray (see page 6709)	This is type WebServExp.TSchemaTypeArray.

70.1 Classes

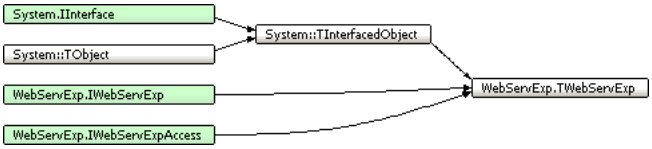
The following table lists classes in this documentation.

Classes

	Name	Description
	TWebServExp (see page 6693)	This is class WebServExp.TWebServExp.

70.1.1 TWebServExp Class

Class Hierarchy



Pascal

```
TWebServExp = class(TInterfacedObject, IWebServExp, IWebServExpAccess);
```

C++

```
class TWebServExp : public TInterfacedObject, public IWebServExp, public IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.TWebServExp.

70.1.1.1 TWebServExp Members

The following tables list the members exposed by TWebServExp.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6695)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6695)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindOrAddSchema (see page 6702)	Ensures that a schema is eventually generated for a specified type.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GenerateXMLSchema (see page 6696)	Adds schema information to represent a specified type.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetWebServExp (see page 6697)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 6697)	This is GetWSDLForInterface, a member of class TWebServExp.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	ImportLocations (see page 6698)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 6698)	This is ImportNames, a member of class TWebServExp.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	TargetNameSpace (see page 6698)	This is TargetNameSpace, a member of class TWebServExp.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 6699)	This is OnAfterPublishingWSDL, a member of class TWebServExp.
	OnBeforePublishingTypes (see page 6699)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 6699)	This is OnPublishingType, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 6697)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 6698)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 6698)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 6699)	This is WSDLElements, a member of class TWebServExp.

70.1.1.2 TWebServExp.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebServExp();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

70.1.1.3 TWebServExp.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebServExp();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and

call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

70.1.1.4 TWebServExp Methods

The methods of the TWebServExp class are listed here.

Public Methods

	Name	Description
	GenerateXMLSchema (see page 6696)	Adds schema information to represent a specified type.
	GetWebServExp (see page 6697)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 6697)	This is GetWSDLForInterface, a member of class TWebServExp.

70.1.1.4.1 TWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: InvString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, InvString Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 6702)

70.1.1.4.2 TWebServExp.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class TWebServExp.

70.1.1.4.3 TWebServExp.GetWSDLForInterface Method

Pascal

```
procedure GetWSDLForInterface(const IntfTypeInfo: Pointer; WSDLDoc: IWSDLDocument; PortNames:
array of InvString; Locations: array of InvString);
```

C++

```
__fastcall GetWSDLForInterface(const void * IntfTypeInfo, IWSDLDocument WSDLDoc, array of
InvString PortNames, array of InvString Locations);
```

Description

This is GetWSDLForInterface, a member of class TWebServExp.

70.1.1.5 TWebServExp Properties

The properties of the TWebServExp class are listed here.

Public Properties

	Name	Description
	ImportLocations (see page 6698)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 6698)	This is ImportNames, a member of class TWebServExp.
	TargetNameSpace (see page 6698)	This is TargetNameSpace, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 6697)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 6698)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 6698)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 6699)	This is WSDLElements, a member of class TWebServExp.

70.1.1.5.1 TWebServExp.ArrayAsComplexContent Property

Pascal

```
property ArrayAsComplexContent: Boolean;
```

C++

```
__property Boolean ArrayAsComplexContent;
```

Description

This is ArrayAsComplexContent, a member of class TWebServExp.

70.1.1.5.2 TWebServExp.BindingType Property

Pascal

```
property BindingType: TWebServiceBindingType;
```

C++

```
__property TWebServiceBindingType BindingType;
```

Description

This is BindingType, a member of class TWebServExp.

70.1.1.5.3 TWebServExp.ImportLocations Property

Pascal

```
property ImportLocations [const Index: Integer]: InvString;
```

C++

```
__property InvString ImportLocations[int const Index];
```

Description

This is ImportLocations, a member of class TWebServExp.

70.1.1.5.4 TWebServExp.ImportNames Property

Pascal

```
property ImportNames [const Index: Integer]: InvString;
```

C++

```
__property InvString ImportNames[int const Index];
```

Description

This is ImportNames, a member of class TWebServExp.

70.1.1.5.5 TWebServExp.ServiceAddress Property

Pascal

```
property ServiceAddress: InvString;
```

C++

```
__property InvString ServiceAddress;
```

Description

This is ServiceAddress, a member of class TWebServExp.

70.1.1.5.6 TWebServExp.TargetNameSpace Property

Pascal

```
property TargetNameSpace: InvString;
```

C++

```
__property InvString TargetNameSpace;
```

Description

This is TargetNameSpace, a member of class TWebServExp.

70.1.1.5.7 TWebServExp.WSDLElements Property

Pascal

```
property WSDLElements: TWSDLElements;
```

C++

```
__property TWSDLElements WSDLElements;
```

Description

This is WSDLElements, a member of class TWebServExp.

70.1.1.6 TWebServExp Events

The events of the TWebServExp class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 6699)	This is OnAfterPublishingWSDL, a member of class TWebServExp.
	OnBeforePublishingTypes (see page 6699)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 6699)	This is OnPublishingType, a member of class TWebServExp.

70.1.1.6.1 TWebServExp.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWebServExp.

70.1.1.6.2 TWebServExp.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWebServExp.

70.1.1.6.3 TWebServExp.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWebServExp.

70.2 Interfaces

The following table lists interfaces in this documentation.

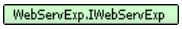
Interfaces

	Name	Description
	IWebServExp (see page 6701)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 6703)	This is class WebServExp.IWebServExpAccess.

70.2.1 IWebServExp Interface

IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.

Class Hierarchy



Pascal

```
IWebServExp = interface;
```

C++

```
__interface IWebServExp;
```

File

WebServExp

Description

IWebServExp is an interface for generating the schema definitions that represent types in a WSDL document.

Note: C++ method declarations that use IWebServExp use the _di_IWebServExp type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWebServExp interface:

Note: typedef System::DelphiInterface (see page 3263)< IWebServExp > _di_IWebServExp;

See Also

TWSDLHTMLPublish (see page 7232)

70.2.1.1 IWebServExp Members

The following tables list the members exposed by IWebServExp.

Public Methods

	Name	Description
	FindOrAddSchema (see page 6702)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 6702)	Adds schema information to represent a specified type.

70.2.1.2 IWebServExp Methods

The methods of the IWebServExp class are listed here.

Public Methods

	Name	Description
	FindOrAddSchema (see page 6702)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 6702)	Adds schema information to represent a specified type.

70.2.1.2.1 IWebServExp.FindOrAddSchema Method

Ensures that a schema is eventually generated for a specified type.

Pascal

```
function FindOrAddSchema(const ATypeInfo: PTypeinfo; const TnsURI: string): Boolean;
```

C++

```
__fastcall Boolean FindOrAddSchema(const PTypeinfo ATypeInfo, const UnicodeString TnsURI);
```

Description

Call FindOrAddSchema to indicate that a schema should be generated for a given type.

ATypeInfo is the runtime type information (RTTI) for the type.

TnsURI is the namespace URI in which the schema for the type should be defined.

FindOrAddSchema returns true if there is already a schema for the specified type. It returns false if the type has not previously been added.

See Also

GenerateXMLSchema (see page 6702)

70.2.1.2.2 IWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: InvString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, InvString Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

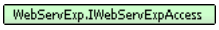
Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 6702)

70.2.2 IWebServExpAccess Interface

Class Hierarchy



Pascal

```
IWebServExpAccess = interface;
```

C++

```
__interface IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.IWebServExpAccess.

70.2.2.1 IWebServExpAccess Members

The following tables list the members exposed by IWebServExpAccess.

Public Methods

	Name	Description
	GetWebServExp (see page 6703)	This is GetWebServExp, a member of class IWebServExpAccess.

70.2.2.2 IWebServExpAccess Methods

The methods of the IWebServExpAccess class are listed here.

Public Methods

	Name	Description
	GetWebServExp (see page 6703)	This is GetWebServExp, a member of class IWebServExpAccess.

70.2.2.2.1 IWebServExpAccess.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class IWebServExpAccess.

70.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 6704)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 6704)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 6705)	This is function WebServExp.IsBaseClassTypeInfo.

70.3.1 WebServExp.GetAliasBaseTypeInfo Function

Pascal

```
function GetAliasBaseTypeInfo(const ParamType: TTypeKind): PTypeInfo;
```

C++

```
PTypeInfo GetAliasBaseTypeInfo(const TTypeKind ParamType);
```

File

WebServExp

Description

This is function WebServExp.GetAliasBaseTypeInfo.

70.3.2 WebServExp.GetBindingType Function

Pascal

```
function GetBindingType(const MethEntry: TIntfMethEntry; Input: Boolean):
TWebServiceBindingType; overload;
```

C++

```
TWebServiceBindingType GetBindingType(const TIntfMethEntry MethEntry, Boolean Input);
```

File

WebServExp

Description

This is function WebServExp.GetBindingType.

70.3.3 WebServExp.IsBaseClassTypeInfo Function

Pascal

```
function IsBaseClassTypeInfo(const ATypeInfo: PTypeInfo): Boolean;
```

C++

```
Boolean IsBaseClassTypeInfo(const PTypeInfo ATypeInfo);
```

File

WebServExp

Description

This is function WebServExp.IsBaseClassTypeInfo.

70.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	ArgumentType (see page 6706)	This is record WebServExp.ArgumentType.
	MessageType (see page 6706)	This is record WebServExp.MessageType.

Records

	Name	Description
	TSchemaType (see page 6707)	This is record WebServExp.TSchemaType.

70.4.1 WebServExp.ArgumentType Enumeration

Pascal

```
ArgumentType = (
  argIn,
  argOut,
  argInOut,
  argReturn
);
```

C++

```
enum ArgumentType {
  argIn,
  argOut,
  argInOut,
  argReturn
};
```

File

WebServExp

Description

This is record WebServExp.ArgumentType.

70.4.2 WebServExp.MessageType Enumeration

Pascal

```
MessageType = (
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
);
```

C++

```
enum MessageType {
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
};
```

File

WebServExp

Description

This is record WebServExp.MessageType.

70.4.3 WebServExp.TSchemaType Record

Pascal

```
TSchemaType = record
  TypeName: InvString;
  NameSpace: InvString;
  TypeInfo: PTypeinfo;
  NSPrefix: InvString;
  XSGenerated: Boolean;
end;
```

C++

```
struct TSchemaType {
  InvString TypeName;
  InvString NameSpace;
  PTypeinfo TypeInfo;
  InvString NSPrefix;
  Boolean XSGenerated;
};
```

File

WebServExp

Description

This is record WebServExp.TSchemaType.

70.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 6708)	This is type WebServExp.TAfterPublishingWSDLEvent.
TBeforePublishingTypesEvent (see page 6708)	This is type WebServExp.TBeforePublishingTypesEvent.
TPublishingTypeEvent (see page 6709)	This is type WebServExp.TPublishingTypeEvent.
TSchemaTypeArray (see page 6709)	This is type WebServExp.TSchemaTypeArray.

70.5.1 WebServExp.TAfterPublishingWSDLEvent Type

Pascal

```
TAfterPublishingWSDLEvent = procedure (const WSDLDoc: IWSDLDocument) of object;
```

C++

```
(const WSDLDoc: IWSDLDocument) ( TAfterPublishingWSDLEvent)();
```

File

WebServExp

Description

This is type WebServExp.TAfterPublishingWSDLEvent.

70.5.2 WebServExp.TBeforePublishingTypesEvent Type

Pascal

```
TBeforePublishingTypesEvent = procedure (const WebServ: IWebServExp) of object;
```

C++

```
(const WebServ: IWebServExp) ( TBeforePublishingTypesEvent)();
```

File

WebServExp

Description

This is type WebServExp.TBeforePublishingTypesEvent.

70.5.3 WebServExp.TPublishingTypeEvent Type

Pascal

```
TPublishingTypeEvent = procedure (const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef;
const ATypeInfo: PTypeinfo; Namespace: InvString) of object;
```

C++

```
(const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo;
Namespace: InvString) ( TPublishingTypeEvent)();
```

File

WebServExp

Description

This is type WebServExp.TPublishingTypeEvent.

70.5.4 WebServExp.TSchemaTypeArray Type

Pascal

```
TSchemaTypeArray = array of TSchemaType;
```

C++

```
array of TSchemaType TSchemaTypeArray;
```

File

WebServExp

Description

This is type WebServExp.TSchemaTypeArray.

71 WebSess Namespace

This is namespace WebSess.

Topics

Name	Description
Classes (see page 6712)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6739)	The following table lists structs, records, enums in this documentation.
Types (see page 6740)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomSessionsService (see page 6712)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 6722)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 6726)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 6729)	This is class WebSess.TWebSessionImpl.

Structs, Records, Enums

	Name	Description
	TEndSessionReason (see page 6739)	This is record WebSess.TEndSessionReason.

Types

Name	Description
TEndSessionEvent (see page 6740)	This is type WebSess.TEndSessionEvent.
TStartSessionEvent (see page 6740)	This is type WebSess.TStartSessionEvent.

71.1 Classes

The following table lists classes in this documentation.

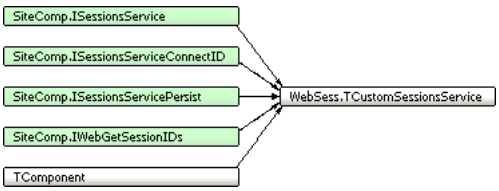
Classes

	Name	Description
	TCustomSessionsService (see page 6712)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 6722)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 6726)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 6729)	This is class WebSess.TWebSessionImpl.

71.1.1 TCustomSessionsService Class

TCustomSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy



Pascal

```
TCustomSessionsService = class(TComponent, ISessionsService, ISessionsServiceConnectID,
IWebGetSessionIDs, ISessionsServicePersist);
```

C++

```
class TCustomSessionsService : public TComponent, public ISessionsService, public
ISessionsServiceConnectID, public IWebGetSessionIDs, public ISessionsServicePersist;
```

File

WebSess

Description

TCustomSessionsService is the base class for classes that manage the sessions in a WebSnap application. When a WebSnap application contains a TCustomSessionsService descendant, then each HTTP request is associated with a session. TCustomSessionsService can manage multiple sessions on behalf of different users, each on a separate thread of execution. Only one session is available when handling a request. This is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TCustomSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Thus, Web application types that do not remain in memory between requests, such as CGI, do not support TCustomSessionsService.

TCustomSessionsService implements the ISessionsService (see page 1007) interface.

See Also

TSessionsService (see page 6722)

TEndUserSessionAdapter (see page 5779)

TAbstractWebContext (see page 5935)

TWebSession (see page 6225)

WebContext

ISessionsService (see page 1007)

71.1.1.1 TCustomSessionsService Members

The following tables list the members exposed by TCustomSessionsService.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6714)	Checks whether any sessions have started or terminated.
	Clear (see page 6715)	Deletes all sessions from memory.
	Connect (see page 6715)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6716)	Obtains the interface for an active Web session, given its Session ID.
	DoEndSession (see page 6716)	Triggers the OnEndSession (see page 6721) event when appropriate.
	DoStartSession (see page 6716)	Triggers the OnStartSession (see page 6721) event when appropriate.
	GetSessionIDs (see page 6717)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6717)	Loads the list of sessions from a file.
	LoadFromStream (see page 6718)	Loads the list of sessions from a stream.
	NewSession (see page 6718)	Starts a new session.
	NotifyStatusChange (see page 6719)	Generates an OnEndSession (see page 6721) or OnStartSession (see page 6721) event, as approrpriate.
	SaveToFile (see page 6719)	Saves the list of sessions to a file.
	SaveToStream (see page 6719)	Saves the list of sessions to a stream.

Public Events

	Name	Description
	OnEndSession (see page 6721)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6721)	Occurs when the sessions service detects that a new session has started.

Public Methods

	Name	Description
	Create (see page 6713)	Creates and initializes an instance of TCustomSessionsService (see page 6712).

Public Properties

	Name	Description
	DefaultTimeout (see page 6720)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6720)	Indicates the maximum number of sessions the TCustomSessionsService (see page 6712) can keep in memory.

71.1.1.2 TCustomSessionsService.Create Constructor

Creates and initializes an instance of TCustomSessionsService (see page 6712).

Pascal

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

C++

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

Description

Call Create to instantiate a sessions service at runtime. Typically, this is unnecessary, because a WebSnap application needs at most a single sessions service component, and in most cases that is added at design time.

AOwner is the component that becomes the value of the TCustomSessionsService (see page 6712) object's Owner property. This should be the Web application module.

See Also

Owner

71.1.1.3 TCustomSessionsService Methods

The methods of the TCustomSessionsService class are listed here.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6714)	Checks whether any sessions have started or terminated.
	Clear (see page 6715)	Deletes all sessions from memory.
	Connect (see page 6715)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6716)	Obtains the interface for an active Web session, given its Session ID.
V	DoEndSession (see page 6716)	Triggers the OnEndSession (see page 6721) event when appropriate.
V	DoStartSession (see page 6716)	Triggers the OnStartSession (see page 6721) event when appropriate.
	GetSessionIDs (see page 6717)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6717)	Loads the list of sessions from a file.
	LoadFromStream (see page 6718)	Loads the list of sessions from a stream.
	NewSession (see page 6718)	Starts a new session.
	NotifyStatusChange (see page 6719)	Generates an OnEndSession (see page 6721) or OnStartSession (see page 6721) event, as approrpriate.
	SaveToFile (see page 6719)	Saves the list of sessions to a file.
	SaveToStream (see page 6719)	Saves the list of sessions to a stream.

71.1.1.3.1 TCustomSessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange is called automatically when the application creates a new Web context or initializes a new Web module. That is, this method is called at least once every time the application receives a new HTTP request message.

If CheckStatusChange detects that a session has started or terminated since the last call, it returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 6719)

TWebContext (see page 5950)

71.1.1.3.2 TCustomSessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory.

Clear does not generate an OnEndSession (see page 6721) event for the sessions it deletes when it frees their memory, it simply removes them from memory. To shut down all sessions in a more orderly fashion, iterate through all the active sessions by calling ConnectActive (see page 6716) for every session ID that the GetSessionIDs (see page 6717) method returns, and terminate each one. Then call NotifyStatusChange (see page 6719) to generate the OnEndSession (see page 6721) events for every terminated session. At that point, calling Clear does not prevent the application from receiving an OnEndSession (see page 6721) event for any session that the sessions service manages.

See Also

NewSession (see page 6718)

GetSessionIDs (see page 6717)

ConnectActive (see page 6716)

NotifyStatusChange (see page 6719)

71.1.1.3.3 TCustomSessionsService.Connect Method

Obtains the interface for a Web session, given its Session ID.

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

Call Connect to access the session that has a specified Session ID. Unlike the ConnectActive (see page 6716) method, Connect may return a session that has expired or been terminated but that has not yet been freed from memory.

AID is the session ID of the session to locate.

Connect returns the IWebSession (see page 1043) interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have a session with the specified session ID, Connect returns nil (Delphi) or NULL (C++).

Note: Calling Connect does not count as accessing a session. That is, if you call Connect for any session, it does not affect the time when that session times out.

See Also

ConnectActive (see page 6716)

GetSessionIDs (see page 6717)

71.1.1.3.4 TCustomSessionsService.ConnectActive Method

Obtains the interface for an active Web session, given its Session ID.

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

Call ConnectActive to access the session that has a specified Session ID. Unlike the Connect (see page 6715) method, ConnectActive never returns a session that has expired or been terminated.

AID is the session ID of the session to locate.

ConnectActive returns the IWebSession (see page 1043) interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have an active session with the specified session ID, ConnectActive returns nil (Delphi) or NULL (C++).

Note: Calling ConnectActive counts as accessing a session. That is, if you call ConnectActive for any session, the session will not time out until the entire time-out period elapses after the call to ConnectActive.

See Also

Connect (see page 6715)

GetSessionIDs (see page 6717)

71.1.1.3.5 TCustomSessionsService.DoEndSession Method

Triggers the OnEndSession (see page 6721) event when appropriate.

Pascal

```
procedure DoEndSession(ASession: TAbstractWebSession; AReason: TEndSessionReason); virtual;
```

C++

```
virtual __fastcall DoEndSession(TAbstractWebSession ASession, TEndSessionReason AReason);
```

Description

Use DoEndSession to trigger an OnEndSession (see page 6721) event, if it is appropriate to do so.

See Also

OnEndSession (see page 6721)

71.1.1.3.6 TCustomSessionsService.DoStartSession Method

Triggers the OnStartSession (see page 6721) event when appropriate.

Pascal

```
procedure DoStartSession(ASession: TAbstractWebSession); virtual;
```

C++

```
virtual __fastcall DoStartSession(TAbstractWebSession ASession);
```

Description

Use DoStartSession to trigger an OnStartSession (see page 6721) event, if it is appropriate to do so.

See Also

OnStartSession (see page 6721)

71.1.1.3.7 TCustomSessionsService.GetSessionIDs Method

Returns a list of the session IDs for all sessions currently in memory.

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

Description

Call GetSessionIDs to obtain a list of the session IDs for all the sessions that the sessions service is storing. GetSessionIDs returns an IWebSessionIDs (see page 1051) interface, which lists the session IDs.

See Also

Connect (see page 6715)

GetSessionIDs

71.1.1.3.8 TCustomSessionsService.LoadFromFile Method

Loads the list of sessions from a file.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

Call LoadFromFile to replace the current list of sessions with a set of sessions stored in a file. The file must be previously created by a call to the SaveToFile (see page 6719) method.

Filename is the name of the file that contains the saved list of sessions.

See Also

SaveToFile (see page 6719)

LoadFromStream (see page 6718)

71.1.1.3.9 TCustomSessionsService.LoadFromStream Method

Loads the list of sessions from a stream.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

Call LoadFromStream to replace the current list of sessions with a set of sessions read from a stream. The sessions must be previously saved by a call to the SaveToStream (see page 6719) method.

S is the stream from which to read the list of sessions.

See Also

SaveToStream (see page 6719)

LoadFromFile (see page 6717)

71.1.1.3.10 TCustomSessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

If the sessions service already has MaxSessions (see page 6720) sessions, NewSession raises an ESessionsException (see page 328) exception.

Note: The OnStartSession (see page 6721) event does not occur immediately when you call NewSession to start a new session. It occurs on the next call to NotifyStatusChange (see page 6719).

Tip: To reconnect to an existing session, call the ConnectActive (see page 6716) method instead.

See Also

MaxSessions (see page 6720)

OnStartSession (see page 6721)

ConnectActive (see page 6716)

NotifyStatusChange (see page 6719)

71.1.1.3.11 TCustomSessionsService.NotifyStatusChange Method

Generates an OnEndSession (see page 6721) or OnStartSession (see page 6721) event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession (see page 6721) event for any new sessions that have been created since the last call and to generate the OnEndSession (see page 6721) event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession (see page 6721) event handler exits.

NotifyStatusChange is called automatically every time the WebSnap application initializes a new Web module.

See Also

OnEndSession (see page 6721)

OnStartSession (see page 6721)

CheckStatusChange (see page 6714)

71.1.1.3.12 TCustomSessionsService.SaveToFile Method

Saves the list of sessions to a file.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

Call SaveToFile to save the current list of sessions to a file. SaveToFile saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The file can be read back into a sessions service by calling the LoadFromFile (see page 6717) method.

Filename is the name of the file to which the list of sessions is saved.

See Also

SaveToStream (see page 6719)

LoadFromFile (see page 6717)

71.1.1.3.13 TCustomSessionsService.SaveToStream Method

Saves the list of sessions to a stream.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

Call SaveToStream to write the current list of sessions to a stream. SaveToStream saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The sessions can be read back into a sessions service by calling the LoadFromStream (see page 6718) method.

S is the stream to which the list of sessions is written.

See Also

LoadFromStream (see page 6718)

SaveToFile (see page 6719)

71.1.1.4 TCustomSessionsService Properties

The properties of the TCustomSessionsService class are listed here.

Public Properties

	Name	Description
	DefaultTimeout (see page 6720)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6720)	Indicates the maximum number of sessions the TCustomSessionsService (see page 6712) can keep in memory.

71.1.1.4.1 TCustomSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

71.1.1.4.2 TCustomSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService (see page 6712) can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException (see page 328) exception. The Exception (see page 4015) occurs before the OnStartSession (see page 6721) event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 328)

OnStartSession (see page 6721)

Terminate (see page 6229)

71.1.1.5 TCustomSessionsService Events

The events of the TCustomSessionsService class are listed here.

Public Events

	Name	Description
	OnEndSession (see page 6721)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6721)	Occurs when the sessions service detects that a new session has started.

71.1.1.5.1 TCustomSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write (see page 3789) an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 6721)

Terminate (see page 6229)

TimeoutMinutes

71.1.1.5.2 TCustomSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

```
__property TStartSessionEvent OnStartSession;
```

Description

Write (see page 3789) an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

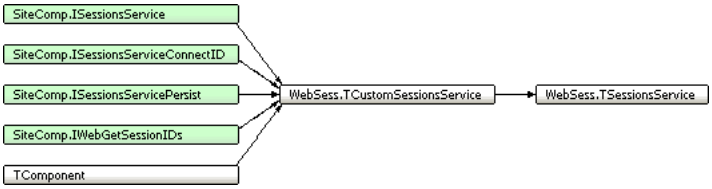
See Also

OnStartSession

71.1.2 TSessionsService Class

TSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy



Pascal

```
TSessionsService = class(TCustomSessionsService);
```

C++

```
class TSessionsService : public TCustomSessionsService;
```

File

WebSess

Description

Add TSessionsService to a WebSnap application when you want to store temporary end user data in a WebSnap session. For example, TSessionsService can keep track of all users that are currently logged in and automatically log a user out after a period of inactivity.

When a WebSnap application contains a TSessionsService component, each HTTP request is associated with a session. The WebSnap application can manage multiple sessions on behalf of different users, each on a separate thread of execution. When handling a request, only one session is available, which is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Web application types that do not remain in memory between requests, such as CGI, do not support TSessionsService. When testing Web App Debugger executables, be sure to start the executables before browsing so that the executable remains running between requests.

Use TEndUserSessionsAdapter and TSessionsAdapter together to keep track of end user name and login state.

See Also

TEndUserSessionAdapter (see page 5779)

TAbstractWebContext (see page 5935)

TWebSession (see page 6225)

WebContext

Adding Login Support

Using the Sessions Service

71.1.2.1 TSessionsService Members

The following tables list the members exposed by TSessionsService.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6714)	Checks whether any sessions have started or terminated.
	Clear (see page 6715)	Deletes all sessions from memory.
	Connect (see page 6715)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6716)	Obtains the interface for an active Web session, given its Session ID.
	DoEndSession (see page 6716)	Triggers the OnEndSession (see page 6721) event when appropriate.
	DoStartSession (see page 6716)	Triggers the OnStartSession (see page 6721) event when appropriate.
	GetSessionIDs (see page 6717)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6717)	Loads the list of sessions from a file.
	LoadFromStream (see page 6718)	Loads the list of sessions from a stream.
	NewSession (see page 6718)	Starts a new session.
	NotifyStatusChange (see page 6719)	Generates an OnEndSession (see page 6721) or OnStartSession (see page 6721) event, as approrpriate.
	SaveToFile (see page 6719)	Saves the list of sessions to a file.
	SaveToStream (see page 6719)	Saves the list of sessions to a stream.

Public Methods

	Name	Description
	CheckStatusChange (see page 1008)	Checks whether any sessions have started or terminated.
	Clear (see page 1008)	Deletes all sessions from memory.
	Connect (see page 1010)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1011)	This is ConnectActive, a member of class ISessionsServiceConnectID.
	Create (see page 6713)	Creates and initializes an instance of TCustomSessionsService (see page 6712).
	GetSessionIDs (see page 1038)	This is GetSessionIDs, a member of class IWebGetSessionIDs.
	LoadFromFile (see page 1012)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1012)	This is LoadFromStream, a member of class ISessionsServicePersist.
	NewSession (see page 1009)	Starts a new session.
	NotifyStatusChange (see page 1009)	Generates an OnEndSession or OnStartSession event, as approrpriate.
	SaveToFile (see page 1012)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 1012)	This is SaveToStream, a member of class ISessionsServicePersist.

Published Events

	Name	Description
	OnEndSession (see page 6725)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6725)	Occurs when the sessions service detects that a new session has started.

Published Properties

	Name	Description
	DefaultTimeout (see page 6724)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6724)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

71.1.2.2 TSessionsService Properties

The properties of the TSessionsService class are listed here.

Published Properties

	Name	Description
	DefaultTimeout (see page 6724)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6724)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

71.1.2.2.1 TSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

71.1.2.2.2 TSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException exception. The Exception occurs before the OnStartSession event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by

MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 328)

OnStartSession (see page 6721)

Terminate (see page 6229)

71.1.2.3 TSessionsService Events

The events of the TSessionsService class are listed here.

Published Events

	Name	Description
	OnEndSession (see page 6725)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6725)	Occurs when the sessions service detects that a new session has started.

71.1.2.3.1 TSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 6721)

Terminate (see page 6229)

TimeoutMinutes

71.1.2.3.2 TSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

```
__property TStartSessionEvent OnStartSession;
```

Description

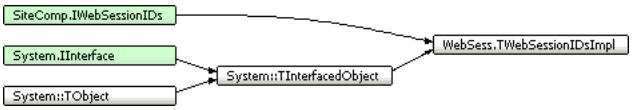
Write an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

See Also

OnStartSession (see page 6721)

71.1.3 TWebSessionIDsImpl Class

Class Hierarchy



Pascal

```
TWebSessionIDsImpl = class(TInterfacedObject, IWebSessionIDs);
```

C++

```
class TWebSessionIDsImpl : public TInterfacedObject, public IWebSessionIDs;
```

File

WebSess

Description

This is class WebSess.TWebSessionIDsImpl.

71.1.3.1 TWebSessionIDsImpl Members

The following tables list the members exposed by TWebSessionIDsImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetID (see page 6728)	Returns the value of the IDs property.
	GetIDCount (see page 6729)	Returns the value of the IDCount property.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.

	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6727)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6728)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	IDCount (see page 1053)	Indicates the number of session IDs (see page 1053) available.
R	IDs (see page 1053)	Lists the session IDs for all the sessions stored by the sessions service.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

71.1.3.2 TWebSessionIDsImpl.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebSessionIDsImpl();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

71.1.3.3 TWebSessionIDsImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionIDsImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

71.1.3.4 TWebSessionIDsImpl Methods

The methods of the TWebSessionIDsImpl class are listed here.

Protected Methods

	Name	Description
	GetID (see page 6728)	Returns the value of the IDs property.
	GetIDCount (see page 6729)	Returns the value of the IDCount property.

71.1.3.4.1 TWebSessionIDsImpl.GetID Method

Returns the value of the IDs property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs property.

I is the index of the session ID to return.

See Also

IDs (see page 1053)

71.1.3.4.2 TWebSessionIDsImpl.GetIDCount Method

Returns the value of the IDCount property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

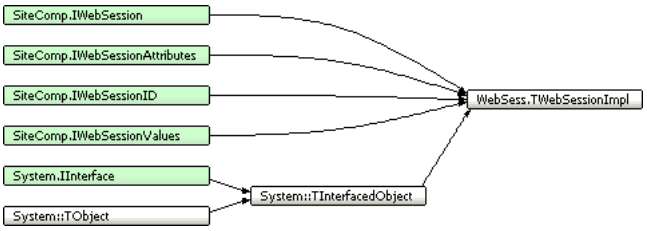
GetIDCount is the read implementation of the IDCount property.

See Also

IDCount (see page 1053)

71.1.4 TWebSessionImpl Class

Class Hierarchy



Pascal

```
TWebSessionImpl = class(TInterfacedObject, IWebSession, IWebSessionID, IWebSessionValues,
IWebSessionAttributes);
```

C++

```
class TWebSessionImpl : public TInterfacedObject, public IWebSession, public IWebSessionID,
public IWebSessionValues, public IWebSessionAttributes;
```

File

WebSess

Description

This is class WebSess.TWebSessionImpl.

71.1.4.1 TWebSessionImpl Members

The following tables list the members exposed by TWebSessionImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	GetExpirationTime (see page 6732)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 6733)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 6733)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 6733)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 6733)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 6734)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 6734)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 6734)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 6734)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 6734)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 6735)	This is GetValue, a member of class TWebSessionImpl.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	SetTimeoutMinutes (see page 6735)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 6735)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 6735)	Terminates the session.

Protected Properties

	Name	Description
R	ExpirationTime (see page 6736)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 6736)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 6736)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 6737)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 6737)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 6737)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 6737)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 6738)	This is Values, a member of class TWebSessionImpl.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6731)	This is Create, a member of class TWebSessionImpl.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 6731)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	IsActive (see page 1048)	This is IsActive, a member of class IWebSessionAttributes.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

71.1.4.2 TWebSessionImpl.Create Constructor

Pascal

```
constructor Create(AID: TSessionID);
```

C++

```
__fastcall TWebSessionImpl(TSessionID AID);
```

Description

This is Create, a member of class TWebSessionImpl.

71.1.4.3 TWebSessionImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

71.1.4.4 TWebSessionImpl Methods

The methods of the TWebSessionImpl class are listed here.

Protected Methods

	Name	Description
	GetExpirationTime (see page 6732)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 6733)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 6733)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 6733)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 6733)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 6734)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 6734)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 6734)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 6734)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 6734)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 6735)	This is GetValue, a member of class TWebSessionImpl.
	SetTimeoutMinutes (see page 6735)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 6735)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 6735)	Terminates the session.

71.1.4.4.1 TWebSessionImpl.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class TWebSessionImpl.

71.1.4.4.2 TWebSessionImpl.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class TWebSessionImpl.

71.1.4.4.3 TWebSessionImpl.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class TWebSessionImpl.

71.1.4.4.4 TWebSessionImpl.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class TWebSessionImpl.

71.1.4.4.5 TWebSessionImpl.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class TWebSessionImpl.

71.1.4.4.6 TWebSessionImpl.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class TWebSessionImpl.

71.1.4.4.7 TWebSessionImpl.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class TWebSessionImpl.

71.1.4.4.8 TWebSessionImpl.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class TWebSessionImpl.

71.1.4.4.9 TWebSessionImpl.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class TWebSessionImpl.

71.1.4.4.10 TWebSessionImpl.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class TWebSessionImpl.

71.1.4.4.11 TWebSessionImpl.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const UnicodeString AName);
```

Description

This is GetValue, a member of class TWebSessionImpl.

71.1.4.4.12 TWebSessionImpl.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class TWebSessionImpl.

71.1.4.4.13 TWebSessionImpl.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

This is SetValue, a member of class TWebSessionImpl.

71.1.4.4.14 TWebSessionImpl.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

See Also

TimeoutMinutes

Values

71.1.4.5 TWebSessionImpl Properties

The properties of the TWebSessionImpl class are listed here.

Protected Properties

	Name	Description
R	ExpirationTime (see page 6736)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 6736)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 6736)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 6737)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 6737)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 6737)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 6737)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 6738)	This is Values, a member of class TWebSessionImpl.

71.1.4.5.1 TWebSessionImpl.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class TWebSessionImpl.

71.1.4.5.2 TWebSessionImpl.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class TWebSessionImpl.

71.1.4.5.3 TWebSessionImpl.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class TWebSessionImpl.

71.1.4.5.4 TWebSessionImpl.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class TWebSessionImpl.

71.1.4.5.5 TWebSessionImpl.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class TWebSessionImpl.

71.1.4.5.6 TWebSessionImpl.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class TWebSessionImpl.

71.1.4.5.7 TWebSessionImpl.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class TWebSessionImpl.

71.1.4.5.8 TWebSessionImpl.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const AName];
```

Description

This is Values, a member of class TWebSessionImpl.

71.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEndSessionReason (see page 6739)	This is record WebSess.TEndSessionReason.

71.2.1 WebSess.TEndSessionReason Enumeration

Pascal

```
TEndSessionReason = (
  esTimeout,
  esTerminate
);
```

C++

```
enum TEndSessionReason {
  esTimeout,
  esTerminate
};
```

File

WebSess

Description

This is record WebSess.TEndSessionReason.

71.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEndSessionEvent (see page 6740)	This is type WebSess.TEndSessionEvent.
TStartSessionEvent (see page 6740)	This is type WebSess.TStartSessionEvent.

71.3.1 WebSess.TEndSessionEvent Type

Pascal

```
TEndSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession; AReason:
TEndSessionReason) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession; AReason: TEndSessionReason) (
TEndSessionEvent)();
```

File

WebSess

Description

This is type WebSess.TEndSessionEvent.

71.3.2 WebSess.TStartSessionEvent Type

Pascal

```
TStartSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession) ( TStartSessionEvent)();
```

File

WebSess

Description

This is type WebSess.TStartSessionEvent.

72 WebUsers Namespace

This is namespace WebUsers.

Topics

Name	Description
Classes (see page 6742)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6764)	The following table lists structs, records, enums in this documentation.
Types (see page 6765)	The following table lists types in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 6742)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 6742)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 6743)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 6759) raises.
	TCustomWebUserList (see page 6744)	The TCustomWebUserList component uses a TWebUserItems (see page 6755) object to store the account information about a collection of users.
	TWebUserItem (see page 6750)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 6755)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 6759)	TWebUserList validates login and checks access rights for a particular user.

Structs, Records, Enums

	Name	Description
	TValidateUserError (see page 6764)	This is record WebUsers.TValidateUserError.

Types

Name	Description
TCheckAccessRightsEvent (see page 6765)	This is type WebUsers.TCheckAccessRightsEvent.
TCheckAccessRightsHandledEvent (see page 6765)	This is type WebUsers.TCheckAccessRightsHandledEvent.
TValidateUserErrorEvent (see page 6766)	This is type WebUsers.TValidateUserErrorEvent.
TValidateUserEvent (see page 6766)	This is type WebUsers.TValidateUserEvent.
TValidateUserHandledEvent (see page 6766)	This is type WebUsers.TValidateUserHandledEvent.

72.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 6742)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 6742)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 6743)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 6759) raises.
	TCustomWebUserList (see page 6744)	The TCustomWebUserList component uses a TWebUserItems (see page 6755) object to store the account information about a collection of users.
	TWebUserItem (see page 6750)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 6755)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 6759)	TWebUserList validates login and checks access rights for a particular user.

72.1.1 EUserIDNotFoundException Class

EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.

Class Hierarchy

Pascal

```
EUserIDNotFoundException = class(EWebUsersListException);
```

C++

```
class EUserIDNotFoundException : public EWebUsersListException;
```

File

WebUsers

Description

When a Web user list (TWebUserList (see page 6759)) attempts to look up information about a user and fails to find an entry for that user, it generates an OnUserIDNotFound event. If that event handler does not handle the situation, then the Web user list raises an EUserIDNotFoundException exception.

See Also

TWebUserList (see page 6759)

TCustomWebUserList (see page 6744)

72.1.2 EValidateUserException Class

EValidateUserException is raised when a Web user list fails to validate a user.

Class Hierarchy

Pascal

```
EValidateUserException = class(EWebUsersListException);
```

C++

```
class EValidateUserException : public EWebUsersListException;
```

File

WebUsers

Description

When a WebSnap application that contains a Web user list (TWebUserList (see page 6759)) tries to log a user in, the Web user list raises an EValidateUserException if it can't validate the supplied user ID and password and its OnValidateUserError event handler does not handle the situation.

The Web user list may fail to validate a user ID and password for any of the following reasons:

The WebSnap application failed to supply a password or user ID when trying to validate the user.

The supplied user ID does not refer to a user in the list.

The supplied password is not correct for the specified user.

See Also

TWebUserList (see page 6759)

TCustomWebUserList (see page 6744)

TCustomWebUserList (see page 6744)

72.1.3 EWebUsersListException Class

EWebUsersListException is the base class for exceptions that TWebUserList (see page 6759) raises.

Class Hierarchy

Pascal

```
EWebUsersListException = class(EWebBrokerException);
```

C++

```
class EWebUsersListException : public EWebBrokerException;
```

File

WebUsers

Description

EWebUsersListException is the common base class for all exceptions that represent errors which TWebUserList (see page 6759) encounters. Its descendants include

EUserIDNotFoundException (see page 6742), which is raised when a WebSnap application tries to use the Web user list to look up information about a user that is not listed.

EValidateUserException (see page 6742), which is raised when the WebSnap application tries to log in a user and the Web user list fails to validate the supplied user id and password.

See Also

TWebUserList (see page 6759)

EUserIDNotFoundException (see page 6742)

EValidateUserException (see page 6742)

72.1.4 TCustomWebUserList Class

The TCustomWebUserList component uses a TWebUserItems (see page 6755) object to store the account information about a collection of users.

Class Hierarchy

Pascal

```
TCustomWebUserList = class(TComponent, IWebUserList);
```

C++

```
class TCustomWebUserList : public TComponent, public IWebUserList;
```

File

WebUsers

Description

The TCustomWebUserList component uses a TWebUserItems (see page 6755) object to store the account information about a collection of users. It also provides a way to save and load the user list from a file. Several events are triggered at stages that occur when a user is logging in that allow checks to be performed during the user validation process, including the verification of user rights.

See Also

TWebUserItems (see page 6755)

TWebUserList (see page 6759)

72.1.4.1 TCustomWebUserList Members

The following tables list the members exposed by TCustomWebUserList.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6746)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6747)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6744) object.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 6748)	Triggered at the start of the CheckAccessRights (see page 6746) protected function.
	OnAfterValidateUser (see page 6749)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6749)	Triggered at the end of the CheckAccessRights (see page 6746) protected function.
	OnBeforeValidateUser (see page 6749)	Triggered at the beginning of the ValidateUser (see page 6747) function.
	OnUserIDNotFound (see page 6750)	Triggered if the UserID is not found in the TCustomWebUserList (see page 6744) object.
	OnValidateUserError (see page 6750)	Triggered if an error occurs during the validation process.

Public Methods

	Name	Description
V	Create (see page 6745)	Creates a newTCustomWebUserList object.
V	Destroy (see page 6745)	Frees the TCustomWebUserList (see page 6744) object.
	LoadFromFile (see page 6746)	Loads the data from the file with the given filename.
	SaveToFile (see page 6747)	This procedure saves the collection of the user's account information to a file with the given filename.

Public Properties

	Name	Description
	UserItems (see page 6747)	Stores the account information about a collection of users.
R	Users (see page 6748)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6750).

72.1.4.2 TCustomWebUserList.Create Constructor

Creates a newTCustomWebUserList object.

Pascal

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

C++

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

Description

Creates a newTCustomWebUserList object and initializes the TWebUserItems (see page 6755) object that will store the user's account information.

72.1.4.3 TCustomWebUserList.Destroy Destructor

Frees the TCustomWebUserList (see page 6744) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebUserList();
```

Description

Frees the TCustomWebUserList (see page 6744) object initialized by the constructor.

72.1.4.4 TCustomWebUserList Methods

The methods of the TCustomWebUserList class are listed here.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6746)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6747)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6744) object.

Public Methods

	Name	Description
	LoadFromFile (see page 6746)	Loads the data from the file with the given filename.
	SaveToFile (see page 6747)	This procedure saves the collection of the user's account information to a file with the given filename.

72.1.4.4.1 TCustomWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(AUserID: Variant; ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant AUserID, UnicodeString ARights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 5497)

HasModifyRights (see page 5497)

HasExecuteRights (see page 5496)

ViewAccess

ViewAccess (see page 5560)

72.1.4.4.2 TCustomWebUserList.LoadFromFile Method

Loads the data from the file with the given filename.

Pascal

```
procedure LoadFromFile(FileName: string);
```

C++

```
__fastcall LoadFromFile(UnicodeString FileName);
```

Description

Loads the data from the file with the given filename into the TCustomWebUserList (see page 6744) object.

72.1.4.4.3 TCustomWebUserList.SaveToFile Method

This procedure saves the collection of the user's account information to a file with the given filename.

Pascal

```
procedure SaveToFile(FileName: string);
```

C++

```
__fastcall SaveToFile(UnicodeString FileName);
```

Description

Saves the collection of the user's account information to a file with the given filename. This file can later be reloaded by calling LoadFromFile (see page 6746).

72.1.4.4.4 TCustomWebUserList.ValidateUser Method

Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6744) object.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

This function verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6744) object. Before the validation begins, it triggers the OnBeforeValidateUser (see page 6749) event. If an error occurs during the validation check, the OnValidateUserError (see page 6750) event will be triggered and a description of the error is sent, which could be either a missing or invalid password, or a missing or unknown user name. After the validation is complete, this function triggers the OnAfterValidateUser (see page 6749) event.

72.1.4.5 TCustomWebUserList Properties

The properties of the TCustomWebUserList class are listed here.

Public Properties

	Name	Description
	UserItems (see page 6747)	Stores the account information about a collection of users.
R	Users (see page 6748)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6750).

72.1.4.5.1 TCustomWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 6748)

72.1.4.5.2 TCustomWebUserList.Users Property

Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6750).

Pascal

```
property Users [I: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Users[int I];
```

Description

Use this property to provide read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6750). An individual TWebUserItem (see page 6750) object can be indexed from the array using the standard bracketed array notation.

See Also

UserItems (see page 6747)

72.1.4.6 TCustomWebUserList Events

The events of the TCustomWebUserList class are listed here.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 6748)	Triggered at the start of the CheckAccessRights (see page 6746) protected function.
	OnAfterValidateUser (see page 6749)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6749)	Triggered at the end of the CheckAccessRights (see page 6746) protected function.
	OnBeforeValidateUser (see page 6749)	Triggered at the beginning of the ValidateUser (see page 6747) function.
	OnUserIDNotFound (see page 6750)	Triggered if the UserID is not found in the TCustomWebUserList (see page 6744) object.
	OnValidateUserError (see page 6750)	Triggered if an error occurs during the validation process.

72.1.4.6.1 TCustomWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights (see page 6746) protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights (see page 6746) protected function. This event can be used if any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 6749)

72.1.4.6.2 TCustomWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser (see page 6747) function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 6749)

ValidateUser (see page 6747)

72.1.4.6.3 TCustomWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights (see page 6746) protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights (see page 6746) protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 6748)

72.1.4.6.4 TCustomWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser (see page 6747) function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser (see page 6747) function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 6749)

72.1.4.6.5 TCustomWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList (see page 6744) object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList (see page 6744) object. The CheckAccessRights (see page 6746) protected function triggers this function if the UserID is not found while checking the access rights.

72.1.4.6.6 TCustomWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser (see page 6747) function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 6747)

72.1.5 TWebUserItem Class

TWebUserItem stores the user identification, password, and access rights for a single user.

Class Hierarchy

Pascal

```
TWebUserItem = class(TCollectionItem);
```

C++

```
class TWebUserItem : public TCollectionItem;
```

File

WebUsers

Description

TWebUserItem stores the user ID and password along with the associated access rights for a single user. Methods are provided to set the access rights as well as methods to retrieve the access rights in the form of a single delimited string, or as a collection in the form of a TString.

See Also

TWebUserList (see page 6759)

TWebUserItems (see page 6755)

72.1.5.1 TWebUserItem Members

The following tables list the members exposed by TWebUserItem.

Public Methods

	Name	Description
	Assign (see page 6752)	Copies the UserName (see page 6754), Password (see page 6754), and AccessRights (see page 6753) from the given TWebUserItem (see page 6750).
	CheckRights (see page 6753)	Checks the user's stored access rights against the access rights passed in.
	Create (see page 6751)	Creates an instance of Create.
	Destroy (see page 6752)	Destroys an instance of TWebUserItem (see page 6750).

Public Properties

	Name	Description
	AccessRightsStrings (see page 6754)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
	UserID (see page 6754)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 6753)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 6754)	Stores the user's password as a string.
	UserName (see page 6754)	Stores the user's name.

72.1.5.2 TWebUserItem.Create Constructor

Creates an instance of Create.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWebUserItem(TCollection Collection);
```

Description

Do not call Create. It is called automatically when the Create object is created.

See Also

Create_c

72.1.5.3 TWebUserItem.Destroy Destructor

Destroys an instance of TWebUserItem (see page 6750).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebUserItem();
```

Description

The Destroy destructor frees all objects owned by the TWebUserItem (see page 6750) object. Never call the destructor explicitly; it is called automatically when the TWebUserItem (see page 6750) object is destroyed (for example, when the object falls out of the scope of the program).

See Also

delete

72.1.5.4 TWebUserItem Methods

The methods of the TWebUserItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 6752)	Copies the UserName (see page 6754), Password (see page 6754), and AccessRights (see page 6753) from the given TWebUserItem (see page 6750).
	CheckRights (see page 6753)	Checks the user's stored access rights against the access rights passed in.

72.1.5.4.1 TWebUserItem.Assign Method

Copies the UserName (see page 6754), Password (see page 6754), and AccessRights (see page 6753) from the given TWebUserItem (see page 6750).

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

The Assign procedure copies the UserName (see page 6754), Password (see page 6754), and AccessRights (see page 6753) from the given TWebUserItem (see page 6750).

See Also

UserName (see page 6754)

Password (see page 6754)

AccessRights (see page 6753)

72.1.5.4.2 TWebUserItem.CheckRights Method

Checks the user's stored access rights against the access rights passed in.

Pascal

```
function CheckRights(const ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckRights(const UnicodeString ARights);
```

Description

This function checks the user's stored access rights against the access rights passed in. If the user has a matching right, then the function returns true or it will return false if the rights do not match.

See Also

AccessRights (see page 6753)

72.1.5.5 TWebUserItem Properties

The properties of the TWebUserItem class are listed here.

Public Properties

	Name	Description
R	AccessRightsStrings (see page 6754)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
R	UserID (see page 6754)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 6753)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 6754)	Stores the user's password as a string.
	UserName (see page 6754)	Stores the user's name.

72.1.5.5.1 TWebUserItem.AccessRights Property

Stores the user rights as a comma, semicolon, or space delimited string.

Pascal

```
property AccessRights: string;
```

C++

```
__property UnicodeString AccessRights;
```

Description

The AccessRights property stores the user rights as a comma, semicolon, or space delimited string. This variable can be read or written directly or set as a single string using SetAccessRights. The individual rights can be obtained as a collection in the form of a TString by calling the GetAccessRightsStrings function.

See Also

CheckRights (see page 6753)

72.1.5.5.2 TWebUserItem.AccessRightsStrings Property

Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

Pascal

```
property AccessRightsStrings: TStrings;
```

C++

```
__property TStrings AccessRightsStrings;
```

Description

The AccessRightsStrings property stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

See Also

AccessRights (see page 6753)

TStrings

72.1.5.5.3 TWebUserItem.Password Property

Stores the user's password as a string.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

The Password property stores the user's password as a string.

72.1.5.5.4 TWebUserItem.UserID Property

Provides read-only access to the user ID.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

Description

The UserID property provides read-only access to the user ID.

See Also

Variant (see page 3430)

72.1.5.5.5 TWebUserItem.UserName Property

Stores the user's name.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

The UserName property stores the user's name. This property can be read or written directly.

72.1.6 TWebUserItems Class

TWebUserItems stores multiple sets of user information.

Class Hierarchy

Pascal

```
TWebUserItems = class(TCollection);
```

C++

```
class TWebUserItems : public TCollection;
```

File

WebUsers

Description

The TWebUserItems stores multiple TWebUserItem (see page 6750) modules as an array. It also provides methods to access a particular user's information by giving the user name or user ID.

72.1.6.1 TWebUserItems Members

The following tables list the members exposed by TWebUserItems.

Protected Methods

	Name	Description
	GetAttr (see page 6757)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 6757)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 6758)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 6758)	Returns the owner of an object.

Public Methods

	Name	Description
	Create (see page 6756)	Constructor used for the TWebUserItems (see page 6755) object.
	FindUserID (see page 6756)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 6756)	Returns the user ID, password, and access rights information for the user.

Public Properties

	Name	Description
	Items (see page 6759)	Allows public read and write access to individual TWebUserItem (see page 6750) records as if they were stored in an array.

72.1.6.2 TWebUserItems.Create Constructor

Constructor used for the TWebUserItems (see page 6755) object.

Pascal

```
constructor Create(AOwner: TComponent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TWebUserItems(TComponent * AOwner, TCollectionItemClass ItemClass);
```

Description

The constructor used for the TWebUserItems (see page 6755) object. This method creates a new TWebUserItem (see page 6750) and sets the owner.

72.1.6.3 TWebUserItems Methods

The methods of the TWebUserItems class are listed here.

Protected Methods

	Name	Description
	GetAttr (see page 6757)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 6757)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 6758)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 6758)	Returns the owner of an object.

Public Methods

	Name	Description
	FindUserID (see page 6756)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 6756)	Returns the user ID, password, and access rights information for the user.

72.1.6.3.1 TWebUserItems.FindUserID Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserID(AUserID: Variant): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserID(Variant AUserID);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user name in the form of a TWebUserItem (see page 6750) object.

72.1.6.3.2 TWebUserItems.FindUserName Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserName(AUserName: string): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserName(UnicodeString AUserName);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user ID in the form of a TWebUserItem (see page 6750) object.

72.1.6.3.3 TWebUserItems.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount

GetItemAttr

72.1.6.3.4 TWebUserItems.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr

GetItemAttr

72.1.6.3.5 TWebUserItems.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

72.1.6.3.6 TWebUserItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

72.1.6.4 TWebUserItems Properties

The properties of the TWebUserItems class are listed here.

Public Properties

	Name	Description
	Items (see page 6759)	Allows public read and write access to individual TWebUserItem (see page 6750) records as if they were stored in an array.

72.1.6.4.1 TWebUserItems.Items Property

Allows public read and write access to individual TWebUserItem (see page 6750) records as if they were stored in an array.

Pascal

```
property Items [Index: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Items[int Index];
```

Description

The Items property allows public read and write access to individual TWebUserItem (see page 6750) records as if they were stored in an array. It also provides methods to access a particular user's information by giving the user name or user ID.

See Also

operator_sb

72.1.7 TWebUserList Class

TWebUserList validates login and checks access rights for a particular user.

Class Hierarchy

Pascal

```
TWebUserList = class(TCustomWebUserList);
```

C++

```
class TWebUserList : public TCustomWebUserList;
```

File

WebUsers

Description

Use TWebUserList, which contains a list of user names, passwords, and access rights, to validate login and to check a user's access rights.

The TWebUserList component provides events that are triggered during the initial stages when a user is logging in, before and after the user is validated, before and after the user's access rights are checked, in the case of an error, and when detecting an invalid user.

See Also

TWebUserItems (see page 6755)

TWebUserItem (see page 6750)

Adding Login Support

72.1.7.1 TWebUserList Members

The following tables list the members exposed by TWebUserList.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6746)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6747)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6744) object.

Public Methods

	Name	Description
	CheckAccessRights (see page 1058)	Checks whether a specified user has a given access rights string.
V	Create (see page 6745)	Creates a newTCustomWebUserList object.
V	Destroy (see page 6745)	Frees the TCustomWebUserList (see page 6744) object.
	LoadFromFile (see page 6746)	Loads the data from the file with the given filename.
	SaveToFile (see page 6747)	This procedure saves the collection of the user's account information to a file with the given filename.
	ValidateUser (see page 1059)	Validates a user login.

Public Properties

	Name	Description
R	Users (see page 6748)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6750).

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 6761)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 6761)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6762)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 6762)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 6762)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 6763)	Triggered if an error occurs during the validation process.

Published Properties

	Name	Description
	UserItems (see page 6761)	Stores the account information about a collection of users.

72.1.7.2 TWebUserList Properties

The properties of the TWebUserList class are listed here.

Published Properties

	Name	Description
	UserItems (see page 6761)	Stores the account information about a collection of users.

72.1.7.2.1 TWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 6748)

72.1.7.3 TWebUserList Events

The events of the TWebUserList class are listed here.

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 6761)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 6761)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6762)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 6762)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 6762)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 6763)	Triggered if an error occurs during the validation process.

72.1.7.3.1 TWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights protected function. This event can be used if any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 6749)

72.1.7.3.2 TWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 6749)

ValidateUser (see page 6747)

72.1.7.3.3 TWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 6748)

72.1.7.3.4 TWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 6749)

72.1.7.3.5 TWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList object. The CheckAccessRights protected function triggers this function if the UserID is not found while checking the access rights.

72.1.7.3.6 TWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 6747)

72.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TValidateUserError (see page 6764)	This is record WebUsers.TValidateUserError.

72.2.1 WebUsers.TValidateUserError Enumeration

Pascal

```
TValidateUserError = (
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
);
```

C++

```
enum TValidateUserError {
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
};
```

File

WebUsers

Description

This is record WebUsers.TValidateUserError.

72.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCheckAccessRightsEvent (see page 6765)	This is type WebUsers.TCheckAccessRightsEvent.
TCheckAccessRightsHandledEvent (see page 6765)	This is type WebUsers.TCheckAccessRightsHandledEvent.
TValidateUserErrorEvent (see page 6766)	This is type WebUsers.TValidateUserErrorEvent.
TValidateUserEvent (see page 6766)	This is type WebUsers.TValidateUserEvent.
TValidateUserHandledEvent (see page 6766)	This is type WebUsers.TValidateUserHandledEvent.

72.3.1 WebUsers.TCheckAccessRightsEvent Type

Pascal

```
TCheckAccessRightsEvent = procedure (UserID: Variant; Rights: string; var HasRight: Boolean)
of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean) ( TCheckAccessRightsEvent)();
```

File

WebUsers

Description

This is type WebUsers.TCheckAccessRightsEvent.

72.3.2 WebUsers.TCheckAccessRightsHandledEvent Type

Pascal

```
TCheckAccessRightsHandledEvent = procedure (UserID: Variant; Rights: string; var HasRight:
Boolean; var Handled: Boolean) of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean; var Handled: Boolean) (
TCheckAccessRightsHandledEvent)();
```

File

WebUsers

Description

This is type WebUsers.TCheckAccessRightsHandledEvent.

72.3.3 WebUsers.TValidateUserErrorEvent Type

Pascal

```
TValidateUserErrorEvent = procedure (Error: TValidateUserError; Strings: TStrings; var UserID:
Variant; var Handled: Boolean) of object;
```

C++

```
(Error: TValidateUserError; Strings: TStrings; var UserID: Variant; var Handled: Boolean) (
TValidateUserErrorEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserErrorEvent.

72.3.4 WebUsers.TValidateUserEvent Type

Pascal

```
TValidateUserEvent = procedure (Strings: TStrings; var UserID: Variant) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant) ( TValidateUserEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserEvent.

72.3.5 WebUsers.TValidateUserHandledEvent Type

Pascal

```
TValidateUserHandledEvent = procedure (Strings: TStrings; var UserID: Variant; var Handled:
Boolean) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant; var Handled: Boolean) ( TValidateUserHandledEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserHandledEvent.

73 WideStrings Namespace

This is namespace WideStrings.

Topics

Name	Description
Classes (see page 6768)	The following table lists classes in this documentation.
Interfaces (see page 6819)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 6821)	The following table lists structs, records, enums in this documentation.
Types (see page 6822)	The following table lists types in this documentation.

Classes

	Name	Description
	TWideStringList (see page 6768)	TWideStringList maintains a list of strings.
	TWideStrings (see page 6785)	TWideStrings is the base class for objects that represent a list of Unicode strings.
	TWideStringsEnumerator (see page 6816)	Enumerates through the strings of a TWideStrings (see page 6785) object.

Interfaces

	Name	Description
	IWideStringsAdapter (see page 6819)	This is class WideStrings.IWideStringsAdapter.

Structs, Records, Enums

	Name	Description
	TWideStringItem (see page 6821)	TWideStringItem defines a TWideStringList (see page 6768) entry.

Types

Name	Description
PWideStringItem (see page 6822)	PWideStringItem is a TWideStringItem (see page 6821) pointer.
PWideStringItemList (see page 6822)	PWideStringItemList is a TWideStringItemList (see page 6823) pointer.
TWideStringItemList (see page 6823)	TWideStringItemList defines an array of TWideStringItems.
TWideStringListSortCompare (see page 6823)	TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6768) class.

73.1 Classes

The following table lists classes in this documentation.

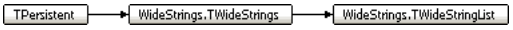
Classes

	Name	Description
	TWideStringList (see page 6768)	TWideStringList maintains a list of strings.
	TWideStrings (see page 6785)	TWideStrings is the base class for objects that represent a list of Unicode strings.
	TWideStringsEnumerator (see page 6816)	Enumerates through the strings of a TWideStrings (see page 6785) object.

73.1.1 TWideStringList Class

TWideStringList maintains a list of strings.

Class Hierarchy



Pascal

```
TWideStringList = class(TWideStrings);
```

C++

```
class TWideStringList : public TWideStrings;
```

File

WideStrings

Description

Use a string list object to store and manipulate a list of strings. TWideStringList implements the abstract properties and methods introduced by TWideStrings (see page 6785), and introduces new properties, events, and methods to

Sort (see page 6781) the strings in the list.

Prohibit duplicate strings in sorted lists.

Respond to changes in the contents of the list.

Control whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

See Also

TCollection

TList

Adding Graphical Objects to a String List

Adding Images to a String List

List Controls

Working with Lists

73.1.1.1 TWideStringList Members

The following tables list the members exposed by TWideStringList.

Protected Methods

	Name	Description
	AssignTo (see page 6792)	Copies the properties of an object to a destination object.
	Changed (see page 6773)	This is Changed, a member of class TWideStringList.
	Changing (see page 6774)	This is Changing, a member of class TWideStringList.
	CompareStrings (see page 6774)	Compares two strings.
	DefineProperties (see page 6794)	Reads and writes the Strings (see page 6813) property as if it were published.
	Error (see page 6795)	This is the overview for the Error method overload.
	ExtractName (see page 6797)	Returns the name portion of a string that is a name value pair.
	Get (see page 6777)	Returns a string given its index.
	GetCapacity (see page 6777)	Returns the currently allocated size of the list of strings.
	GetCount (see page 6777)	Returns the number of strings in the list
	GetObject (see page 6778)	Returns the object associated with the string at a specified index.
	GetTextStr (see page 6799)	Returns the value of the Text (see page 6814) property.
	InsertItem (see page 6779)	This is InsertItem, a member of class TWideStringList.
	Put (see page 6780)	Changes the value of the string with a specified index.
	PutObject (see page 6780)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6780)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6806)	Sets the Text (see page 6814) property.
	SetUpdateState (see page 6781)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 6815)	Indicates the number of calls to BeginUpdate (see page 6792) that have not been matched by a call to EndUpdate (see page 6795).

Public Events

	Name	Description
	OnChange (see page 6784)	Occurs immediately after the list of strings changes.
	OnChanging (see page 6785)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 6772)	Adds a new string to the list.
	AddObject (see page 6773)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6790)	This is the overview for the AddStrings method overload.
	Append (see page 6791)	Adds a string to the list.
	Assign (see page 6791)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6792)	Enables the TWideStrings (see page 6785) object to track when the list of strings is changing.
	Clear (see page 6774)	Deletes all the strings from the list.
	Create (see page 6770)	This is the overview for the Create constructor overload.
	CustomSort (see page 6775)	Sorts the strings in the list in a customized order.
	Delete (see page 6775)	Removes the string specified by the Index parameter.
	Destroy (see page 6771)	Destroys an instance of TWideStringList (see page 6768).
	EndUpdate (see page 6795)	Enables the TWideStrings (see page 6785) object to keep track of when the list of strings has finished changing.
	Equals (see page 6795)	Compares the list of strings to the list from another TWideStrings (see page 6785) object and returns true if the two lists match.
	Exchange (see page 6776)	Swaps the position of two strings in the list.

	Find (see page 6776)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	GetEnumerator (see page 6798)	Returns a TWideStrings (see page 6785) enumerator.
	GetText (see page 6799)	Allocates a text buffer and fills it with the value of the Text (see page 6814) property.
	IndexOf (see page 6778)	Returns the position of a string in the list.
	IndexOfName (see page 6800)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6800)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6779)	Inserts a string to the list at the position specified by Index.
	InsertObject (see page 6779)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6802)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6802)	Fills the list with lines of text read from a stream.
	Move (see page 6803)	Changes the position of a string in the list.
	SaveToFile (see page 6804)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6804)	Writes the value of the Text (see page 6814) property to a stream object.
	SetText (see page 6805)	Sets the Text (see page 6814) property.
	Sort (see page 6781)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 6807)	Indicates the number of strings the TWideStrings (see page 6785) object can hold.
	CaseSensitive (see page 6782)	Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 6807)	Lists the strings in the TWideStrings (see page 6785) object in system data format (SDF).
R	Count (see page 6809)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 6809)	Represents all the strings in the TWideStrings (see page 6785) object as a single delimited string.
	Delimiter (see page 6810)	Specifies the delimiter used by the DelimitedText (see page 6809) property.
	Duplicates (see page 6782)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 6810)	Defines the line break character(s).
R	Names (see page 6810)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6811)	Indicates the character used to separate names from values.
	Objects (see page 6811)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6813) property.
	OwnsObjects (see page 6783)	This is OwnsObjects, a member of class TWideStringList.
	QuoteChar (see page 6812)	Specifies the quote character used by the DelimitedText (see page 6809) property.
	Sorted (see page 6783)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 6812)	Determines how the Delimiter (see page 6810) property is used.
	Strings (see page 6813)	References the strings in the list by their positions.
	StringsAdapter (see page 6814)	Implements an IWideStringsAdapter (see page 6819) interface for the TWideStrings (see page 6785) object.
	Text (see page 6814)	Lists the strings in the TWideStrings (see page 6785) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6815)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6816)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

73.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWideStringList.Create () (see page 6771)	Constructs an object and initializes its data before the object is first used.
	TWideStringList.Create (Boolean) (see page 6771)	This is Create, a member of class TWideStringList.

73.1.1.2.1 TWideStringList.Create Constructor ()

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; overload;
```

C++

```
__fastcall TWideStringList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3366)

System::TObject::Free (see page 3382)

73.1.1.2.2 TWideStringList.Create Constructor (Boolean)

Pascal

```
constructor Create(OwnsObjects: Boolean); overload;
```

C++

```
__fastcall TWideStringList(Boolean OwnsObjects);
```

Description

This is Create, a member of class TWideStringList.

73.1.1.3 TWideStringList.Destroy Destructor

Destroys an instance of TWideStringList (see page 6768).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWideStringList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWideStringList (see page 6768) reference is not nil and only then calls Destroy.

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

System::TObject::Free (see page 3382)

73.1.1.4 TWideStringList Methods

The methods of the TWideStringList class are listed here.

Protected Methods

	Name	Description
	Changed (see page 6773)	This is Changed, a member of class TWideStringList.
	Changing (see page 6774)	This is Changing, a member of class TWideStringList.
	CompareStrings (see page 6774)	Compares two strings.
	Get (see page 6777)	Returns a string given its index.
	GetCapacity (see page 6777)	Returns the currently allocated size of the list of strings.
	GetCount (see page 6777)	Returns the number of strings in the list
	GetObject (see page 6778)	Returns the object associated with the string at a specified index.
	InsertItem (see page 6779)	This is InsertItem, a member of class TWideStringList.
	Put (see page 6780)	Changes the value of the string with a specified index.
	PutObject (see page 6780)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6780)	Changes the amount of memory allocated to hold strings in the list.
	SetUpdateState (see page 6781)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 6772)	Adds a new string to the list.
	AddObject (see page 6773)	Adds a string to the list, and associates an object with the string.
	Clear (see page 6774)	Deletes all the strings from the list.
	CustomSort (see page 6775)	Sorts the strings in the list in a customized order.
	Delete (see page 6775)	Removes the string specified by the Index parameter.
	Exchange (see page 6776)	Swaps the position of two strings in the list.
	Find (see page 6776)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	IndexOf (see page 6778)	Returns the position of a string in the list.
	Insert (see page 6779)	Inserts a string to the list at the position specified by Index.
	InsertObject (see page 6779)	Inserts a string into the list at the specified position, and associates it with an object.
	Sort (see page 6781)	Sorts the strings in the list in ascending order.

73.1.1.4.1 TWideStringList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: WideString): Integer; override;
```

C++

```
virtual __fastcall int Add(const BSTR S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates (see page 6782) is set to dupError. If Duplicates (see page 6782) is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

See Also

AddStrings (see page 6790)

Append (see page 6791)

Delete (see page 6775)

Duplicates (see page 6782)

Insert (see page 6779)

Sorted (see page 6783)

73.1.1.4.2 TWideStringList.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: WideString; AObject: TObject): Integer; override;
```

C++

```
virtual __fastcall int AddObject(const BSTR S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TWideStringList (see page 6768) object does not own the objects you add this way. Objects (see page 6811) added to the TWideStringList (see page 6768) object still exist even if the TWideStringList (see page 6768) instance is destroyed. They must be explicitly destroyed by the application.

Note: For sorted lists, AddObject raises an EListError exception if the string S already appears in the list and Duplicates (see page 6782) is set to dupError. If Duplicates (see page 6782) is set to dupIgnore, trying to add a duplicate string causes AddObject to return the index of the existing entry.

See Also

Add (see page 6772)

AddStrings (see page 6790)

73.1.1.4.3 TWideStringList.Changed Method

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TWideStringList.

73.1.1.4.4 TWideStringList.Changing Method

Pascal

```
procedure Changing; virtual;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TWideStringList.

73.1.1.4.5 TWideStringList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 6775)

Objects

Strings

73.1.1.4.6 TWideStringList.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: WideString; const S2: WideString): Integer; override;
```

C++

```
virtual __fastcall int CompareStrings(const BSTR S1, const BSTR S2);
```

Description

TWideStrings uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf and IndexOfName methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TWideStrings, CompareStrings uses the global WideCompareText function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

WideCompareText (see page 4355)

IndexOf (see page 6800)

IndexOfName (see page 6800)

73.1.1.4.7 TWideStringList.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TWideStringListSortCompare); virtual;
```

C++

```
virtual __fastcall CustomSort(TWideStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The List parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted (see page 6783) property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort (see page 6781) method.

See Also

Sort (see page 6781)

Strings (see page 6813)

Sorted (see page 6783)

73.1.1.4.8 TWideStringList.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 6772)

AddObject (see page 6773)

Clear (see page 6774)

IndexOf (see page 6778)

InsertObject

Objects (see page 6811)

Strings (see page 6813)

73.1.1.4.9 TWideStringList.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move (see page 6803)

Objects (see page 6811)

Sorted (see page 6783)

Strings (see page 6813)

73.1.1.4.10 TWideStringList.Find Method

Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.

Pascal

```
function Find(const S: WideString; var Index: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Find(const BSTR S, int Index);
```

Description

Use Find to obtain the index in a sorted list where the string S should be added. If the string S, or a string that differs from S only in case when CaseSensitive (see page 6782) is false, already exists in the list, Find returns true. If the list does not contain a string that matches S, Find returns false. The index where S should go is returned in the Index parameter. The value of Index is zero-based, where the first string has the index 0, the second string has the index 1, and so on.

Note: Only use Find with sorted lists. For unsorted lists, use the IndexOf (see page 6778) method instead.

See Also

Duplicates (see page 6782)

IndexOf (see page 6778)

Sorted (see page 6783)

Strings (see page 6813)

73.1.1.4.11 TWideStringList.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): WideString; override;
```

C++

```
virtual __fastcall BSTR Get(int Index);
```

Description

Get is the protected read implementation of the Strings property.

In TWideStrings Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 6813)

73.1.1.4.12 TWideStringList.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property. In TWideStrings, GetCapacity returns the value of the Count property. Descendants of TWideStrings can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 6807)

Count (see page 6809)

73.1.1.4.13 TWideStringList.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TWideStrings GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 6809)

73.1.1.4.14 TWideStringList.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property.

Index is the index of the string with which the object is associated.

In TWideStrings, GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 6811)

73.1.1.4.15 TWideStringList.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: WideString): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const BSTR S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. A string matches S if it is identical to S or, if CaseSensitive (see page 6782) is false, if it differs only in case.

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

Find (see page 6776)

IndexOfName (see page 6800)

IndexOfObject (see page 6800)

Strings (see page 6813)

73.1.1.4.16 TWideStringList.Insert Method

Inserts a string to the list at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; const S: WideString); override;
```

C++

```
virtual __fastcall Insert(int Index, const BSTR S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject (see page 6779) method instead.

Note: If the list is sorted, calling Insert or InsertObject (see page 6779) will raise an EListError exception. Use Add (see page 6772) or AddObject (see page 6773) with sorted lists.

See Also

Add (see page 6772)

AddObject (see page 6773)

EListError

InsertObject (see page 6779)

Sorted (see page 6783)

Strings (see page 6813)

73.1.1.4.17 TWideStringList.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: WideString; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertItem(int Index, const BSTR S, TObject * AObject);
```

Description

This is InsertItem, a member of class TWideStringList.

73.1.1.4.18 TWideStringList.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: WideString; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertObject(int Index, const BSTR S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by Index, and associate it with the object AObject. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

See Also

Insert (see page 6779)

Objects (see page 6811)

73.1.1.4.19 TWideStringList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: WideString); override;
```

C++

```
virtual __fastcall Put(int Index, const BSTR S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 6813)

Objects (see page 6811)

73.1.1.4.20 TWideStringList.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property.

As implemented in TWideStrings, PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 6811)

73.1.1.4.21 TWideStringList.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity property.

Integer is the number of strings the list can hold after the capacity has changed.

In TWideStrings, SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count removes strings from the end of the list. Assigning a value greater than Count allocates space for more strings to be added.

See Also

Capacity (see page 6807)

Count (see page 6809)

73.1.1.4.22 TWideStringList.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TWideStrings object is not already in the middle of an update, TWideStrings calls SetUpdateState internally, with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, TWideStrings calls SetUpdateState internally, with Updating set to false.

As implemented in TWideStrings, SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

73.1.1.4.23 TWideStringList.Sort Method

Sorts the strings in the list in ascending order.

Pascal

```
procedure Sort; virtual;
```

C++

```
virtual __fastcall Sort();
```

Description

Call Sort to sort the strings in a list that has the Sorted (see page 6783) property set to false. String lists with the Sorted (see page 6783) property set to true are automatically sorted.

Note: Sort uses WideCompareStr (see page 4354) to sort the strings when CaseSensitive (see page 6782) is true and WideCompareText (see page 4355) when CaseSensitive (see page 6782) is false. To provide your own comparison operator instead, use the CustomSort (see page 6775) method.

See Also

CustomSort (see page 6775)

Sorted (see page 6783)

Strings (see page 6813)

WideCompareStr (see page 4354)

WideCompareText (see page 4355)

73.1.1.5 TWideStringList Properties

The properties of the TWideStringList class are listed here.

Public Properties

	Name	Description
	CaseSensitive (see page 6782)	Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	Duplicates (see page 6782)	Specifies whether duplicate strings can be added to sorted lists.
	OwnsObjects (see page 6783)	This is OwnsObjects, a member of class TWideStringList.
	Sorted (see page 6783)	Specifies whether the strings in the list should be automatically sorted.

73.1.1.5.1 TWideStringList.CaseSensitive Property

Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether strings in the list should be compared in a case-sensitive or case-insensitive manner. Set CaseSensitive to true to make the string list locate, check for duplicates, and sort its strings in a case-sensitive manner. Set CaseSensitive to false to make the string list perform these operations case-insensitively.

See Also

CustomSort (see page 6775)

Sort (see page 6781)

73.1.1.5.2 TWideStringList.Duplicates Property

Specifies whether duplicate strings can be added to sorted lists.

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

Set Duplicates to specify what should happen when an attempt is made to add a duplicate string to a sorted list. The CaseSensitive (see page 6782) property controls whether two strings are considered duplicates if they are identical except for differences in case.

The value of Duplicates should be one of the following.

Value	Meaning
dupIgnore	Ignore attempts to add duplicate strings to the list.
dupError	raise an EStringListError exception when an attempt is made to add duplicate strings to the sorted list.
dupAccept	Permit duplicate strings in the sorted list.

Set Duplicates before adding any strings to the list. Setting Duplicates to dupIgnore or dupError does nothing about duplicate strings that are already in the list.

Note: Duplicates does nothing if the list is not sorted.

See Also

Add (see page 6772)

EStringListError

Sorted (see page 6783)

CaseSensitive (see page 6782)

73.1.1.5.3 TWideStringList.OwnsObjects Property

Pascal

```
property OwnsObjects: Boolean;
```

C++

```
__property Boolean OwnsObjects;
```

Description

This is OwnsObjects, a member of class TWideStringList.

73.1.1.5.4 TWideStringList.Sorted Property

Specifies whether the strings in the list should be automatically sorted.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to cause the strings in the list to be automatically sorted in ascending order. Set Sorted to false to allow strings to remain where they are inserted. When Sorted is false, the strings in the list can be put in ascending order at any time by calling the Sort (see page 6781) method.

When Sorted is true, do not use Insert (see page 6779) to add strings to the list. Instead, use Add (see page 6772), which will insert the new strings in the appropriate position. When Sorted is false, use Insert (see page 6779) to add strings to an arbitrary position in the list, or Add (see page 6772) to add strings to the end of the list.

Note: The CaseSensitive (see page 6782) property controls whether the strings in the list are sorted based on a case-sensitive or case-insensitive comparison. The sort order takes into account the locale of the system on which the application is running.

See Also

Add (see page 6772)

Duplicates (see page 6782)

Insert (see page 6779)

Sort (see page 6781)

WideCompareStr (see page 4354)

CaseSensitive (see page 6782)

73.1.1.6 TWideStringList Events

The events of the TWideStringList class are listed here.

Public Events

	Name	Description
	OnChange (see page 6784)	Occurs immediately after the list of strings changes.
	OnChanging (see page 6785)	Occurs immediately before the list of strings changes.

73.1.1.6.1 TWideStringList.OnChange Event

Occurs immediately after the list of strings changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3789) an OnChange event handler to respond to changes in the list of strings. For example, if the string list is associated with a control, the OnChange event handler could tell the control to repaint itself whenever the content of the list changes.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1. An OnChanging (see page 6785) event occurs before the change.
2. The strings are added, deleted, moved, or modified.

3. An OnChange event occurs.

   Note: OnChange occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 6792) and EndUpdate (see page 6795) around a series of changes.

See Also

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

OnChanging (see page 6785)

73.1.1.6.2 TWideStringList.OnChanging Event

Occurs immediately before the list of strings changes.

Pascal

```
property OnChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChanging;
```

Description

Write (see page 3789) an OnChanging event handler to prepare for changes in the list of strings. For example, if the string list is associated with a control, the OnChanging event handler could tell the control to disable repaints until the OnChange (see page 6784) event when the list has finished changing.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1. An OnChanging event occurs.
2. The strings are added, deleted, moved, or modified.
3. An OnChange (see page 6784) event occurs after the changes are complete.

   Note: OnChanging occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 6792) and EndUpdate (see page 6795) around a series of changes.

See Also

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

OnChange (see page 6784)

73.1.2 TWideStrings Class

TWideStrings is the base class for objects that represent a list of Unicode strings.

Class Hierarchy

Pascal

```
TWideStrings = class(TPersistent);
```

C++

```
class TWideStrings : public TPersistent;
```

File

WideStrings

Description

Derive a class from TWideStrings to store and manipulate a list of strings. TWideStrings contains abstract or, in C++ terminology, pure virtual methods and should not be directly instantiated.

Descendants of TWideStrings can represent several individual strings, such as the individual lines that appear in a list box. Some objects use descendants of TWideStrings to represent one long body of text so that it can be manipulated in smaller chunks.

TWideStrings introduces many properties and methods to

Add (see page 6789) or delete strings at specified positions in the list.

Rearrange the strings in the list.

Access the string at a particular location.

Read (see page 3727) the strings from or write the strings to a file or stream.

Associate an object with each string in the list.

Store and retrieve strings as name-value pairs.

For an explanation of name-value pairs, refer to the NameValueSeparator (see page 6811) property.

See Also

TWideStringList (see page 6768)

TStrings

TStringList

TList

THashedStringList

73.1.2.1 TWideStrings Members

The following tables list the members exposed by TWideStrings.

Protected Methods

	Name	Description
	AssignTo (see page 6792)	Copies the properties of an object to a destination object.
	CompareStrings (see page 6793)	Compares two strings.
	DefineProperties (see page 6794)	Reads and writes the Strings (see page 6813) property as if it were published.
	Error (see page 6795)	This is the overview for the Error method overload.
	ExtractName (see page 6797)	Returns the name portion of a string that is a name value pair.
	Get (see page 6797)	Returns a string given its index.
	GetCapacity (see page 6797)	Returns the currently allocated size of the list of strings.
	GetCount (see page 6798)	Returns the number of strings in the list

	GetObject (see page 6798)	Returns the object associated with the string at a specified index.
	GetTextStr (see page 6799)	Returns the value of the Text (see page 6814) property.
	Put (see page 6803)	Changes the value of the string with a specified index.
	PutObject (see page 6804)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6805)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6806)	Sets the Text (see page 6814) property.
	SetUpdateState (see page 6806)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 6815)	Indicates the number of calls to BeginUpdate (see page 6792) that have not been matched by a call to EndUpdate (see page 6795).

Public Methods

	Name	Description
	Add (see page 6789)	Adds a string at the end of the list.
	AddObject (see page 6790)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6790)	This is the overview for the AddStrings method overload.
	Append (see page 6791)	Adds a string to the list.
	Assign (see page 6791)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6792)	Enables the TWideStrings (see page 6785) object to track when the list of strings is changing.
	Clear (see page 6793)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Delete (see page 6794)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.
	Destroy (see page 6788)	Destroys the TPersistent instance and frees its memory.
	EndUpdate (see page 6795)	Enables the TWideStrings (see page 6785) object to keep track of when the list of strings has finished changing.
	Equals (see page 6795)	Compares the list of strings to the list from another TWideStrings (see page 6785) object and returns true if the two lists match.
	Exchange (see page 6796)	Swaps the position of two strings in the list.
	GetEnumerator (see page 6798)	Returns a TWideStrings (see page 6785) enumerator.
	GetText (see page 6799)	Allocates a text buffer and fills it with the value of the Text (see page 6814) property.
	IndexOf (see page 6800)	Returns the position of a string in the list.
	IndexOfName (see page 6800)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6800)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6801)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
	InsertObject (see page 6801)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6802)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6802)	Fills the list with lines of text read from a stream.
	Move (see page 6803)	Changes the position of a string in the list.
	SaveToFile (see page 6804)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6804)	Writes the value of the Text (see page 6814) property to a stream object.
	SetText (see page 6805)	Sets the Text (see page 6814) property.

Public Properties

	Name	Description
	Capacity (see page 6807)	Indicates the number of strings the TWideStrings (see page 6785) object can hold.
	CommaText (see page 6807)	Lists the strings in the TWideStrings (see page 6785) object in system data format (SDF).
	Count (see page 6809)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 6809)	Represents all the strings in the TWideStrings (see page 6785) object as a single delimited string.

	Delimiter (see page 6810)	Specifies the delimiter used by the DelimitedText (see page 6809) property.
	LineBreak (see page 6810)	Defines the line break character(s).
R	Names (see page 6810)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6811)	Indicates the character used to separate names from values.
	Objects (see page 6811)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6813) property.
	QuoteChar (see page 6812)	Specifies the quote character used by the DelimitedText (see page 6809) property.
	StrictDelimiter (see page 6812)	Determines how the Delimiter (see page 6810) property is used.
	Strings (see page 6813)	References the strings in the list by their positions.
	StringsAdapter (see page 6814)	Implements an IWideStringsAdapter (see page 6819) interface for the TWideStrings (see page 6785) object.
	Text (see page 6814)	Lists the strings in the TWideStrings (see page 6785) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6815)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6816)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

73.1.2.2 TWideStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWideStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

System::TObject::Destroy (see page 3366)

73.1.2.3 TWideStrings Methods

The methods of the TWideStrings class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 6792)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 6793)	Compares two strings.
V	DefineProperties (see page 6794)	Reads and writes the Strings (see page 6813) property as if it were published.
	Error (see page 6795)	This is the overview for the Error method overload.
	ExtractName (see page 6797)	Returns the name portion of a string that is a name value pair.
A	Get (see page 6797)	Returns a string given its index.
V	GetCapacity (see page 6797)	Returns the currently allocated size of the list of strings.
A	GetCount (see page 6798)	Returns the number of strings in the list
V	GetObject (see page 6798)	Returns the object associated with the string at a specified index.
V	GetTextStr (see page 6799)	Returns the value of the Text (see page 6814) property.
V	Put (see page 6803)	Changes the value of the string with a specified index.

	PutObject (see page 6804)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6805)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6806)	Sets the Text (see page 6814) property.
	SetUpdateState (see page 6806)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 6789)	Adds a string at the end of the list.
	AddObject (see page 6790)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6790)	This is the overview for the AddStrings method overload.
	Append (see page 6791)	Adds a string to the list.
	Assign (see page 6791)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6792)	Enables the TWideStrings (see page 6785) object to track when the list of strings is changing.
	Clear (see page 6793)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Delete (see page 6794)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.
	EndUpdate (see page 6795)	Enables the TWideStrings (see page 6785) object to keep track of when the list of strings has finished changing.
	Equals (see page 6795)	Compares the list of strings to the list from another TWideStrings (see page 6785) object and returns true if the two lists match.
	Exchange (see page 6796)	Swaps the position of two strings in the list.
	GetEnumerator (see page 6798)	Returns a TWideStrings (see page 6785) enumerator.
	GetText (see page 6799)	Allocates a text buffer and fills it with the value of the Text (see page 6814) property.
	IndexOf (see page 6800)	Returns the position of a string in the list.
	IndexOfName (see page 6800)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6800)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6801)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
	InsertObject (see page 6801)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6802)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6802)	Fills the list with lines of text read from a stream.
	Move (see page 6803)	Changes the position of a string in the list.
	SaveToFile (see page 6804)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6804)	Writes the value of the Text (see page 6814) property to a stream object.
	SetText (see page 6805)	Sets the Text (see page 6814) property.

73.1.2.3.1 TWideStrings.Add Method

Adds a string at the end of the list.

Pascal

```
function Add(const S: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const BSTR S);
```

Description

Call Add to add a string to the end of the list. Add returns the index of the new string.

See Also

AddStrings (see page 6790)

Append (see page 6791)

Count (see page 6809)

IndexOf (see page 6800)

Insert (see page 6801)

73.1.2.3.2 TWideStrings.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: WideString; AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int AddObject(const BSTR S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TWideStrings (see page 6785) object does not own the objects you add this way. Objects (see page 6811) added to the TWideStrings (see page 6785) object still exist even if the TWideStrings (see page 6785) instance is destroyed. They must be explicitly destroyed by the application.

See Also

Add (see page 6789)

AddStrings (see page 6790)

Count (see page 6809)

IndexOfObject (see page 6800)

Objects (see page 6811)

73.1.2.3.3 AddStrings Method

This is the overview for the AddStrings method overload.

Overload List

	Name	Description
	TWideStrings.AddStrings (TStrings) (see page 6790)	Adds a group of strings to the list.
	TWideStrings.AddStrings (TWideStrings) (see page 6790)	Adds a group of strings to the list.

73.1.2.3.3.1 TWideStrings.AddStrings Method (TStrings)

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); virtual; overload;
procedure AddStrings(Strings: TWideStrings); virtual; overload;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
virtual __fastcall AddStrings(TWideStrings Strings);
```

Description

Call AddStrings to add the strings from another TWideStrings (see page 6785) object to the list. If both the source and destination TWideStrings (see page 6785) objects support objects associated with their strings, references to the associated objects will be added as well.

See Also

Add (see page 6789)

AddObject (see page 6790)

Assign (see page 6791)

Count (see page 6809)

Objects (see page 6811)

73.1.2.3.4 TWideStrings.Append Method

Adds a string to the list.

Pascal

```
procedure Append(const S: WideString);
```

C++

```
__fastcall Append(const BSTR S);
```

Description

Append is the same as the Add (see page 6789) method, except that it does not return a value. Use Append when there is no need to know the index of the string after it has been added, or with descendants of TWideStrings (see page 6785) for which the index returned is not meaningful.

For example, the TWideStrings (see page 6785) descendant used by memo objects uses an index to determine where to insert a string, but the inserted string does not necessarily end up as a single string in the list. Part of the inserted text may become part of the previous string, and part may be broken off into a subsequent string. The index returned by Add (see page 6789) is not meaningful in this case.

See Also

Add (see page 6789)

73.1.2.3.5 TWideStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TWideStrings (see page 6785) object from another object. If Source is of type TWideStrings (see page 6785), the list is set to the list of the source TWideStrings (see page 6785) object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TWideStrings (see page 6785), the inherited Assign will set the value of the list from any object that

supports TWideStrings (see page 6785) in its AssignTo (see page 6792) method.

See Also

AddStrings (see page 6790)

Objects (see page 6811)

Strings (see page 6813)

AssignTo

73.1.2.3.6 TWideStrings.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3985)

73.1.2.3.7 TWideStrings.BeginUpdate Method

Enables the TWideStrings (see page 6785) object to track when the list of strings is changing.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate is called automatically by any property or method that changes the list of strings. Once the changes are complete, the property or method calls EndUpdate (see page 6795). Call BeginUpdate before directly modifying the strings in the list, and EndUpdate (see page 6795) after. When implementing properties or methods that change the list in descendants of TWideStrings (see page 6785), call BeginUpdate before the changes are made, and EndUpdate (see page 6795) when the changes are complete.

TWideStrings (see page 6785) simply keeps track of when the list of strings is being changed. Some descendants of TWideStrings (see page 6785) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

See Also

EndUpdate (see page 6795)

UpdateCount (see page 6815)

73.1.2.3.8 TWideStrings.Clear Method

Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Descendants of TWideStrings (see page 6785) implement a Clear method to delete all the strings in the list, and to remove any references to associated objects.

See Also

Delete (see page 6794)

Objects (see page 6811)

Strings (see page 6813)

73.1.2.3.9 TWideStrings.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: WideString; const S2: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const BSTR S1, const BSTR S2);
```

Description

TWideStrings (see page 6785) uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf (see page 6800) and IndexOfName (see page 6800) methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TWideStrings (see page 6785), CompareStrings uses the global WideCompareText (see page 4355) function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

WideCompareText (see page 4355)

IndexOf (see page 6800)

IndexOfName (see page 6800)

73.1.2.3.10 TWideStrings.DefineProperties Method

Reads and writes the Strings (see page 6813) property as if it were published.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TWideStrings (see page 6785) overrides DefineProperties so that the strings in the list can be loaded and saved with a form file as if the Strings (see page 6813) property were published.

See Also

DefineProperties

Strings (see page 6813)

73.1.2.3.11 TWideStrings.Delete Method

Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Descendants of TWideStrings (see page 6785) implement a Delete method to remove a specified string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

AddObject (see page 6790)

Clear (see page 6793)

IndexOf (see page 6800)

InsertObject (see page 6801)

Objects (see page 6811)

Strings (see page 6813)

73.1.2.3.12 TWideStrings.EndUpdate Method

Enables the TWideStrings (see page 6785) object to keep track of when the list of strings has finished changing.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

EndUpdate is called automatically by any property or method that changes the list of strings. Call BeginUpdate (see page 6792) before directly modifying the strings in the list, and EndUpdate after. When implementing properties or methods that change the list in descendants of TWideStrings (see page 6785), call BeginUpdate (see page 6792) before the changes are made, and EndUpdate when the changes are complete.

TWideStrings (see page 6785) simply keeps track of when the list of strings is being changed. Some descendants of TWideStrings (see page 6785) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

See Also

BeginUpdate (see page 6792)

UpdateCount (see page 6815)

73.1.2.3.13 TWideStrings.Equals Method

Compares the list of strings to the list from another TWideStrings (see page 6785) object and returns true if the two lists match.

Pascal

```
function Equals(Strings: TWideStrings): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TWideStrings Strings);
```

Description

Call Equals to compare the lists in two TWideStrings (see page 6785) objects. Equals compares only the strings, not any references to associated objects. Equals returns true if the lists for both TWideStrings (see page 6785) objects have the same number of strings and the strings in each list match. Equals returns false if the lists are different in length, if they contain different strings, or if the order of the strings in the two lists differ.

Note: The Equals method uses an overridden Get (see page 6797) method to perform the compare, thus it does not provide a context-insensitive option.

See Also

Count (see page 6809)

Strings (see page 6813)

CompareStrings (see page 6793)

73.1.2.3.14 Error Method

This is the overview for the Error method overload.

Overload List

	Name	Description
	TWideStrings.Error (PResStringRec, Integer) (see page 6796)	raises an EStringListError exception.
	TWideStrings.Error (WideString, Integer) (see page 6796)	raises an EStringListError exception.

73.1.2.3.14.1 TWideStrings.Error Method (PResStringRec, Integer)

raises an EStringListError exception.

Pascal

```
procedure Error(const Msg: WideString; Data: Integer); overload;
procedure Error(Msg: PResStringRec; Data: Integer); overload;
```

C++

```
__fastcall Error(const BSTR Msg, int Data);
__fastcall Error(PResStringRec Msg, int Data);
```

Description

TWideStrings (see page 6785) calls Error internally to raise an EStringListError exception when it encounters a problem.

Msg specifies the string with a single format specifier for an integer, that appears in the exception message box. It can be either a string, or a pointer to a record structure that indicates the module and resource identifier for a string.

Data is an integer value that is inserted into Msg.

See Also

EStringListError

73.1.2.3.15 TWideStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the position of the object as well.

See Also

Move (see page 6803)

Objects (see page 6811)

Strings (see page 6813)

73.1.2.3.16 TWideStrings.ExtractName Method

Returns the name portion of a string that is a name value pair.

Pascal

```
function ExtractName(const S: WideString): WideString;
```

C++

```
__fastcall BSTR ExtractName(const BSTR S);
```

Description

TWideStrings (see page 6785) calls ExtractName internally to parse strings that are name-value pairs and return the name portion.

S is the string to parse.

If S is a name-value pair, ExtractName returns the name portion. Otherwise, ExtractName returns an empty string.

See Also

Names (see page 6810)

73.1.2.3.17 TWideStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR Get(int Index) = 0;
```

Description

Get is the protected read implementation of the Strings (see page 6813) property.

In TWideStrings (see page 6785) Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 6813)

73.1.2.3.18 TWideStrings.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 6807) property. In TWideStrings (see page 6785), GetCapacity returns the value of the Count (see page 6809) property. Descendants of TWideStrings (see page 6785) can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 6807)

Count (see page 6809)

73.1.2.3.19 TWideStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCount is the protected read implementation of the Count (see page 6809) property.

In TWideStrings (see page 6785) GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 6809)

73.1.2.3.20 TWideStrings.GetEnumerator Method

Returns a TWideStrings (see page 6785) enumerator.

Pascal

```
function GetEnumerator: TWideStringsEnumerator;
```

C++

```
__fastcall TWideStringsEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TWideStringsEnumerator (see page 6816) reference, which enumerates the strings in a TWideStrings (see page 6785) object.

To process all these strings, call the TWideStringsEnumerator (see page 6816) GetCurrent (see page 3679) method within a While MoveNext do loop.

See Also

TWideStringsEnumerator (see page 6816)

73.1.2.3.21 TWideStrings.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; virtual;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects (see page 6811) property.

Index is the index of the string with which the object is associated.

In TWideStrings (see page 6785), GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 6811)

73.1.2.3.22 TWideStrings.GetText Method

Allocates a text buffer and fills it with the value of the Text (see page 6814) property.

Pascal

```
function GetText: PwideChar; virtual;
```

C++

```
virtual __fastcall PwideChar GetText();
```

Description

Call GetText to obtain a dynamically allocated character buffer containing all of the strings in the list. Individual strings are separated by a carriage return and (on Windows) line feed. The caller is responsible for freeing the returned value using the StrDispose (see page 4291) procedure.

See Also

SetText (see page 6805)

Strings (see page 6813)

Text (see page 6814)

StrDispose (see page 4291)

73.1.2.3.23 TWideStrings.GetTextStr Method

Returns the value of the Text (see page 6814) property.

Pascal

```
function GetTextStr: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetTextStr();
```

Description

GetTextStr is the protected read implementation of the Text (see page 6814) property. It returns a string that lists all the strings in the list, with individual strings separated by a carriage return and (on Windows) line feed.

See Also

GetText (see page 6799)

Text (see page 6814)

73.1.2.3.24 TWideStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOf(const BSTR S);
```

Description

Call IndexOf to obtain the position of the first occurrence of the string S, or of a string that differs from S only by case. IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string is not in the string list, IndexOf returns -1.

Note: If the string appears in the list more than once, IndexOf returns the position of the first occurrence.

See Also

IndexOfName (see page 6800)

IndexOfObject (see page 6800)

Strings (see page 6813)

73.1.2.3.25 TWideStrings.IndexOfName Method

Returns the position of the first name-value pair with the specified name.

Pascal

```
function IndexOfName(const Name: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const BSTR Name);
```

Description

Call IndexOfName to locate the first occurrence of a name-value pair where the name part is equal to the Name parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

Note: If there is more than one name-value pair with a name portion matching the Name parameter, IndexOfName returns the position of the first such string.

See Also

IndexOf (see page 6800)

Names (see page 6810)

Values (see page 6816)

73.1.2.3.26 TWideStrings.IndexOfObject Method

Returns the index of the first string in the list associated with a given object.

Pascal

```
function IndexOfObject(AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfObject(TObject * AObject);
```

Description

Call IndexOfObject to locate the first string in the list associated with the object AObject. Specify the object you want to locate as the value of the AObject parameter. IndexOfObject returns the 0-based index of the string and object. If the object is not associated with any of the strings, IndexOfObject returns -1.

See Also

IndexOf (see page 6800)

Objects (see page 6811)

73.1.2.3.27 TWideStrings.Insert Method

Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: WideString); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const BSTR S) = 0;
```

Description

Descendants of TWideStrings (see page 6785) implement an Insert method to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

All methods that add strings to the list use the Insert method to add the string.

If the string has an associated object, use the InsertObject (see page 6801) method instead.

See Also

Add (see page 6789)

AddObject (see page 6790)

AddStrings (see page 6790)

Append (see page 6791)

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

InsertObject (see page 6801)

Strings (see page 6813)

73.1.2.3.28 TWideStrings.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: WideString; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertObject(int Index, const BSTR S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by Index, and associate it with the object AObject. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

See Also

Insert

Objects

73.1.2.3.29 TWideStrings.LoadFromFile Method

Fills the list with the lines of text in a specified file.

Pascal

```
procedure LoadFromFile(const FileName: WideString); virtual;
```

C++

```
virtual __fastcall LoadFromFile(const BSTR FileName);
```

Description

Call LoadFromFile to fill the list of the TWideStrings (see page 6785) object from the file specified by FileName. LoadFromFile first clears any strings already in the list. Then, each line in the file, as indicated by carriage return or linefeed characters, is appended as a string in the list.

Note: LoadFromFile uses the Add (see page 6789) method to add the strings that are read from the file.

See Also

Add (see page 6789)

LoadFromStream (see page 6802)

SaveToFile (see page 6804)

Strings (see page 6813)

Text (see page 6814)

73.1.2.3.30 TWideStrings.LoadFromStream Method

Fills the list with lines of text read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to fill the list of the TWideStrings (see page 6785) object from the stream specified by Stream. The text read from the stream is parsed into strings separated by carriage return or linefeed characters. Thus, LoadFromStream reads the value of the Text (see page 6814) property.

If the stream is a file stream, LoadFromStream does the same thing as LoadFromFile (see page 6802), except the application must create and destroy the file stream.

See Also

Add (see page 6789)

LoadFromFile (see page 6802)

SaveToStream (see page 6804)

Text (see page 6814)

73.1.2.3.31 TWideStrings.Move Method

Changes the position of a string in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Use Move to move the string at position CurIndex so that it occupies the position NewIndex. The positions are specified as 0-based indexes. For example, the following lines of code move the string in the first position to the last position.

```
MyStringsObject.Move(0, MyStringsObject.Count)
MyStringsObject->Move(0, MyStringsObject->Count - 1);
```

If the string has an associated object, the object remains associated with the string in its new position.

See Also

Count (see page 6809)

Objects (see page 6811)

Strings (see page 6813)

73.1.2.3.32 TWideStrings.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: WideString); virtual;
```

C++

```
virtual __fastcall Put(int Index, const BSTR S);
```

Description

Put is the protected write implementation of the Strings (see page 6813) property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 6813)

Objects (see page 6811)

73.1.2.3.33 TWideStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); virtual;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects (see page 6811) property.

As implemented in TWideStrings (see page 6785), PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 6811)

73.1.2.3.34 TWideStrings.SaveToFile Method

Saves the strings in the list to the specified file.

Pascal

```
procedure SaveToFile(const FileName: WideString); virtual;
```

C++

```
virtual __fastcall SaveToFile(const BSTR FileName);
```

Description

Call SaveToFile to save the strings in the list to the file specified by FileName. Each string in the list is written to a separate line in the file.

Note: In Linux, SaveToFile saves the strings as they appear in the list. If the last string does not have an end-of-line marker, the resulting file may be considered incomplete by some editors.

See Also

LoadFromFile (see page 6802)

SaveToStream (see page 6804)

Strings (see page 6813)

Text (see page 6814)

73.1.2.3.35 TWideStrings.SaveToStream Method

Writes the value of the Text (see page 6814) property to a stream object.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the strings in the list to the stream specified by the Stream parameter. SaveToStream writes the strings delimited by carriage return, line feed pairs. If the stream is a file stream, SaveToStream does the same thing as SaveToFile (see page 6804), except the application must create and destroy the file stream.

See Also

LoadFromStream (see page 6802)

SaveToFile (see page 6804)

Strings (see page 6813)

Text (see page 6814)

73.1.2.3.36 TWideStrings.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity (see page 6807) property.

Integer is the number of strings the list can hold after the capacity has changed.

In TWideStrings (see page 6785), SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count (see page 6809) removes strings from the end of the list. Assigning a value greater than Count (see page 6809) allocates space for more strings to be added.

See Also

Capacity (see page 6807)

Count (see page 6809)

73.1.2.3.37 TWideStrings.SetText Method

Sets the Text (see page 6814) property.

Pascal

```
procedure SetText(Text: PwideChar); virtual;
```

C++

```
virtual __fastcall SetText(PwideChar Text);
```

Description

Call SetText to replace the list with the strings specified by the Text (see page 6814) parameter. SetText adds strings one at a time to the list, using the carriage returns or linefeed characters in Text (see page 6814) as delimiters indicating when to add a new string.

See Also

SetTextStr (see page 6806)

Text (see page 6814)

73.1.2.3.38 TWideStrings.SetTextStr Method

Sets the Text (see page 6814) property.

Pascal

```
procedure SetTextStr(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetTextStr(const BSTR Value);
```

Description

GetTextStr (see page 6799) is the protected write implementation of the Text (see page 6814) property. It replaces the list with the strings specified by the Value parameter. SetTextStr adds strings one at a time to the list, using the carriage returns or linefeed characters in Value as delimiters indicating when to add a new string.

See Also

SetText (see page 6805)

Text (see page 6814)

73.1.2.3.39 TWideStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate (see page 6792) method is first called and the TWideStrings (see page 6785) object is not already in the middle of an update, TWideStrings (see page 6785) calls SetUpdateState internally, with Updating set to true. When the EndUpdate (see page 6795) method is called and it cancels out the last unmatched call to BeginUpdate (see page 6792), TWideStrings (see page 6785) calls SetUpdateState internally, with Updating set to false.

As implemented in TWideStrings (see page 6785), SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

73.1.2.4 TWideStrings Properties

The properties of the TWideStrings class are listed here.

Protected Properties

	Name	Description
R	UpdateCount (see page 6815)	Indicates the number of calls to BeginUpdate (see page 6792) that have not been matched by a call to EndUpdate (see page 6795).

Public Properties

	Name	Description
	Capacity (see page 6807)	Indicates the number of strings the TWideStrings (see page 6785) object can hold.
	CommaText (see page 6807)	Lists the strings in the TWideStrings (see page 6785) object in system data format (SDF).
R	Count (see page 6809)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 6809)	Represents all the strings in the TWideStrings (see page 6785) object as a single delimited string.
	Delimiter (see page 6810)	Specifies the delimiter used by the DelimitedText (see page 6809) property.
	LineBreak (see page 6810)	Defines the line break character(s).
R	Names (see page 6810)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6811)	Indicates the character used to separate names from values.
	Objects (see page 6811)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6813) property.
	QuoteChar (see page 6812)	Specifies the quote character used by the DelimitedText (see page 6809) property.
	StrictDelimiter (see page 6812)	Determines how the Delimiter (see page 6810) property is used.
	Strings (see page 6813)	References the strings in the list by their positions.
	StringsAdapter (see page 6814)	Implements an IWideStringsAdapter (see page 6819) interface for the TWideStrings (see page 6785) object.
	Text (see page 6814)	Lists the strings in the TWideStrings (see page 6785) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6815)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6816)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

73.1.2.4.1 TWideStrings.Capacity Property

Indicates the number of strings the TWideStrings (see page 6785) object can hold.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Read (see page 3727) Capacity to determine the currently allocated size of the string list. For the TWideStrings (see page 6785) object, reading Capacity returns the Count (see page 6809) property, and setting Capacity does nothing. Descendants of TWideStrings (see page 6785) can override this property to allow a string list to allocate memory for entries that have not been added to the list.

See Also

Count (see page 6809)

73.1.2.4.2 TWideStrings.CommaText Property

Lists the strings in the TWideStrings (see page 6785) object in system data format (SDF).

Pascal

```
property CommaText: WideString;
```

C++

```
__property BSTR CommaText;
```

Description

Use CommaText to get or set all the strings in the TWideStrings (see page 6785) object in a single comma-delimited string.

When retrieving CommaText, any string in the list that include spaces, commas or quotes will be contained in double quotes, and any double quotes in a string will be repeated. For example, if the list contains the following strings:

```
Stri,ng 1
Stri"ng 2
String 3
String4
```

CommaText will return:

```
"Stri,ng 1","Stri""ng 2","String 3",String4
```

When assigning CommaText, the value is parsed as SDF formatted text. For SDF format, strings are separated by commas or spaces, and optionally enclosed in double quotes. Double quote marks that are part of the string are repeated to distinguish them from the quotes that surround the string. Spaces and commas that are not contained within double quote marks are delimiters. Two commas next to each other will indicate an empty string, but spaces that appear next to another delimiter are ignored. For example, suppose CommaText is set to:

```
"Stri,ng 1", "Stri""ng 2" , String 3,String4
```

The list will then contain:

```
Stri,ng 1
Stri"ng 2
String
3
String4
```

Note: CommaText is the same as the DelimitedText (see page 6809) property with a delimiter of ',' and a quote character of "".

Including a trailing comma in the source string causes a blank item to be included in the string list. For example, if CommaText is set to

```
"String1, String 2, String 3,"
```

the string list will contain

```
String1
String2
String3
<Blank>
```

See Also

Add (see page 6789)

Strings (see page 6813)

Text (see page 6814)

DelimitedText (see page 6809)

73.1.2.4.3 TWideStrings.Count Property

Introduces an abstract property to represent the number of strings in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Descendants of TWideStrings (see page 6785) implement a Count property to indicate the number of strings in the list.

Use the Count property when iterating over all the strings in the list, or when trying to locate the position of a string relative to the last string in the list.

See Also

IndexOf (see page 6800)

Strings (see page 6813)

73.1.2.4.4 TWideStrings.DelimitedText Property

Represents all the strings in the TWideStrings (see page 6785) object as a single delimited string.

Pascal

```
property DelimitedText: WideString;
```

C++

```
__property BSTR DelimitedText;
```

Description

Use DelimitedText to get or set all the strings in the TWideStrings (see page 6785) object in a single string.

When retrieving DelimitedText, the resulting value delimits individual strings in two ways: each string is surrounded (before and after) by the quote character specified by the QuoteChar (see page 6812) property. In addition, individual strings are separated by the character specified by the Delimiter (see page 6810) property.

When assigning DelimitedText, individual strings must be separated using QuoteChar (see page 6812) at both ends, using Delimiter (see page 6810) as a separator, or using both these methods.

A space is always interpreted as a delimiter, regardless of the value of Delimiter (see page 6810), unless the space occurs between quote characters.

Note: CommaText (see page 6807) is the same as the DelimitedText property when Delimiter (see page 6810) is ',' and QuoteChar (see page 6812) is "".

See Also

CommaText (see page 6807)

Delimiter (see page 6810)

Text (see page 6814)

QuoteChar (see page 6812)

73.1.2.4.5 TWideStrings.Delimiter Property

Specifies the delimiter used by the DelimitedText (see page 6809) property.

Pascal

```
property Delimiter: WideChar;
```

C++

```
__property WideChar Delimiter;
```

Description

Use Delimiter to get or set the delimiter used by the DelimitedText (see page 6809) property. DelimitedText (see page 6809) represents all of the strings in the TWideStrings (see page 6785) object as a single string, with individual strings separated by the character that is the value of Delimiter.

See Also

DelimitedText (see page 6809)

QuoteChar (see page 6812)

73.1.2.4.6 TWideStrings.LineBreak Property

Defines the line break character(s).

Pascal

```
property LineBreak: WideString;
```

C++

```
__property BSTR LineBreak;
```

Description

LineBreak can be read and set. It is used internally in TWideStrings (see page 6785) to terminate strings in multi-string output operations.

For example, the GetText (see page 6799) method returns a long string containing all TWideStrings (see page 6785) strings, each of which is terminated by the LineBreak value.

Note: In Windows, the default LineBreak value is a carriage return and line feed combination (#13#10), whereas on Linux, it is just a carriage return (#13).

See Also

GetText (see page 6799)

73.1.2.4.7 TWideStrings.Names Property

Indicates the name part of strings that are name-value pairs.

Pascal

```
property Names [Index: Integer]: WideString;
```

C++

```
__property BSTR Names[int Index];
```

Description

When the list of strings for the TWideStrings (see page 6785) object includes strings that are name-value pairs, read Names to access the name part of a string. Names is the name part of the string at Index, where 0 is the first string, 1 is the second string, and so on. If the string is not a name-value pair, Names contains an empty string.

See Also

NameValueSeparator (see page 6811)

IndexOfName (see page 6800)

Strings (see page 6813)

Values (see page 6816)

73.1.2.4.8 TWideStrings.NameValueSeparator Property

Indicates the character used to separate names from values.

Pascal

```
property NameValueSeparator: WideChar;
```

C++

```
__property WideChar NameValueSeparator;
```

Description

Strings (see page 6813) that contain the NameValueSeparator character are considered name-value pairs. NameValueSeparator defaults to the equal sign (=). TWideStrings (see page 6785) defines various methods for accessing names and values and for searching for specific names.

Strings (see page 6813) that are name-value pairs consist of a name part, the separator character, and a value part. Any spaces around the separator character are part of the name or value. This convention corresponds to the format used in many initialization files. For example:

```
DisplayGrid=1
SnapToGrid=1
GridSizeX=8
GridSizeY=8
```

See Also

ValueFromIndex (see page 6815)

Names (see page 6810)

IndexOfName (see page 6800)

Strings (see page 6813)

Values (see page 6816)

73.1.2.4.9 TWideStrings.Objects Property

Represents a set of objects that are associated one with each of the strings in the Strings (see page 6813) property.

Pascal

```
property Objects [Index: Integer]: TObject;
```

C++

```
__property TObject * Objects[int Index];
```

Description

Setting the Objects property for TWideStrings (see page 6785) does nothing. Reading the Objects property for TWideStrings (see page 6785) returns nil (Delphi) or NULL (C++). Descendant classes can associate objects with the strings in the set by implementing the Objects property.

Use the Objects property of a descendant of TWideStrings (see page 6785) to get or set the object associated with the string at the position indicated by Index. Index gives the position of the string associated with the object, where 0 is the first string, 1 is the second string, and so on. If a descendant of TWideStrings (see page 6785) does not support the Objects property, reading this property returns nil (Delphi) or NULL (C++)

Note: The TWideStrings (see page 6785) object does not own the objects in the Objects array. Objects added to the Objects array still exist even if the TWideStrings (see page 6785) object is destroyed. They must be explicitly destroyed by the application.

See Also

AddObject (see page 6790)

IndexOfObject (see page 6800)

InsertObject (see page 6801)

Strings (see page 6813)

73.1.2.4.10 TWideStrings.QuoteChar Property

Specifies the quote character used by the DelimitedText (see page 6809) property.

Pascal

```
property QuoteChar: WideChar;
```

C++

```
__property WideChar QuoteChar;
```

Description

Use QuoteChar to get or set the quote character that is used to enclose individual strings in the DelimitedText (see page 6809) property.

See Also

Delimiter (see page 6810)

DelimitedText (see page 6809)

CommaText (see page 6807)

73.1.2.4.11 TWideStrings.StrictDelimiter Property

Determines how the Delimiter (see page 6810) property is used.

Pascal

```
property StrictDelimiter: Boolean;
```

C++

```
__property Boolean StrictDelimiter;
```

Description

Use this property to specify whether the Delimiter (see page 6810) is the only value used within the DelimitedText (see page 6809) property. If set to True, individual strings in DelimitedText (see page 6809) are separated only by the character that is the value of Delimiter (see page 6810) If set to False, individual strings in DelimitedText (see page 6809) can be separated by a space, a non-printable character, or the character that is the value of Delimiter (see page 6810).

73.1.2.4.12 TWideStrings.Strings Property

References the strings in the list by their positions.

Pascal

```
property Strings [Index: Integer]: WideString;
```

C++

```
__property BSTR Strings[int Index];
```

Description

Descendants of TWideStrings (see page 6785) must implement an accessor function for the Strings property to return the string at the position indicated by Index. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

Use the Strings property to get or set the string at a particular position.

To store a string as a name-value pair, assign Strings a value that includes the NameValueSeparator (see page 6811) character. The name and value will then be accessible separately using the Names (see page 6810) and Values (see page 6816) properties.

Note: In Delphi, Strings is the default property of TWideStrings (see page 6785) objects. The Strings identifier can be omitted when accessing the Strings property of a descendant of TWideStrings (see page 6785). For example, the following two lines of code are both acceptable and do the same thing:

```
MyStrings.Strings[0] := 'This is the first string';
        MyStrings[0] := 'This is the first string';
```

See Also

ValueFromIndex (see page 6815)

NameValueSeparator (see page 6811)

Names (see page 6810)

Values (see page 6816)

CommaText (see page 6807)

IndexOf (see page 6800)

Names (see page 6810)

Objects (see page 6811)

Text (see page 6814)

Values (see page 6816)

73.1.2.4.13 TWideStrings.StringsAdapter Property

Implements an IWideStringsAdapter (see page 6819) interface for the TWideStrings (see page 6785) object.

Pascal

```
property StringsAdapter: IWideStringsAdapter;
```

C++

```
__property IWideStringsAdapter StringsAdapter;
```

Description

Do not use the StringsAdapter property. StringsAdapter is used internally.

See Also

TStringsAdapter

73.1.2.4.14 TWideStrings.Text Property

Lists the strings in the TWideStrings (see page 6785) object as a single string with the individual strings delimited by carriage returns and line feeds.

Pascal

```
property Text: WideString;
```

C++

```
__property BSTR Text;
```

Description

Use Text to get or set all the strings in the TWideStrings (see page 6785) object in a single string delimited by carriage return, line feed pairs.

When reading Text, the strings in the list will be separated by carriage return and (on Windows) line feed. If any of the strings in the list contain a carriage return (and line feed), the resulting value of Text will appear to contain more strings than is indicated by the Count (see page 6809) property.

When setting Text, the value will be parsed by separating into substrings whenever a carriage return or linefeed is encountered. (The two do not need to form pairs).

If the strings in the list contain carriage return or linefeed characters, a less ambiguous format for the strings is available through the CommaText (see page 6807) or DelimitedText (see page 6809) property.

See Also

Add (see page 6789)

CommaText (see page 6807)

Count (see page 6809)

Strings (see page 6813)

GetText (see page 6799)

SetText (see page 6805)

DelimitedText (see page 6809)

73.1.2.4.15 TWideStrings.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 6792) that have not been matched by a call to EndUpdate (see page 6795).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

TWideStrings (see page 6785) uses UpdateCount to keep track of calls to the BeginUpdate (see page 6792) and EndUpdate (see page 6795) methods. Every time a call is made to BeginUpdate (see page 6792), TWideStrings (see page 6785) increments the value of UpdateCount. Every call to EndUpdate (see page 6795) causes TWideStrings (see page 6785) to decrement UpdateCount.

When UpdateCount changes from 0 to 1, TWideStrings (see page 6785) calls the SetUpdateState (see page 6806) method with a parameter of true. When UpdateCount changes from 1 to 0, TStrings calls the SetUpdateState (see page 6806) method with a parameter of false. This allows descendant classes to perform optimizations when handling multiple updates.

See Also

BeginUpdate (see page 6792)

EndUpdate (see page 6795)

SetUpdateState (see page 6806)

73.1.2.4.16 TWideStrings.ValueFromIndex Property

Represents the value part of a string with a given index, on strings that are name-value pairs.

Pascal

```
property ValueFromIndex [Index: Integer]: WideString;
```

C++

```
__property BSTR ValueFromIndex[int Index];
```

Description

When the list of strings for the TWideStrings (see page 6785) object includes strings that are name-value pairs, use ValueFromIndex to get or set the value part of a string associated with an index.

For more information on name-value pairs, refer to the NameValueSeparator (see page 6811) property.

See Also

NameValueSeparator (see page 6811)

IndexOfName (see page 6800)

Strings (see page 6813)

Values (see page 6816)

73.1.2.4.17 TWideStrings.Values Property

Represents the value part of a string associated with a given name, on strings that are name-value pairs.

Pascal

```
property Values [const Name: WideString]: WideString;
```

C++

```
__property BSTR Values[BSTR const Name];
```

Description

When the list of strings for the TWideStrings (see page 6785) object includes strings that are name-value pairs, use Values to get or set the value part of a string associated with a specific name part.

For more information on name-value pairs, refer to the NameValueSeparator (see page 6811) property.

Note: The Name index is case-insensitive. That is, Values is the value part for the first occurrence of Name or an equivalent string that differs only in case.

See Also

ValueFromIndex (see page 6815)

NameValueSeparator (see page 6811)

IndexOfName (see page 6800)

Names (see page 6810)

Strings (see page 6813)

73.1.3 TWideStringsEnumerator Class

Enumerates through the strings of a TWideStrings (see page 6785) object.

Class Hierarchy

Pascal

```
TWideStringsEnumerator = class;
```

C++

```
class TWideStringsEnumerator;
```

File

WideStrings

Description

TWideStringsEnumerator provides an enumerator for TWideStrings (see page 6785) objects.

73.1.3.1 TWideStringsEnumerator Members

The following tables list the members exposed by TWideStringsEnumerator.

Public Methods

	Name	Description
	Create (see page 6817)	Creates a TWideStrings (see page 6785) enumerator.
	GetCurrent (see page 6817)	Gets the current string.
	MoveNext (see page 6817)	Moves the enumerator to the next string in the TWideStrings (see page 6785) collection.

Public Properties

	Name	Description
	Current (see page 6818)	The current string.

73.1.3.2 TWideStringsEnumerator.Create Constructor

Creates a TWideStrings (see page 6785) enumerator.

Pascal

```
constructor Create(AStrings: TWideStrings);
```

C++

```
__fastcall TWideStringsEnumerator(TWideStrings AStrings);
```

Description

Creates an instance of TWideStringsEnumerator (see page 6816).

73.1.3.3 TWideStringsEnumerator Methods

The methods of the TWideStringsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 6817)	Gets the current string.
	MoveNext (see page 6817)	Moves the enumerator to the next string in the TWideStrings (see page 6785) collection.

73.1.3.3.1 TWideStringsEnumerator.GetCurrent Method

Gets the current string.

Pascal

```
function GetCurrent: WideString;
```

C++

```
__fastcall BSTR GetCurrent();
```

Description

Use GetCurrent to get the current string in the TWideStrings (see page 6785) object being enumerated.

73.1.3.3.2 TWideStringsEnumerator.MoveNext Method

Moves the enumerator to the next string in the TWideStrings (see page 6785) collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TWideStringsEnumerator (see page 6816) to the next string in the TWideStrings (see page 6785) object being enumerated.

73.1.3.4 TWideStringsEnumerator Properties

The properties of the TWideStringsEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 6818)	The current string.

73.1.3.4.1 TWideStringsEnumerator.Current Property

The current string.

Pascal

```
property Current: WideString;
```

C++

```
__property BSTR Current;
```

Description

Current contains the current string in the TWideStrings (see page 6785) object being enumerated.

73.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWideStringsAdapter (see page 6819)	This is class WideStrings.IWideStringsAdapter.

73.2.1 IWideStringsAdapter Interface

Class Hierarchy

WideStrings.IWideStringsAdapter

Pascal

```
IWideStringsAdapter = interface;
```

C++

```
__interface IWideStringsAdapter;
```

File

WideStrings

Description

This is class WideStrings.IWideStringsAdapter.

73.2.1.1 IWideStringsAdapter Members

The following tables list the members exposed by IWideStringsAdapter.

Public Methods

	Name	Description
	ReferenceStrings (see page 6820)	This is ReferenceStrings, a member of class IWideStringsAdapter.
	ReleaseStrings (see page 6820)	This is ReleaseStrings, a member of class IWideStringsAdapter.

73.2.1.2 IWideStringsAdapter Methods

The methods of the IWideStringsAdapter class are listed here.

Public Methods

	Name	Description
	ReferenceStrings (see page 6820)	This is ReferenceStrings, a member of class IWideStringsAdapter.
	ReleaseStrings (see page 6820)	This is ReleaseStrings, a member of class IWideStringsAdapter.

73.2.1.2.1 IWideStringsAdapter.ReferenceStrings Method

Pascal

```
procedure ReferenceStrings(S: TWideStrings);
```

C++

```
__fastcall ReferenceStrings(TWideStrings S);
```

Description

This is ReferenceStrings, a member of class IWideStringsAdapter.

73.2.1.2.2 IWideStringsAdapter.ReleaseStrings Method

Pascal

```
procedure ReleaseStrings;
```

C++

```
__fastcall ReleaseStrings();
```

Description

This is ReleaseStrings, a member of class IWideStringsAdapter.

73.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TWideStringItem (see page 6821)	TWideStringItem defines a TWideStringList (see page 6768) entry.

73.3.1 WideStrings.TWideStringItem Record

TWideStringItem defines a TWideStringList (see page 6768) entry.

Pascal

```
TWideStringItem = record
  FString: WideString;
  FObject: TObject;
end;
```

C++

```
struct TWideStringItem {
  BSTR FString;
  TObject * FObject;
};
```

File

WideStrings

Description

The TWideStringItem type defines a data structure used internally in Delphi.

It is used by TWideStringList (see page 6768) to hold a list entry.

FString defines an Unicode string for a list entry.

FObject points to an optional TObject (see page 3364) associated with the string.

See Also

TWideStringList (see page 6768)

73.4 Types

The following table lists types in this documentation.

Types

Name	Description
PWideStringItem (see page 6822)	PWideStringItem is a TWideStringItem (see page 6821) pointer.
PWideStringItemList (see page 6822)	PWideStringItemList is a TWideStringItemList (see page 6823) pointer.
TWideStringItemList (see page 6823)	TWideStringItemList defines an array of TWideStringItems.
TWideStringListSortCompare (see page 6823)	TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6768) class.

73.4.1 WideStrings.PWideStringItem Type

PWideStringItem is a TWideStringItem (see page 6821) pointer.

Pascal

```
PWideStringItem = ^TWideStringItem;
```

C++

```
^TWideStringItem PWideStringItem;
```

File

WideStrings

Description

PWideStringItem is used internally by TWideStringList (see page 6768) to point to a TWideStringItem (see page 6821) value.

TWideStringItem (see page 6821) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional TObject (see page 3364) associated with the string.

See Also

TWideStringItem (see page 6821)

TWideStringList (see page 6768)

73.4.2 WideStrings.PWideStringItemList Type

PWideStringItemList is a TWideStringItemList (see page 6823) pointer.

Pascal

```
PWideStringItemList = ^TWideStringItemList;
```

C++

```
^TWideStringItemList PWideStringItemList;
```

File

WideStrings

Description

PWideStringItemList is used internally by TWideStringList (see page 6768) to refer to its internal array of TWideStringItem (see page 6821) values. This array holds the list of strings.

TWideStringItem (see page 6821) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional TObject (see page 3364) associated with the string.

See Also

TWideStringItem (see page 6821)

TWideStringList (see page 6768)

73.4.3 WideStrings.TWideStringItemList Type

TWideStringItemList defines an array of TWideStringItems.

Pascal

```
TWideStringItemList = array[0..MaxListSize] of TWideStringItem;
```

C++

```
array[0..MaxListSize] of TWideStringItem TWideStringItemList;
```

File

WideStrings

Description

TWideStringItemList is used internally by TWideStringList (see page 6768) to refer to its internal array of TWideStringItem (see page 6821) values. This array holds the list of strings.

TWideStringItem (see page 6821) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional TObject (see page 3364) associated with the string.

See Also

TWideStringItem (see page 6821)

TWideStringList (see page 6768)

73.4.4 WideStrings.TWideStringListSortCompare Type

TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6768) class.

Pascal

```
TWideStringListSortCompare = function (List: TWideStringList; Index1, Index2: Integer):
```

```
Integer;
```

C++

```
(List: TWideStringList; Index1, Index2: Integer): Integer ( TWideStringListSortCompare)();
```

File

WideStrings

Description

You define a TWideStringListSortCompare function when you need to use the TWideStringList (see page 6768) CustomSort method. The procedure defined by TWideStringListSortCompare is invoked by CustomSort to sort the TWideStringList (see page 6768) elements into sequence.

CustomSort is used instead of Sort when the sort sequence required differs from the default alphanumeric sequence.

List is the list of Unicode strings.

Index1 and **Index2** are indexes into the list. When these are passed to the TWideListSortCompare function, the CustomSort method is asking which order they should be in.

You return 0 if the entry referred to by **Index1** equals the entry referred to by **Index2**

You return less than 0 if the entry referred to by **Index1** is less than the entry referred to by **Index2**

You return a number above 0 if the entry referred to by **Index1** is greater than the entry referred to by **Index2**

Note: One of the principle advantages of the CustomSort method is that a TWideStringList (see page 6768) may be sorted according to the objects optionally attachable to each string.

See Also

CustomSort (see page 6775)

TWideStringList (see page 6768)

74 WideStrUtils Namespace

This is namespace WideStrUtils.

Topics

Name	Description
Functions (see page 6827)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6842)	The following table lists structs, records, enums in this documentation.
Types (see page 6843)	The following table lists types in this documentation.
Constants (see page 6844)	The following table lists constants in this documentation.

Constants

Name	Description
sUTF8BOMString (see page 6844)	This is constant WideStrUtils.sUTF8BOMString.

Functions

	Name	Description
	AnsiToUtf8Ex (see page 6828)	This is function WideStrUtils.AnsiToUtf8Ex.
	ConvertStreamFromAnsiToUTF8 (see page 6828)	This is function WideStrUtils.ConvertStreamFromAnsiToUTF8.
	ConvertStreamFromUTF8ToAnsi (see page 6828)	This is function WideStrUtils.ConvertStreamFromUTF8ToAnsi.
	DetectUTF8Encoding (see page 6829)	This is function WideStrUtils.DetectUTF8Encoding.
	HasExtendCharacter (see page 6829)	This is function WideStrUtils.HasExtendCharacter.
	HasUTF8BOM (see page 6829)	This is function WideStrUtils.HasUTF8BOM.
	HasUTF8BOM (see page 6830)	This is function WideStrUtils.HasUTF8BOM.
	inOpArray (see page 6830)	This is function WideStrUtils.inOpArray.
	inOpSet (see page 6830)	This is function WideStrUtils.inOpSet.
	IsUTF8LeadByte (see page 6831)	This is function WideStrUtils.IsUTF8LeadByte.
	IsUTF8String (see page 6831)	This is function WideStrUtils.IsUTF8String.
	IsUTF8TrailByte (see page 6831)	This is function WideStrUtils.IsUTF8TrailByte.
	LoadResWideString (see page 6832)	This is function WideStrUtils.LoadResWideString.
	LoadWideStr (see page 6832)	This is function WideStrUtils.LoadWideStr.
	UTF8CharLength (see page 6832)	This is function WideStrUtils.UTF8CharLength.
	UTF8CharSize (see page 6833)	This is function WideStrUtils.UTF8CharSize.
	UTF8LowerCase (see page 6833)	This is function WideStrUtils.UTF8LowerCase.
	Utf8ToAnsiEx (see page 6833)	This is function WideStrUtils.Utf8ToAnsiEx.
	UTF8UpperCase (see page 6834)	This is function WideStrUtils.UTF8UpperCase.
	WideAdjustLineBreaks (see page 6834)	This is function WideStrUtils.WideAdjustLineBreaks.

	WideDequotedStr (see page 6834)	This is function WideStrUtils.WideDequotedStr.
	WideExtractQuotedStr (see page 6835)	This is function WideStrUtils.WideExtractQuotedStr.
	WideLastChar (see page 6835)	This is function WideStrUtils.WideLastChar.
	WideQuotedStr (see page 6835)	This is function WideStrUtils.WideQuotedStr.
	WideReplaceStr (see page 6836)	This is function WideStrUtils.WideReplaceStr.
	WideReplaceText (see page 6836)	This is function WideStrUtils.WideReplaceText.
	WideStringReplace (see page 6836)	This is function WideStrUtils.WideStringReplace.
	WStrAlloc (see page 6837)	This is function WideStrUtils.WStrAlloc.
	WStrBufSize (see page 6837)	This is function WideStrUtils.WStrBufSize.
	WStrCat (see page 6837)	This is function WideStrUtils.WStrCat.
	WStrComp (see page 6838)	This is function WideStrUtils.WStrComp.
	WStrCopy (see page 6838)	This is function WideStrUtils.WStrCopy.
	WStrDispose (see page 6838)	This is function WideStrUtils.WStrDispose.
	WStrEnd (see page 6839)	This is function WideStrUtils.WStrEnd.
	WStrLCopy (see page 6839)	This is function WideStrUtils.WStrLCopy.
	WStrLen (see page 6839)	This is function WideStrUtils.WStrLen.
	WStrMove (see page 6840)	This is function WideStrUtils.WStrMove.
	WStrNew (see page 6840)	This is function WideStrUtils.WStrNew.
	WStrPCopy (see page 6840)	This is function WideStrUtils.WStrPCopy.
	WStrPLCopy (see page 6841)	This is function WideStrUtils.WStrPLCopy.
	WStrPos (see page 6841)	This is function WideStrUtils.WStrPos.
	WStrScan (see page 6841)	This is function WideStrUtils.WStrScan.

Structs, Records, Enums

	Name	Description
	TEncodeType (see page 6842)	This is record WideStrUtils.TEncodeType.

Types

Name	Description
CharSet (see page 6843)	This is type WideStrUtils.CharSet.

74.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiToUtf8Ex (see page 6828)	This is function WideStrUtils.AnsiToUtf8Ex.
	ConvertStreamFromAnsiToUTF8 (see page 6828)	This is function WideStrUtils.ConvertStreamFromAnsiToUTF8.
	ConvertStreamFromUTF8ToAnsi (see page 6828)	This is function WideStrUtils.ConvertStreamFromUTF8ToAnsi.
	DetectUTF8Encoding (see page 6829)	This is function WideStrUtils.DetectUTF8Encoding.
	HasExtendCharacter (see page 6829)	This is function WideStrUtils.HasExtendCharacter.
	HasUTF8BOM (see page 6829)	This is function WideStrUtils.HasUTF8BOM.
	HasUTF8BOM (see page 6830)	This is function WideStrUtils.HasUTF8BOM.
	inOpArray (see page 6830)	This is function WideStrUtils.inOpArray.
	inOpSet (see page 6830)	This is function WideStrUtils.inOpSet.
	IsUTF8LeadByte (see page 6831)	This is function WideStrUtils.IsUTF8LeadByte.
	IsUTF8String (see page 6831)	This is function WideStrUtils.IsUTF8String.
	IsUTF8TrailByte (see page 6831)	This is function WideStrUtils.IsUTF8TrailByte.
	LoadResWideString (see page 6832)	This is function WideStrUtils.LoadResWideString.
	LoadWideStr (see page 6832)	This is function WideStrUtils.LoadWideStr.
	UTF8CharLength (see page 6832)	This is function WideStrUtils.UTF8CharLength.
	UTF8CharSize (see page 6833)	This is function WideStrUtils.UTF8CharSize.
	UTF8LowerCase (see page 6833)	This is function WideStrUtils.UTF8LowerCase.
	Utf8ToAnsiEx (see page 6833)	This is function WideStrUtils.Utf8ToAnsiEx.
	UTF8UpperCase (see page 6834)	This is function WideStrUtils.UTF8UpperCase.
	WideAdjustLineBreaks (see page 6834)	This is function WideStrUtils.WideAdjustLineBreaks.
	WideDequotedStr (see page 6834)	This is function WideStrUtils.WideDequotedStr.
	WideExtractQuotedStr (see page 6835)	This is function WideStrUtils.WideExtractQuotedStr.
	WideLastChar (see page 6835)	This is function WideStrUtils.WideLastChar.
	WideQuotedStr (see page 6835)	This is function WideStrUtils.WideQuotedStr.
	WideReplaceStr (see page 6836)	This is function WideStrUtils.WideReplaceStr.
	WideReplaceText (see page 6836)	This is function WideStrUtils.WideReplaceText.
	WideStringReplace (see page 6836)	This is function WideStrUtils.WideStringReplace.
	WStrAlloc (see page 6837)	This is function WideStrUtils.WStrAlloc.
	WStrBufSize (see page 6837)	This is function WideStrUtils.WStrBufSize.
	WStrCat (see page 6837)	This is function WideStrUtils.WStrCat.
	WStrComp (see page 6838)	This is function WideStrUtils.WStrComp.
	WStrCopy (see page 6838)	This is function WideStrUtils.WStrCopy.
	WStrDispose (see page 6838)	This is function WideStrUtils.WStrDispose.
	WStrEnd (see page 6839)	This is function WideStrUtils.WStrEnd.
	WStrLCopy (see page 6839)	This is function WideStrUtils.WStrLCopy.
	WStrLen (see page 6839)	This is function WideStrUtils.WStrLen.
	WStrMove (see page 6840)	This is function WideStrUtils.WStrMove.
	WStrNew (see page 6840)	This is function WideStrUtils.WStrNew.
	WStrPCopy (see page 6840)	This is function WideStrUtils.WStrPCopy.
	WStrPLCopy (see page 6841)	This is function WideStrUtils.WStrPLCopy.
	WStrPos (see page 6841)	This is function WideStrUtils.WStrPos.
	WStrScan (see page 6841)	This is function WideStrUtils.WStrScan.

74.1.1 WideStrUtils.AnsiToUtf8Ex Function

Pascal

```
function AnsiToUtf8Ex(const S: AnsiString; const cp: integer): UTF8String;
```

C++

```
UTF8String AnsiToUtf8Ex(const AnsiString S, const int cp);
```

File

WideStrUtils

Description

This is function WideStrUtils.AnsiToUtf8Ex.

74.1.2 WideStrUtils.ConvertStreamFromAnsiToUTF8 Function

Pascal

```
procedure ConvertStreamFromAnsiToUTF8(Src: TStream; Dst: TStream; cp: integer = CP_ACP);
```

C++

```
ConvertStreamFromAnsiToUTF8(TStream Src, TStream Dst, int cp = CP_ACP);
```

File

WideStrUtils

Description

This is function WideStrUtils.ConvertStreamFromAnsiToUTF8.

74.1.3 WideStrUtils.ConvertStreamFromUTF8ToAnsi Function

Pascal

```
procedure ConvertStreamFromUTF8ToAnsi(Src: TStream; Dst: TStream; cp: integer = CP_ACP);
```

C++

```
ConvertStreamFromUTF8ToAnsi(TStream Src, TStream Dst, int cp = CP_ACP);
```

File

WideStrUtils

Description

This is function WideStrUtils.ConvertStreamFromUTF8ToAnsi.

74.1.4 WideStrUtils.DetectUTF8Encoding Function

Pascal

```
function DetectUTF8Encoding(const s: RawByteString): TEncodeType;
```

C++

```
TEncodeType DetectUTF8Encoding(const RawByteString s);
```

File

WideStrUtils

Description

This is function WideStrUtils.DetectUTF8Encoding.

74.1.5 WideStrUtils.HasExtendCharacter Function

Pascal

```
function HasExtendCharacter(const s: RawByteString): Boolean;
```

C++

```
Boolean HasExtendCharacter(const RawByteString s);
```

File

WideStrUtils

Description

This is function WideStrUtils.HasExtendCharacter.

74.1.6 WideStrUtils.HasUTF8BOM Function

Pascal

```
function HasUTF8BOM(const S: RawByteString): boolean; overload;
```

C++

```
boolean HasUTF8BOM(const RawByteString S);
```

File

WideStrUtils

Description

This is function WideStrUtils.HasUTF8BOM.

74.1.7 WideStrUtils.HasUTF8BOM Function

Pascal

```
function HasUTF8BOM(S: TStream): boolean; overload;
```

C++

```
boolean HasUTF8BOM(TStream S);
```

File

WideStrUtils

Description

This is function WideStrUtils.HasUTF8BOM.

74.1.8 WideStrUtils.inOpArray Function

Pascal

```
function inOpArray(W: WideChar; sets: array of WideChar): boolean;
```

C++

```
boolean inOpArray(WideChar W, array of WideChar sets);
```

File

WideStrUtils

Description

This is function WideStrUtils.inOpArray.

74.1.9 WideStrUtils.inOpSet Function

Pascal

```
function inOpSet(W: WideChar; sets: CharSet): boolean;
```

C++

```
boolean inOpSet(WideChar W, CharSet sets);
```

File

WideStrUtils

Description

This is function WideStrUtils.inOpSet.

74.1.10 WideStrUtils.IsUTF8LeadByte Function

Pascal

```
function IsUTF8LeadByte(Lead: AnsiChar): Boolean;
```

C++

```
Boolean IsUTF8LeadByte(AnsiChar Lead);
```

File

WideStrUtils

Description

This is function WideStrUtils.IsUTF8LeadByte.

74.1.11 WideStrUtils.IsUTF8String Function

Pascal

```
function IsUTF8String(const s: RawByteString): Boolean;
```

C++

```
Boolean IsUTF8String(const RawByteString s);
```

File

WideStrUtils

Description

This is function WideStrUtils.IsUTF8String.

74.1.12 WideStrUtils.IsUTF8TrailByte Function

Pascal

```
function IsUTF8TrailByte(Lead: AnsiChar): Boolean;
```

C++

```
Boolean IsUTF8TrailByte(AnsiChar Lead);
```

File

WideStrUtils

Description

This is function WideStrUtils.IsUTF8TrailByte.

74.1.13 WideStrUtils.LoadResWideString Function

Pascal

```
function LoadResWideString(ResStringRec: PResStringRec): WideString;
```

C++

```
BSTR LoadResWideString(PResStringRec ResStringRec);
```

File

WideStrUtils

Description

This is function WideStrUtils.LoadResWideString.

74.1.14 WideStrUtils.LoadWideStr Function

Pascal

```
function LoadWideStr(Ident: Integer): WideString;
```

C++

```
BSTR LoadWideStr(int Ident);
```

File

WideStrUtils

Description

This is function WideStrUtils.LoadWideStr.

74.1.15 WideStrUtils.UTF8CharLength Function

Pascal

```
function UTF8CharLength(Lead: AnsiChar): Integer;
```

C++

```
int UTF8CharLength(AnsiChar Lead);
```

File

WideStrUtils

Description

This is function WideStrUtils.UTF8CharLength.

74.1.16 WideStrUtils.UTF8CharSize Function

Pascal

```
function UTF8CharSize(Lead: AnsiChar): Integer;
```

C++

```
int UTF8CharSize(AnsiChar Lead);
```

File

WideStrUtils

Description

This is function WideStrUtils.UTF8CharSize.

74.1.17 WideStrUtils.UTF8LowerCase Function

Pascal

```
function UTF8LowerCase(const S: UTF8string): UTF8string;
```

C++

```
UTF8string UTF8LowerCase(const UTF8string S);
```

File

WideStrUtils

Description

This is function WideStrUtils.UTF8LowerCase.

74.1.18 WideStrUtils.Utf8ToAnsiEx Function

Pascal

```
function Utf8ToAnsiEx(const S: UTF8String; const cp: integer): AnsiString;
```

C++

```
AnsiString Utf8ToAnsiEx(const UTF8String S, const int cp);
```

File

WideStrUtils

Description

This is function WideStrUtils.Utf8ToAnsiEx.

74.1.19 WideStrUtils.UTF8UpperCase Function

Pascal

```
function UTF8UpperCase(const S: UTF8string): UTF8string;
```

C++

```
UTF8string UTF8UpperCase(const UTF8string S);
```

File

WideStrUtils

Description

This is function WideStrUtils.UTF8UpperCase.

74.1.20 WideStrUtils.WideAdjustLineBreaks Function

Pascal

```
function WideAdjustLineBreaks(const S: WideString; Style: TTextLineBreakStyle = tlbsCRLF):
WideString;
```

C++

```
BSTR WideAdjustLineBreaks(const BSTR S, TTextLineBreakStyle Style = tlbsCRLF);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideAdjustLineBreaks.

74.1.21 WideStrUtils.WideDequotedStr Function

Pascal

```
function WideDequotedStr(const S: WideString; AQuote: WideChar): WideString;
```

C++

```
BSTR WideDequotedStr(const BSTR S, WideChar AQuote);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideDequotedStr.

74.1.22 WideStrUtils.WideExtractQuotedStr Function

Pascal

```
function WideExtractQuotedStr(var Src: PWideChar; Quote: WideChar): WideString;
```

C++

```
BSTR WideExtractQuotedStr(PWideChar Src, WideChar Quote);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideExtractQuotedStr.

74.1.23 WideStrUtils.WideLastChar Function

Pascal

```
function WideLastChar(const S: WideString): PWideChar;
```

C++

```
PWideChar WideLastChar(const BSTR S);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideLastChar.

74.1.24 WideStrUtils.WideQuotedStr Function

Pascal

```
function WideQuotedStr(const S: WideString; Quote: WideChar): WideString;
```

C++

```
BSTR WideQuotedStr(const BSTR S, WideChar Quote);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideQuotedStr.

74.1.25 WideStrUtils.WideReplaceStr Function

Pascal

```
function WideReplaceStr(const AText: WideString; const AFromText: WideString; const AToText:
WideString): WideString;
```

C++

```
BSTR WideReplaceStr(const BSTR AText, const BSTR AFromText, const BSTR AToText);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideReplaceStr.

74.1.26 WideStrUtils.WideReplaceText Function

Pascal

```
function WideReplaceText(const AText: WideString; const AFromText: WideString; const AToText:
WideString): WideString;
```

C++

```
BSTR WideReplaceText(const BSTR AText, const BSTR AFromText, const BSTR AToText);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideReplaceText.

74.1.27 WideStrUtils.WideStringReplace Function

Pascal

```
function WideStringReplace(const S: Widestring; const OldPattern: Widestring; const
NewPattern: Widestring; Flags: TReplaceFlags): Widestring;
```

C++

```
BSTR WideStringReplace(const BSTR S, const BSTR OldPattern, const BSTR NewPattern,
TReplaceFlags Flags);
```

File

WideStrUtils

Description

This is function WideStrUtils.WideStringReplace.

74.1.28 WideStrUtils.WStrAlloc Function

Pascal

```
function WStrAlloc(Size: Cardinal): PWideChar;
```

C++

```
PWideChar WStrAlloc(unsigned Size);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrAlloc.

74.1.29 WideStrUtils.WStrBufSize Function

Pascal

```
function WStrBufSize(const Str: PWideChar): Cardinal;
```

C++

```
unsigned WStrBufSize(const PWideChar Str);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrBufSize.

74.1.30 WideStrUtils.WStrCat Function

Pascal

```
function WStrCat(Dest: PWideChar; const Source: PWideChar): PWideChar;
```

C++

```
PWideChar WStrCat(PWideChar Dest, const PWideChar Source);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrCat.

74.1.31 WideStrUtils.WStrComp Function

Pascal

```
function WStrComp(const Str1: PWideChar; const Str2: PWideChar): Integer;
```

C++

```
int WStrComp(const PWideChar Str1, const PWideChar Str2);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrComp.

74.1.32 WideStrUtils.WStrCopy Function

Pascal

```
function WStrCopy(Dest: PWideChar; const Source: PWideChar): PWideChar;
```

C++

```
PWideChar WStrCopy(PWideChar Dest, const PWideChar Source);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrCopy.

74.1.33 WideStrUtils.WStrDispose Function

Pascal

```
procedure WStrDispose(Str: PWideChar);
```

C++

```
WStrDispose(PWideChar Str);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrDispose.

74.1.34 WideStrUtils.WStrEnd Function

Pascal

```
function WStrEnd(const Str: PWideChar): PWideChar;
```

C++

```
PWideChar WStrEnd(const PWideChar Str);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrEnd.

74.1.35 WideStrUtils.WStrLCopy Function

Pascal

```
function WStrLCopy(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
```

C++

```
PWideChar WStrLCopy(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrLCopy.

74.1.36 WideStrUtils.WStrLen Function

Pascal

```
function WStrLen(const Str: PWideChar): Cardinal;
```

C++

```
unsigned WStrLen(const PWideChar Str);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrLen.

74.1.37 WideStrUtils.WStrMove Function

Pascal

```
function WStrMove(Dest: PWideChar; const Source: PWideChar; Count: Cardinal): PWideChar;
```

C++

```
PWideChar WStrMove(PWideChar Dest, const PWideChar Source, unsigned Count);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrMove.

74.1.38 WideStrUtils.WStrNew Function

Pascal

```
function WStrNew(const Str: PWideChar): PWideChar;
```

C++

```
PWideChar WStrNew(const PWideChar Str);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrNew.

74.1.39 WideStrUtils.WStrPCopy Function

Pascal

```
function WStrPCopy(Dest: PWideChar; const Source: WideString): PWideChar;
```

C++

```
PWideChar WStrPCopy(PWideChar Dest, const BSTR Source);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrPCopy.

74.1.40 WideStrUtils.WStrPLCopy Function

Pascal

```
function WStrPLCopy(Dest: PWideChar; const Source: WideString; MaxLen: Cardinal): PWideChar;
```

C++

```
PWideChar WStrPLCopy(PWideChar Dest, const BSTR Source, unsigned MaxLen);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrPLCopy.

74.1.41 WideStrUtils.WStrPos Function

Pascal

```
function WStrPos(const Str1: PWideChar; const Str2: PWideChar): PWideChar;
```

C++

```
PWideChar WStrPos(const PWideChar Str1, const PWideChar Str2);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrPos.

74.1.42 WideStrUtils.WStrScan Function

Pascal

```
function WStrScan(Str: PWideChar; Chr: WideChar): PWideChar;
```

C++

```
PWideChar WStrScan(PWideChar Str, WideChar Chr);
```

File

WideStrUtils

Description

This is function WideStrUtils.WStrScan.

74.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEncodeType (see page 6842)	This is record WideStrUtils.TEncodeType.

74.2.1 WideStrUtils.TEncodeType Enumeration

Pascal

```
TEncodeType = (
  etUSASCII,
  etUTF8,
  etANSI
);
```

C++

```
enum TEncodeType {
  etUSASCII,
  etUTF8,
  etANSI
};
```

File

WideStrUtils

Description

This is record WideStrUtils.TEncodeType.

74.3 Types

The following table lists types in this documentation.

Types

Name	Description
CharSet (see page 6843)	This is type WideStrUtils.CharSet.

74.3.1 WideStrUtils.CharSet Type

Pascal

```
CharSet = set of ansichar;
```

C++

```
ansichar CharSet;
```

File

WideStrUtils

Description

This is type WideStrUtils.CharSet.

74.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sUTF8BOMString (see page 6844)	This is constant WideStrUtils.sUTF8BOMString.

74.4.1 WideStrUtils.sUTF8BOMString Constant

Pascal

```
sUTF8BOMString: array[1..3] of AnsiChar = (AnsiChar(#$EF), AnsiChar(#$BB), AnsiChar(#$BF));
```

C++

```
array[1..3] of AnsiChar sUTF8BOMString = (AnsiChar(#$EF), AnsiChar(#$BB), AnsiChar(#$BF));
```

File

WideStrUtils

Description

This is constant WideStrUtils.sUTF8BOMString.

75 WinHelpViewer Namespace

This is namespace WinHelpViewer.

Topics

Name	Description
Interfaces (see page 6846)	The following table lists interfaces in this documentation.
Variables (see page 6849)	The following table lists variables in this documentation.

Interfaces

	Name	Description
	IWinHelpTester (see page 6846)	This is class WinHelpViewer.IWinHelpTester.

Variables

Name	Description
ViewerName (see page 6849)	This is variable WinHelpViewer.ViewerName.
WinHelpTester (see page 6849)	This is variable WinHelpViewer.WinHelpTester.

75.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWinHelpTester (see page 6846)	This is class WinHelpViewer.IWinHelpTester.

75.1.1 IWinHelpTester Interface

Class Hierarchy

Pascal

```
IWinHelpTester = interface(IInterface);
```

C++

```
__interface IWinHelpTester : public IInterface;
```

File

WinHelpViewer

Description

This is class WinHelpViewer.IWinHelpTester.

75.1.1.1 IWinHelpTester Members

The following tables list the members exposed by IWinHelpTester.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	CanShowALink (see page 6847)	This is CanShowALink, a member of class IWinHelpTester.
	CanShowContext (see page 6847)	This is CanShowContext, a member of class IWinHelpTester.
	CanShowTopic (see page 6847)	This is CanShowTopic, a member of class IWinHelpTester.
	GetDefaultHelpFile (see page 6847)	This is GetDefaultHelpFile, a member of class IWinHelpTester.
	GetHelpPath (see page 6848)	This is GetHelpPath, a member of class IWinHelpTester.
	GetHelpStrings (see page 6848)	This is GetHelpStrings, a member of class IWinHelpTester.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

75.1.1.2 IWinHelpTester Methods

The methods of the IWinHelpTester class are listed here.

Public Methods

	Name	Description
	CanShowALink (see page 6847)	This is CanShowALink, a member of class IWinHelpTester.
	CanShowContext (see page 6847)	This is CanShowContext, a member of class IWinHelpTester.
	CanShowTopic (see page 6847)	This is CanShowTopic, a member of class IWinHelpTester.
	GetDefaultHelpFile (see page 6847)	This is GetDefaultHelpFile, a member of class IWinHelpTester.
	GetHelpPath (see page 6848)	This is GetHelpPath, a member of class IWinHelpTester.
	GetHelpStrings (see page 6848)	This is GetHelpStrings, a member of class IWinHelpTester.

75.1.1.2.1 IWinHelpTester.CanShowALink Method

Pascal

```
function CanShowALink(const ALink: String; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowALink(const UnicodeString ALink, const UnicodeString FileName);
```

Description

This is CanShowALink, a member of class IWinHelpTester.

75.1.1.2.2 IWinHelpTester.CanShowContext Method

Pascal

```
function CanShowContext(const Context: Integer; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowContext(const int Context, const UnicodeString FileName);
```

Description

This is CanShowContext, a member of class IWinHelpTester.

75.1.1.2.3 IWinHelpTester.CanShowTopic Method

Pascal

```
function CanShowTopic(const Topic: String; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowTopic(const UnicodeString Topic, const UnicodeString FileName);
```

Description

This is CanShowTopic, a member of class IWinHelpTester.

75.1.1.2.4 IWinHelpTester.GetDefaultHelpFile Method

Pascal

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
function GetDefaultHelpFile: String;
```

C++

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
__fastcall UnicodeString GetDefaultHelpFile();
```

Description

This is GetDefaultHelpFile, a member of class IWinHelpTester.

75.1.1.2.5 IWinHelpTester.GetHelpPath Method

Pascal

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
function GetHelpPath: String;
```

C++

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
__fastcall UnicodeString GetHelpPath();
```

Description

This is GetHelpPath, a member of class IWinHelpTester.

75.1.1.2.6 IWinHelpTester.GetHelpStrings Method

Pascal

```
function GetHelpStrings(const ALink: String): TStringList;
```

C++

```
__fastcall TStringList GetHelpStrings(const UnicodeString ALink);
```

Description

This is GetHelpStrings, a member of class IWinHelpTester.

75.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ViewerName (see page 6849)	This is variable WinHelpViewer.ViewerName.
WinHelpTester (see page 6849)	This is variable WinHelpViewer.WinHelpTester.

75.2.1 WinHelpViewer.ViewerName Variable

Pascal

```
ViewerName: String;
```

C++

```
UnicodeString ViewerName;
```

File

WinHelpViewer

Description

This is variable WinHelpViewer.ViewerName.

75.2.2 WinHelpViewer.WinHelpTester Variable

Pascal

```
WinHelpTester: IWinHelpTester;
```

C++

```
IWinHelpTester WinHelpTester;
```

File

WinHelpViewer

Description

This is variable WinHelpViewer.WinHelpTester.

76 WSDLBind Namespace

This is namespace WSDLBind.

Topics

Name	Description
Classes (see page 6854)	The following table lists classes in this documentation.
Interfaces (see page 7029)	The following table lists interfaces in this documentation.
Functions (see page 7122)	The following table lists functions in this documentation.
Constants (see page 7124)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBinding (see page 6854)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 6861)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 6869)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 6875)	This is class WSDLBind.TBindings.
	TDefinition (see page 6882)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 6890)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 6891)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 6897)	This is class WSDLBind.TDocumented.
	TFault (see page 6903)	This is class WSDLBind.TFault.
	TFaults (see page 6904)	This is class WSDLBind.TFaults.
	TImport (see page 6910)	This is class WSDLBind.TImport.
	TImports (see page 6916)	This is class WSDLBind.TImports.
	TMessage (see page 6923)	This is class WSDLBind.TMessage.
	TMessages (see page 6930)	This is class WSDLBind.TMessages.
	TOperation (see page 6936)	This is class WSDLBind.TOperation.
	TOperations (see page 6943)	This is class WSDLBind.TOperations.
	TParam (see page 6950)	This is class WSDLBind.TParam.
	TPart (see page 6957)	This is class WSDLBind.TPart.
	TParts (see page 6964)	This is class WSDLBind.TParts.
	TPort (see page 6970)	This is class WSDLBind.TPort.
	TPorts (see page 6977)	This is class WSDLBind.TPorts.
	TPortType (see page 6983)	This is class WSDLBind.TPortType.
	TPortTypes (see page 6990)	This is class WSDLBind.TPortTypes.
	TService (see page 6996)	This is class WSDLBind.TService.

	TServices (see page 7003)	This is class WSDLBind.TServices.
	TTypes (see page 7009)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 7015)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 7022)	This is class WSDLBind.TXMLSchemaDefs.

Constants

Name	Description
SBinding (see page 7124)	This is constant WSDLBind.SBinding.
SDefinitions (see page 7124)	This is constant WSDLBind.SDefinitions.
SFault (see page 7125)	This is constant WSDLBind.SFault.
SImport (see page 7125)	This is constant WSDLBind.SImport.
SInput (see page 7125)	This is constant WSDLBind.SInput.
SLocation (see page 7126)	This is constant WSDLBind.SLocation.
SMessage (see page 7126)	This is constant WSDLBind.SMessage.
SNamespace (see page 7126)	This is constant WSDLBind.SNamespace.
SOperation (see page 7127)	This is constant WSDLBind.SOperation.
SOutput (see page 7127)	This is constant WSDLBind.SOutput.
SPart (see page 7127)	This is constant WSDLBind.SPart.
SPort (see page 7128)	This is constant WSDLBind.SPort.
SPortType (see page 7128)	This is constant WSDLBind.SPortType.
SService (see page 7128)	This is constant WSDLBind.SService.
SType (see page 7129)	This is constant WSDLBind.SType.
STypes (see page 7129)	This is constant WSDLBind.STypes.

Functions

	Name	Description
	GetDefinition (see page 7122)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 7122)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 7122)	This is function WSDLBind.NewWSDLDoc.

Interfaces

	Name	Description
	IBinding (see page 7029)	This is class WSDLBind.IBinding.
	IBindingInput (see page 7033)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 7034)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 7038)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 7041)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 7042)	This is class WSDLBind.IBindings.
	IDefinition (see page 7045)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 7051)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 7052)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 7055)	This is class WSDLBind.IDocumented.
	IFault (see page 7057)	This is class WSDLBind.IFault.
	IFaults (see page 7058)	This is class WSDLBind.IFaults.
	IImport (see page 7061)	This is class WSDLBind.IImport.
	IImports (see page 7064)	This is class WSDLBind.IImports.
	IMessage (see page 7067)	This is class WSDLBind.IMessage.
	IMessages (see page 7071)	This is class WSDLBind.IMessages.
	IOperation (see page 7074)	This is class WSDLBind.IOperation.
	IOperations (see page 7078)	This is class WSDLBind.IOperations.
	IParam (see page 7082)	This is class WSDLBind.IParam.
	IPart (see page 7085)	This is class WSDLBind.IPart.
	IParts (see page 7090)	This is class WSDLBind.IParts.
	IPort (see page 7093)	This is class WSDLBind.IPort.
	IPorts (see page 7096)	This is class WSDLBind.IPorts.

	IPortType (see page 7099)	This is class WSDLBind.IPortType.
	IPortTypes (see page 7103)	This is class WSDLBind.IPortTypes.
	IService (see page 7106)	This is class WSDLBind.IService.
	IServices (see page 7109)	This is class WSDLBind.IServices.
	ITypes (see page 7112)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 7115)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 7119)	This is class WSDLBind.IXMLSchemaDefs.

76.1 Classes

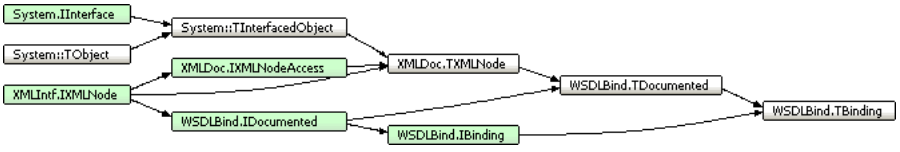
The following table lists classes in this documentation.

Classes

	Name	Description
	TBinding (see page 6854)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 6861)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 6869)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 6875)	This is class WSDLBind.TBindings.
	TDefinition (see page 6882)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 6890)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 6891)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 6897)	This is class WSDLBind.TDocumented.
	TFault (see page 6903)	This is class WSDLBind.TFault.
	TFaults (see page 6904)	This is class WSDLBind.TFaults.
	TImport (see page 6910)	This is class WSDLBind.TImport.
	TImports (see page 6916)	This is class WSDLBind.TImports.
	TMessage (see page 6923)	This is class WSDLBind.TMessage.
	TMessages (see page 6930)	This is class WSDLBind.TMessages.
	TOperation (see page 6936)	This is class WSDLBind.TOperation.
	TOperations (see page 6943)	This is class WSDLBind.TOperations.
	TParam (see page 6950)	This is class WSDLBind.TParam.
	TPart (see page 6957)	This is class WSDLBind.TPart.
	TParts (see page 6964)	This is class WSDLBind.TParts.
	TPort (see page 6970)	This is class WSDLBind.TPort.
	TPorts (see page 6977)	This is class WSDLBind.TPorts.
	TPortType (see page 6983)	This is class WSDLBind.TPortType.
	TPortTypes (see page 6990)	This is class WSDLBind.TPortTypes.
	TService (see page 6996)	This is class WSDLBind.TService.
	TServices (see page 7003)	This is class WSDLBind.TServices.
	TTypes (see page 7009)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 7015)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 7022)	This is class WSDLBind.TXMLSchemaDefs.

76.1.1 TBinding Class

Class Hierarchy



Pascal

```
TBinding = class(TDocumented, IBinding);
```

C++

```
class TBinding : public TDocumented, public IBinding;
```

File

WSDLBind

Description

This is class WSDLBind.TBinding.

76.1.1.1 TBinding Members

The following tables list the members exposed by TBinding.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.

	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperations (see page 6860)	This is GetBindingOperations, a member of class TBinding.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6860)	This is GetName, a member of class TBinding.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetType_ (see page 6861)	This is GetType_, a member of class TBinding.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6861)	This is SetName, a member of class TBinding.
	SetNodeValue (see page 7387)	Sets the value of this node.
	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	SetType_ (see page 6861)	This is SetType_, a member of class TBinding.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 6860)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.

	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.

	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	BindingOperations (see page 7033)	This is BindingOperations, a member of class IBinding.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7033)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	Type_ (see page 7033)	This is Type_, a member of class IBinding.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.1.2 TBinding Methods

The methods of the TBinding class are listed here.

Protected Methods

	Name	Description
	GetBindingOperations (see page 6860)	This is GetBindingOperations, a member of class TBinding.
	GetName (see page 6860)	This is GetName, a member of class TBinding.
	GetType_ (see page 6861)	This is GetType_, a member of class TBinding.
	SetName (see page 6861)	This is SetName, a member of class TBinding.
	SetType_ (see page 6861)	This is SetType_, a member of class TBinding.

Public Methods

	Name	Description
	AfterConstruction (see page 6860)	Responds after the last constructor executes.

76.1.1.2.1 TBinding.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.1.2.2 TBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class TBinding.

76.1.1.2.3 TBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBinding.

76.1.1.2.4 TBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TBinding.

76.1.1.2.5 TBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBinding.

76.1.1.2.6 TBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

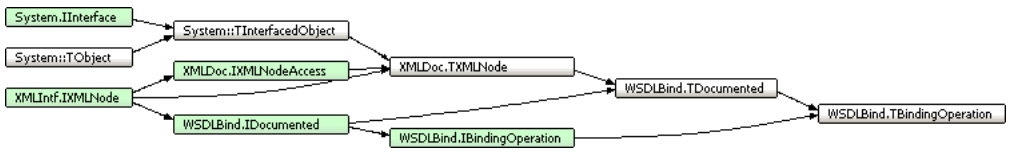
```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TBinding.

76.1.2 TBindingOperation Class

Class Hierarchy



Pascal

```
TBindingOperation = class(TDocumented, IBindingOperation);
```

C++

```
class TBindingOperation : public TDocumented, public IBindingOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperation.

76.1.2.1 TBindingOperation Members

The following tables list the members exposed by TBindingOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.

	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetFaults (see page 6867)	This is GetFaults, a member of class TBindingOperation.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetInput (see page 6868)	This is GetInput, a member of class TBindingOperation.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6868)	This is GetName, a member of class TBindingOperation.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOutput (see page 6868)	This is GetOutput, a member of class TBindingOperation.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.

	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6868)	This is SetName, a member of class TBindingOperation.
	SetNodeValue (see page 7387)	Sets the value of this node.
	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 6867)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.

	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.

	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Faults (see page 7037)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 7037)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7038)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Output (see page 7038)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.2.2 TBindingOperation Methods

The methods of the TBindingOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 6867)	This is GetFaults, a member of class TBindingOperation.
	GetInput (see page 6868)	This is GetInput, a member of class TBindingOperation.
	GetName (see page 6868)	This is GetName, a member of class TBindingOperation.
	GetOutput (see page 6868)	This is GetOutput, a member of class TBindingOperation.
	SetName (see page 6868)	This is SetName, a member of class TBindingOperation.

Public Methods

	Name	Description
	AfterConstruction (see page 6867)	Responds after the last constructor executes.

76.1.2.2.1 TBindingOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.2.2.2 TBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TBindingOperation.

76.1.2.2.3 TBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class TBindingOperation.

76.1.2.2.4 TBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBindingOperation.

76.1.2.2.5 TBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class TBindingOperation.

76.1.2.2.6 TBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

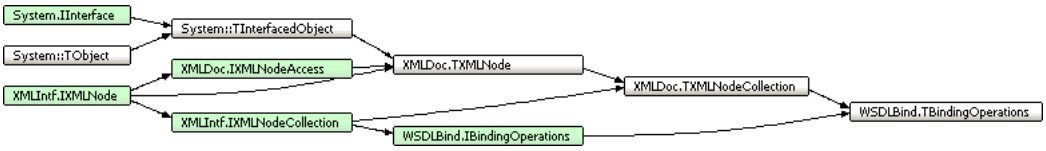
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBindingOperation.

76.1.3 TBindingOperations Class

Class Hierarchy



Pascal

```
TBindingOperations = class(TXMLNodeCollection, IBindingOperations);
```

C++

```
class TBindingOperations : public TXMLNodeCollection, public IBindingOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperations.

76.1.3.1 TBindingOperations Members

The following tables list the members exposed by TBindingOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6874)	This is Add, a member of class TBindingOperations.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperation (see page 6875)	This is GetBindingOperation, a member of class TBindingOperations.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6875)	This is Insert, a member of class TBindingOperations.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	BindingOperation (see page 7041)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.3.2 TBindingOperations Methods

The methods of the TBindingOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 6874)	This is Add, a member of class TBindingOperations.
	GetBindingOperation (see page 6875)	This is GetBindingOperation, a member of class TBindingOperations.
	Insert (see page 6875)	This is Insert, a member of class TBindingOperations.

76.1.3.2.1 TBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TBindingOperations.

76.1.3.2.2 TBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class TBindingOperations.

76.1.3.2.3 TBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

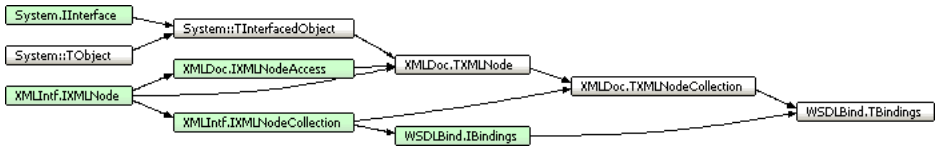
```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class TBindingOperations.

76.1.4 TBindings Class

Class Hierarchy



Pascal

```
TBindings = class(TXMLNodeCollection, IBindings);
```

C++

```
class TBindings : public TXMLNodeCollection, public IBindings;
```

File

WSDLBind

Description

This is class WSDLBind.TBindings.

76.1.4.1 TBindings Members

The following tables list the members exposed by TBindings.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6881)	This is Add, a member of class TBindings.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 6881)	This is GetBinding, a member of class TBindings.

	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6881)	This is Insert, a member of class TBindings.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.

	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.

	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.

	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	Binding (see page 7045)	This is Binding, a member of class IBindings.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.

	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.4.2 TBindings Methods

The methods of the TBindings class are listed here.

Protected Methods

	Name	Description
	Add (see page 6881)	This is Add, a member of class TBindings.
	GetBinding (see page 6881)	This is GetBinding, a member of class TBindings.
	Insert (see page 6881)	This is Insert, a member of class TBindings.

76.1.4.2.1 TBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class TBindings.

76.1.4.2.2 TBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class TBindings.

76.1.4.2.3 TBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

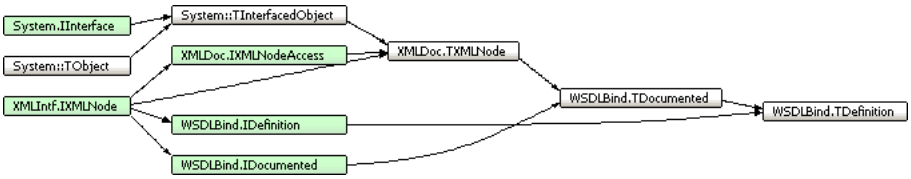
```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class TBindings.

76.1.5 TDefinition Class

Class Hierarchy



Pascal

```
TDefinition = class(TDocumented, IDefinition);
```

C++

```
class TDefinition : public TDocumented, public IDefinition;
```

File

WSDLBind

Description

This is class WSDLBind.TDefinition.

76.1.5.1 TDefinition Members

The following tables list the members exposed by TDefinition.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.

	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindings (see page 6888)	This is GetBindings, a member of class TDefinition.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetImports (see page 6888)	This is GetImports, a member of class TDefinition.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 6888)	This is GetMessages, a member of class TDefinition.
	GetName (see page 6889)	This is GetName, a member of class TDefinition.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPortTypes (see page 6889)	This is GetPortTypes, a member of class TDefinition.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetServices (see page 6889)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 6889)	This is GetTargetNamespace, a member of class TDefinition.
	GetText (see page 7378)	Returns the text value of the node.
	GetTypes (see page 6889)	This is GetTypes, a member of class TDefinition.

	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6890)	This is SetName, a member of class TDefinition.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 6890)	This is SetTargetNamespace, a member of class TDefinition.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6887)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.

	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.

	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	Bindings (see page 7050)	This is Bindings, a member of class IDefinition.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.

	Imports (see page 7050)	This is Imports, a member of class IDefinition.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Messages (see page 7050)	This is Messages, a member of class IDefinition.
	Name (see page 7050)	This is Name, a member of class IDefinition.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	PortTypes (see page 7050)	This is PortTypes, a member of class IDefinition.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Services (see page 7051)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7051)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 7643)	Represents the Text of the node.
	Types (see page 7051)	This is Types, a member of class IDefinition.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.5.2 TDefinition Methods

The methods of the TDefinition class are listed here.

Protected Methods

	Name	Description
	GetBindings (see page 6888)	This is GetBindings, a member of class TDefinition.
	GetImports (see page 6888)	This is GetImports, a member of class TDefinition.
	GetMessages (see page 6888)	This is GetMessages, a member of class TDefinition.
	GetName (see page 6889)	This is GetName, a member of class TDefinition.
	GetPortTypes (see page 6889)	This is GetPortTypes, a member of class TDefinition.
	GetServices (see page 6889)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 6889)	This is GetTargetNamespace, a member of class TDefinition.
	GetTypes (see page 6889)	This is GetTypes, a member of class TDefinition.
	SetName (see page 6890)	This is SetName, a member of class TDefinition.
	SetTargetNamespace (see page 6890)	This is SetTargetNamespace, a member of class TDefinition.

Public Methods

	Name	Description
	AfterConstruction (see page 6887)	Responds after the last constructor executes.

76.1.5.2.1 TDefinition.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.5.2.2 TDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class TDefinition.

76.1.5.2.3 TDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class TDefinition.

76.1.5.2.4 TDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class TDefinition.

76.1.5.2.5 TDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TDefinition.

76.1.5.2.6 TDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class TDefinition.

76.1.5.2.7 TDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class TDefinition.

76.1.5.2.8 TDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class TDefinition.

76.1.5.2.9 TDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class TDefinition.

76.1.5.2.10 TDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TDefinition.

76.1.5.2.11 TDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

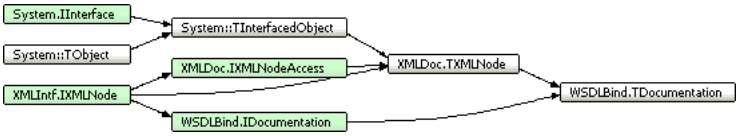
```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class TDefinition.

76.1.6 TDocumentation Class

Class Hierarchy



Pascal

```
TDocumentation = class(TXMLNode, IDocumentation);
```

C++

```
class TDocumentation : public TXMLNode, public IDocumentation;
```

File

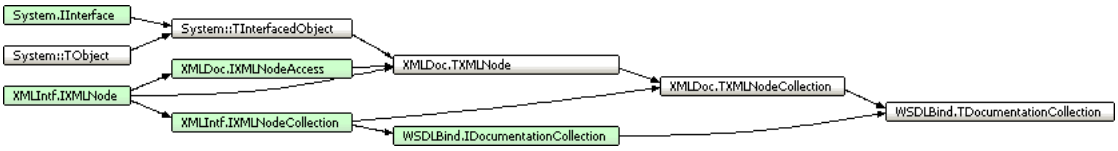
WSDLBind

Description

This is class WSDLBind.TDocumentation.

76.1.7 TDocumentationCollection Class

Class Hierarchy



Pascal

```
TDocumentationCollection = class(TXMLNodeCollection, IDocumentationCollection);
```

C++

```
class TDocumentationCollection : public TXMLNodeCollection, public IDocumentationCollection;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumentationCollection.

76.1.7.1 TDocumentationCollection Members

The following tables list the members exposed by TDocumentationCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6896)	This is Add, a member of class TDocumentationCollection.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDocumentation (see page 6897)	This is GetDocumentation, a member of class TDocumentationCollection.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6897)	This is Insert, a member of class TDocumentationCollection.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	Documentation (see page 7055)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.7.2 TDocumentationCollection Methods

The methods of the TDocumentationCollection class are listed here.

Protected Methods

	Name	Description
	Add (see page 6896)	This is Add, a member of class TDocumentationCollection.
	GetDocumentation (see page 6897)	This is GetDocumentation, a member of class TDocumentationCollection.
	Insert (see page 6897)	This is Insert, a member of class TDocumentationCollection.

76.1.7.2.1 TDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class TDocumentationCollection.

76.1.7.2.2 TDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class TDocumentationCollection.

76.1.7.2.3 TDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

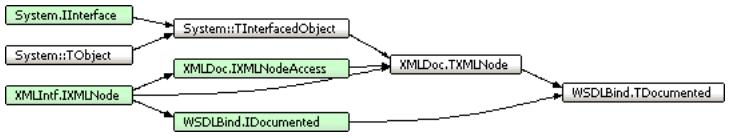
```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class TDocumentationCollection.

76.1.8 TDocumented Class

Class Hierarchy



Pascal

```
TDocumented = class(TXMLNode, IDocumented);
```

C++

```
class TDocumented : public TXMLNode, public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumented.

76.1.8.1 TDocumented Members

The following tables list the members exposed by TDocumented.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.

R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6902)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.

	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
	Collection (see page 7636)	Indicates the collection in which this node appears.
	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.8.2 TDocumented Methods

The methods of the TDocumented class are listed here.

Protected Methods

	Name	Description
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.

Public Methods

	Name	Description
	AfterConstruction (see page 6902)	Responds after the last constructor executes.

76.1.8.2.1 TDocumented.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is

incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.8.2.2 TDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

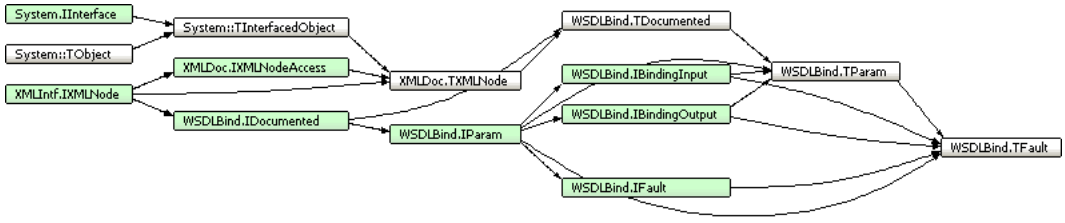
```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class TDocumented.

76.1.9 TFault Class

Class Hierarchy



Pascal

```
TFault = class(TParam, IParam, IFault, IBindingInput, IBindingOutput);
```

C++

```
class TFault : public TParam, public IParam, public IFault, public IBindingInput, public
IBindingOutput;
```

File

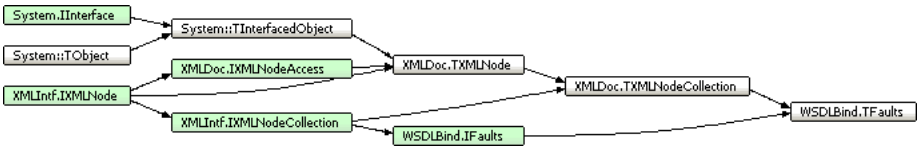
WSDLBind

Description

This is class WSDLBind.TFault.

76.1.10 TFaults Class

Class Hierarchy



Pascal

```
TFaults = class(TXMLNodeCollection, IFaults);
```

C++

```
class TFaults : public TXMLNodeCollection, public IFaults;
```

File

WSDLBind

Description

This is class WSDLBind.TFaults.

76.1.10.1 TFaults Members

The following tables list the members exposed by TFaults.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6909)	This is Add, a member of class TFaults.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetFault (see page 6910)	This is GetFault, a member of class TFaults.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6910)	This is Insert, a member of class TFaults.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Fault (see page 7061)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.10.2 TFaults Methods

The methods of the TFaults class are listed here.

Protected Methods

	Name	Description
	Add (see page 6909)	This is Add, a member of class TFaults.
	GetFault (see page 6910)	This is GetFault, a member of class TFaults.
	Insert (see page 6910)	This is Insert, a member of class TFaults.

76.1.10.2.1 TFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class TFaults.

76.1.10.2.2 TFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class TFaults.

76.1.10.2.3 TFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

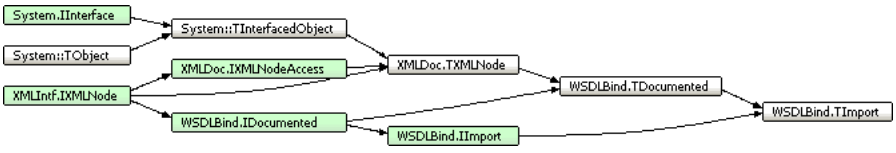
```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class TFaults.

76.1.11 TImport Class

Class Hierarchy



Pascal

```
TImport = class(TDocumented, IImport);
```

C++

```
class TImport : public TDocumented, public IImport;
```

File

WSDLBind

Description

This is class WSDLBind.TImport.

76.1.11.1 TImport Members

The following tables list the members exposed by TImport.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 6915)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 6916)	This is GetNamespace, a member of class TImport.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetLocation (see page 6916)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 6916)	This is SetNamespace, a member of class TImport.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.

R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 6902)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.

	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.

	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Location (see page 7064)	This is Location, a member of class IImport.
	Namespace (see page 7064)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.11.2 TImport Methods

The methods of the TImport class are listed here.

Protected Methods

	Name	Description
	GetLocation (see page 6915)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 6916)	This is GetNamespace, a member of class TImport.
	SetLocation (see page 6916)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 6916)	This is SetNamespace, a member of class TImport.

76.1.11.2.1 TImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class TImport.

76.1.11.2.2 TImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class TImport.

76.1.11.2.3 TImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class TImport.

76.1.11.2.4 TImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

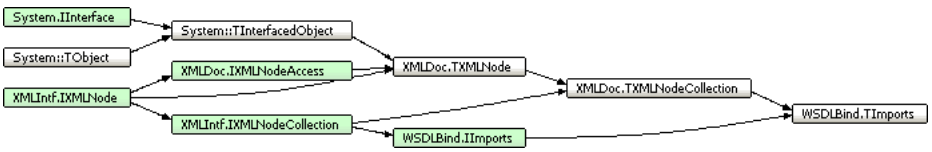
```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class TImport.

76.1.12 TImports Class

Class Hierarchy



Pascal

```
TImports = class(TXMLNodeCollection, IImports);
```

C++

```
class TImports : public TXMLNodeCollection, public IImports;
```

File

WSDLBind

Description

This is class WSDLBind.TImports.

76.1.12.1 TImports Members

The following tables list the members exposed by TImports.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6922)	This is Add, a member of class TImports.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.

	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetImport (see page 6922)	This is GetImport, a member of class TImports.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6923)	This is Insert, a member of class TImports.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.

	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.

	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.

	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.

	Collection (see page 7636)	Indicates the collection in which this node appears.
	Count (see page 7648)	Indicates the number of repeating child nodes.
	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
	Import (see page 7067)	This is Import, a member of class IImports.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.12.2 TImports Methods

The methods of the TImports class are listed here.

Protected Methods

	Name	Description
	Add (see page 6922)	This is Add, a member of class TImports.
	GetImport (see page 6922)	This is GetImport, a member of class TImports.
	Insert (see page 6923)	This is Insert, a member of class TImports.

76.1.12.2.1 TImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class TImports.

76.1.12.2.2 TImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class TImports.

76.1.12.2.3 TImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

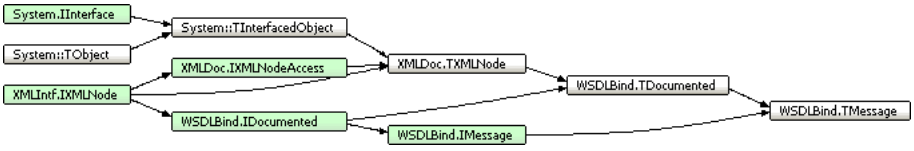
```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class TImports.

76.1.13 TMessage Class

Class Hierarchy



Pascal

```
TMessage = class(TDocumented, IMessage);
```

C++

```
class TMessage : public TDocumented, public IMessage;
```

File

WSDLBind

Description

This is class WSDLBind.TMessage.

76.1.13.1 TMessage Members

The following tables list the members exposed by TMessage.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6929)	This is GetName, a member of class TMessage.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.

	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetParts (see page 6929)	This is GetParts, a member of class TMessage.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6929)	This is SetName, a member of class TMessage.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6928)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.

	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.

	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7070)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Parts (see page 7070)	This is Parts, a member of class IMessage.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.13.2 TMessage Methods

The methods of the TMessage class are listed here.

Protected Methods

	Name	Description
	GetName (see page 6929)	This is GetName, a member of class TMessage.
	GetParts (see page 6929)	This is GetParts, a member of class TMessage.
	SetName (see page 6929)	This is SetName, a member of class TMessage.

Public Methods

	Name	Description
V	AfterConstruction (see page 6928)	Responds after the last constructor executes.

76.1.13.2.1 TMessage.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is

incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.13.2.2 TMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TMessage.

76.1.13.2.3 TMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class TMessage.

76.1.13.2.4 TMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

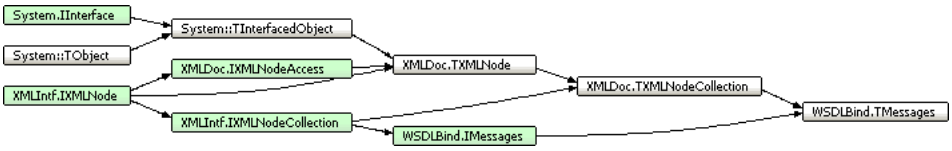
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TMessage.

76.1.14 TMessages Class

Class Hierarchy



Pascal

```
TMessages = class(TXMLNodeCollection, IMessages);
```

C++

```
class TMessages : public TXMLNodeCollection, public IMessages;
```

File

WSDLBind

Description

This is class WSDLBind.TMessages.

76.1.14.1 TMessages Members

The following tables list the members exposed by TMessages.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6935)	This is Add, a member of class TMessages.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 6936)	This is GetMessage, a member of class TMessages.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6936)	This is Insert, a member of class TMessages.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	Message (see page 7074)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.14.2 TMessages Methods

The methods of the TMessages class are listed here.

Protected Methods

	Name	Description
	Add (see page 6935)	This is Add, a member of class TMessages.
	GetMessage (see page 6936)	This is GetMessage, a member of class TMessages.
	Insert (see page 6936)	This is Insert, a member of class TMessages.

76.1.14.2.1 TMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class TMessages.

76.1.14.2.2 TMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class TMessages.

76.1.14.2.3 TMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

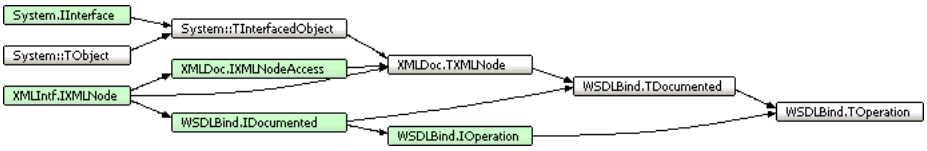
```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class TMessages.

76.1.15 TOperation Class

Class Hierarchy



Pascal

```
TOperation = class(TDocumented, IOperation);
```

C++

```
class TOperation : public TDocumented, public IOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TOperation.

76.1.15.1 TOperation Members

The following tables list the members exposed by TOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetFaults (see page 6942)	This is GetFaults, a member of class TOperation.

	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetInput (see page 6942)	This is GetInput, a member of class TOperation.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6942)	This is GetName, a member of class TOperation.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOutput (see page 6943)	This is GetOutput, a member of class TOperation.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6943)	This is SetName, a member of class TOperation.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.

R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6942)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.

	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Faults (see page 7077)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 7077)	This is Input, a member of class IOperation.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7078)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Output (see page 7078)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.15.2 TOperation Methods

The methods of the TOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 6942)	This is GetFaults, a member of class TOperation.
	GetInput (see page 6942)	This is GetInput, a member of class TOperation.
	GetName (see page 6942)	This is GetName, a member of class TOperation.
	GetOutput (see page 6943)	This is GetOutput, a member of class TOperation.
	SetName (see page 6943)	This is SetName, a member of class TOperation.

Public Methods

	Name	Description
V	AfterConstruction (see page 6942)	Responds after the last constructor executes.

76.1.15.2.1 TOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.15.2.2 TOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TOperation.

76.1.15.2.3 TOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class TOperation.

76.1.15.2.4 TOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TOperation.

76.1.15.2.5 TOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class TOperation.

76.1.15.2.6 TOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

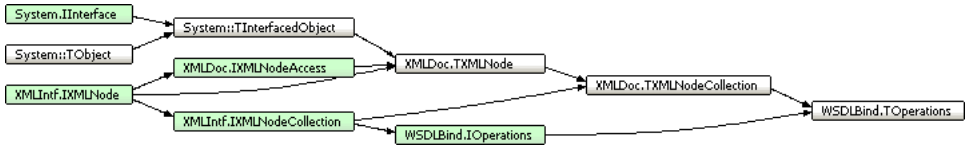
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TOperation.

76.1.16 TOperations Class

Class Hierarchy



Pascal

```
TOperations = class(TXMLNodeCollection, IOperations);
```

C++

```
class TOperations : public TXMLNodeCollection, public IOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TOperations.

76.1.16.1 TOperations Members

The following tables list the members exposed by TOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6949)	This is the overview for the Add method overload.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.

	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOperation (see page 6950)	This is GetOperation, a member of class TOperations.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6950)	This is Insert, a member of class TOperations.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.

	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
	Count (see page 7409)	Indicates the number of repeating child nodes.
	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
	List (see page 7411)	Lists the interfaces for the children that are repeated elements.
	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	Add (see page 7080)	This is the overview for the Add method overload.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.

	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.

	NodeValue (see page 7641)	Specifies the value of this node.
R	Operation (see page 7081)	This is Operation, a member of class IOperations.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.16.2 TOperations Methods

The methods of the TOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 6949)	This is the overview for the Add method overload.
	GetOperation (see page 6950)	This is GetOperation, a member of class TOperations.
	Insert (see page 6950)	This is Insert, a member of class TOperations.

76.1.16.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TOperations.Add (WideString) (see page 6949)	This is Add, a member of class TOperations.
	TOperations.Add (WideString, WideString, WideString, WideString) (see page 6949)	This is Add, a member of class TOperations.

76.1.16.2.1.1 TOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TOperations.

76.1.16.2.1.2 TOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class TOperations.

76.1.16.2.2 TOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class TOperations.

76.1.16.2.3 TOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

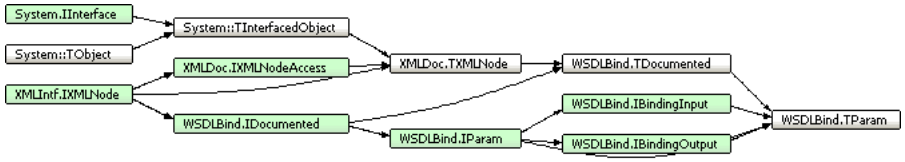
```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class TOperations.

76.1.17 TParam Class

Class Hierarchy



Pascal

```
TParam = class(TDocumented, IParam, IBindingInput, IBindingOutput);
```

C++

```
class TParam : public TDocumented, public IParam, public IBindingInput, public IBindingOutput;
```

File

WSDLBind

Description

This is class WSDLBind.TParam.

76.1.17.1 TParam Members

The following tables list the members exposed by TParam.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.

	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 6956)	This is GetMessage, a member of class TParam.
	GetName (see page 6956)	This is GetName, a member of class TParam.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetMessage (see page 6956)	This is SetMessage, a member of class TParam.
	SetName (see page 6956)	This is SetName, a member of class TParam.
	SetNodeValue (see page 7387)	Sets the value of this node.
	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	AfterConstruction (see page 6902)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.

	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Message (see page 7085)	This is Message, a member of class IParam.
	Name (see page 7085)	This is Name, a member of class IParam.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.17.2 TParam Methods

The methods of the TParam class are listed here.

Protected Methods

	Name	Description
	GetMessage (see page 6956)	This is GetMessage, a member of class TParam.
	GetName (see page 6956)	This is GetName, a member of class TParam.
	SetMessage (see page 6956)	This is SetMessage, a member of class TParam.
	SetName (see page 6956)	This is SetName, a member of class TParam.

76.1.17.2.1 TParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class TParam.

76.1.17.2.2 TParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TParam.

76.1.17.2.3 TParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class TParam.

76.1.17.2.4 TParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

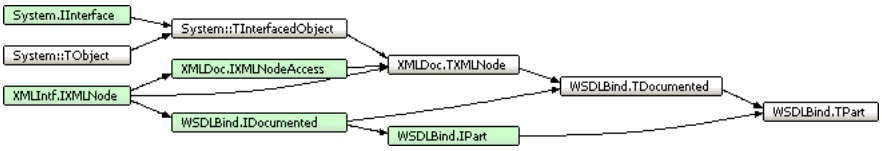
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TParam.

76.1.18 TPart Class

Class Hierarchy



Pascal

```
TPart = class(TDocumented, IPart);
```

C++

```
class TPart : public TDocumented, public IPart;
```

File

WSDLBind

Description

This is class WSDLBind.TPart.

76.1.18.1 TPart Members

The following tables list the members exposed by TPart.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.

	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetElement (see page 6962)	This is GetElement, a member of class TPart.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6962)	This is GetName, a member of class TPart.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetType_ (see page 6963)	This is GetType_, a member of class TPart.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.

	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetElement (see page 6963)	This is SetElement, a member of class TPart.
	SetName (see page 6963)	This is SetName, a member of class TPart.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	SetType_ (see page 6963)	This is SetType_, a member of class TPart.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 6902)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.

	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	Element (see page 7089)	This is Element, a member of class IPart.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7089)	This is Name, a member of class IPart.

	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	Type_ (see page 7089)	This is Type_, a member of class IPart.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.18.2 TPart Methods

The methods of the TPart class are listed here.

Protected Methods

	Name	Description
	GetElement (see page 6962)	This is GetElement, a member of class TPart.
	GetName (see page 6962)	This is GetName, a member of class TPart.
	GetType_ (see page 6963)	This is GetType_, a member of class TPart.
	SetElement (see page 6963)	This is SetElement, a member of class TPart.
	SetName (see page 6963)	This is SetName, a member of class TPart.
	SetType_ (see page 6963)	This is SetType_, a member of class TPart.

76.1.18.2.1 TPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class TPart.

76.1.18.2.2 TPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPart.

76.1.18.2.3 TPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TPart.

76.1.18.2.4 TPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class TPart.

76.1.18.2.5 TPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPart.

76.1.18.2.6 TPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

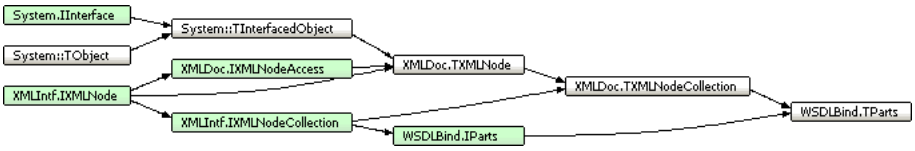
```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TPart.

76.1.19 TParts Class

Class Hierarchy



Pascal

```
TParts = class(TXMLNodeCollection, IParts);
```

C++

```
class TParts : public TXMLNodeCollection, public IParts;
```

File

WSDLBind

Description

This is class WSDLBind.TParts.

76.1.19.1 TParts Members

The following tables list the members exposed by TParts.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6969)	This is Add, a member of class TParts.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPart (see page 6970)	This is GetPart, a member of class TParts.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6970)	This is Insert, a member of class TParts.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Part (see page 7093)	This is Part, a member of class IParts.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.19.2 TParts Methods

The methods of the TParts class are listed here.

Protected Methods

	Name	Description
	Add (see page 6969)	This is Add, a member of class TParts.
	GetPart (see page 6970)	This is GetPart, a member of class TParts.
	Insert (see page 6970)	This is Insert, a member of class TParts.

76.1.19.2.1 TParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class TParts.

76.1.19.2.2 TParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class TParts.

76.1.19.2.3 TParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

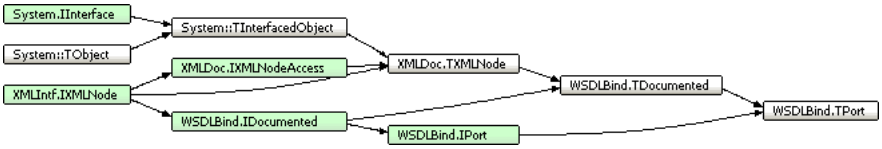
```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class TParts.

76.1.20 TPort Class

Class Hierarchy



Pascal

```
TPort = class(TDocumented, IPort);
```

C++

```
class TPort : public TDocumented, public IPort;
```

File

WSDLBind

Description

This is class WSDLBind.TPort.

76.1.20.1 TPort Members

The following tables list the members exposed by TPort.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 6976)	This is GetBinding, a member of class TPort.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.

	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6976)	This is GetName, a member of class TPort.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 6976)	This is SetBinding, a member of class TPort.
	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6976)	This is SetName, a member of class TPort.
	SetNodeValue (see page 7387)	Sets the value of this node.
	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3358)	Responds after the last constructor executes.
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	AfterConstruction (see page 6902)	Responds after the last constructor executes.
	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.

	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	Binding (see page 7096)	This is Binding, a member of class IPort.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7096)	This is Name, a member of class IPort.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.20.2 TPort Methods

The methods of the TPort class are listed here.

Protected Methods

	Name	Description
	GetBinding (see page 6976)	This is GetBinding, a member of class TPort.
	GetName (see page 6976)	This is GetName, a member of class TPort.
	SetBinding (see page 6976)	This is SetBinding, a member of class TPort.
	SetName (see page 6976)	This is SetName, a member of class TPort.

76.1.20.2.1 TPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class TPort.

76.1.20.2.2 TPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPort.

76.1.20.2.3 TPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class TPort.

76.1.20.2.4 TPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

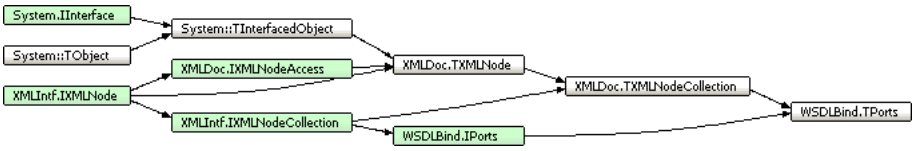
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPort.

76.1.21 TPorts Class

Class Hierarchy



Pascal

```
TPorts = class(TXMLNodeCollection, IPorts);
```

C++

```
class TPorts : public TXMLNodeCollection, public IPorts;
```

File

WSDLBind

Description

This is class WSDLBind.TPorts.

76.1.21.1 TPorts Members

The following tables list the members exposed by TPorts.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6982)	This is Add, a member of class TPorts.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPort (see page 6983)	This is GetPort, a member of class TPorts.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6983)	This is Insert, a member of class TPorts.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
	Collection (see page 7636)	Indicates the collection in which this node appears.
	Count (see page 7648)	Indicates the number of repeating child nodes.
	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Port (see page 7099)	This is Port, a member of class IPorts.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.21.2 TPorts Methods

The methods of the TPorts class are listed here.

Protected Methods

	Name	Description
	Add (see page 6982)	This is Add, a member of class TPorts.
	GetPort (see page 6983)	This is GetPort, a member of class TPorts.
	Insert (see page 6983)	This is Insert, a member of class TPorts.

76.1.21.2.1 TPorts.Add Method

Pascal

```
function Add(const Name: WideString; const Binding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR Binding);
```

Description

This is Add, a member of class TPorts.

76.1.21.2.2 TPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class TPorts.

76.1.21.2.3 TPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

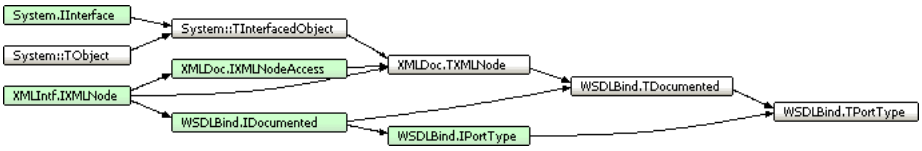
```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class TPorts.

76.1.22 TPortType Class

Class Hierarchy



Pascal

```
TPortType = class(TDocumented, IPortType);
```

C++

```
class TPortType : public TDocumented, public IPortType;
```

File

WSDLBind

Description

This is class WSDLBind.TPortType.

76.1.22.1 TPortType Members

The following tables list the members exposed by TPortType.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 6989)	This is GetName, a member of class TPortType.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOperations (see page 6989)	This is GetOperations, a member of class TPortType.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 6989)	This is SetName, a member of class TPortType.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.

	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6988)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.

	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.

	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
	Collection (see page 7636)	Indicates the collection in which this node appears.
	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7102)	This is Name, a member of class IPortType.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	Operations (see page 7102)	This is Operations, a member of class IPortType.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.22.2 TPortType Methods

The methods of the TPortType class are listed here.

Protected Methods

	Name	Description
	GetName (see page 6989)	This is GetName, a member of class TPortType.
	GetOperations (see page 6989)	This is GetOperations, a member of class TPortType.
	SetName (see page 6989)	This is SetName, a member of class TPortType.

Public Methods

	Name	Description
	AfterConstruction (see page 6988)	Responds after the last constructor executes.

76.1.22.2.1 TPortType.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.22.2.2 TPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPortType.

76.1.22.2.3 TPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class TPortType.

76.1.22.2.4 TPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

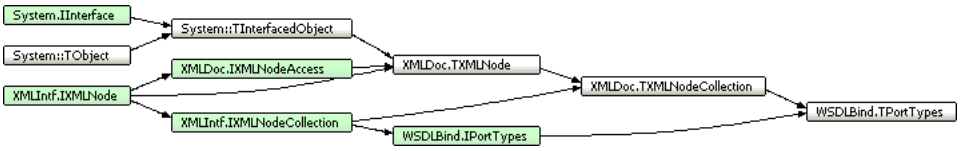
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPortType.

76.1.23 TPortTypes Class

Class Hierarchy



Pascal

```
TPortTypes = class(TXMLNodeCollection, IPortTypes);
```

C++

```
class TPortTypes : public TXMLNodeCollection, public IPortTypes;
```

File

WSDLBind

Description

This is class WSDLBind.TPortTypes.

76.1.23.1 TPortTypes Members

The following tables list the members exposed by TPortTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 6995)	This is Add, a member of class TPortTypes.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPortType (see page 6996)	This is GetPortType, a member of class TPortTypes.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 6996)	This is Insert, a member of class TPortTypes.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	PortType (see page 7106)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.23.2 TPortTypes Methods

The methods of the TPortTypes class are listed here.

Protected Methods

	Name	Description
	Add (see page 6995)	This is Add, a member of class TPortTypes.
	GetPortType (see page 6996)	This is GetPortType, a member of class TPortTypes.
	Insert (see page 6996)	This is Insert, a member of class TPortTypes.

76.1.23.2.1 TPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType; overload;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class TPortTypes.

76.1.23.2.2 TPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class TPortTypes.

76.1.23.2.3 TPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

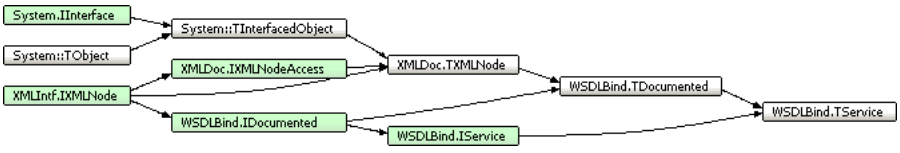
```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class TPortTypes.

76.1.24 TService Class

Class Hierarchy



Pascal

```
TService = class(TDocumented, IService);
```

C++

```
class TService : public TDocumented, public IService;
```

File

WSDLBind

Description

This is class WSDLBind.TService.

76.1.24.1 TService Members

The following tables list the members exposed by TService.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetName (see page 7002)	This is GetName, a member of class TService.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPorts (see page 7002)	This is GetPorts, a member of class TService.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetName (see page 7002)	This is SetName, a member of class TService.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.

	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7001)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.

	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.

	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
	Collection (see page 7636)	Indicates the collection in which this node appears.
	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7109)	This is Name, a member of class IService.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Ports (see page 7109)	This is Ports, a member of class IService.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.24.2 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	GetName (see page 7002)	This is GetName, a member of class TService.
	GetPorts (see page 7002)	This is GetPorts, a member of class TService.
	SetName (see page 7002)	This is SetName, a member of class TService.

Public Methods

	Name	Description
	AfterConstruction (see page 7001)	Responds after the last constructor executes.

76.1.24.2.1 TService.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.24.2.2 TService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TService.

76.1.24.2.3 TService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class TService.

76.1.24.2.4 TService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

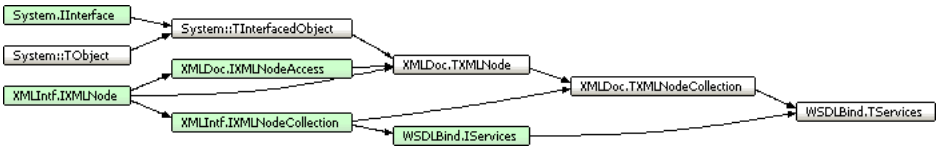
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TService.

76.1.25 TServices Class

Class Hierarchy



Pascal

```
TServices = class(TXMLNodeCollection, IServices);
```

C++

```
class TServices : public TXMLNodeCollection, public IServices;
```

File

WSDLBind

Description

This is class WSDLBind.TServices.

76.1.25.1 TServices Members

The following tables list the members exposed by TServices.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 7008)	This is Add, a member of class TServices.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetService (see page 7009)	This is GetService, a member of class TServices.

	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	Insert (see page 7009)	This is Insert, a member of class TServices.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.

	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
R	Service (see page 7112)	This is Service, a member of class IServices.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.25.2 TServices Methods

The methods of the TServices class are listed here.

Protected Methods

	Name	Description
	Add (see page 7008)	This is Add, a member of class TServices.
	GetService (see page 7009)	This is GetService, a member of class TServices.
	Insert (see page 7009)	This is Insert, a member of class TServices.

76.1.25.2.1 TServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class TServices.

76.1.25.2.2 TServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class TServices.

76.1.25.2.3 TServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

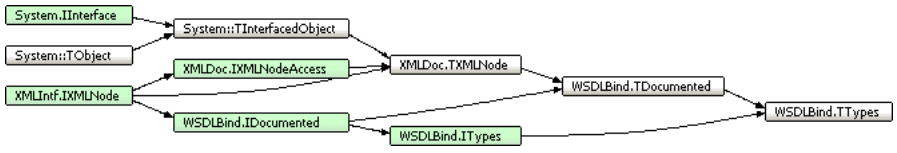
```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class TServices.

76.1.26 TTypes Class

Class Hierarchy



Pascal

```
TTypes = class(TDocumented, ITypes);
```

C++

```
class TTypes : public TDocumented, public ITypes;
```

File

WSDLBind

Description

This is class WSDLBind.TTypes.

76.1.26.1 TTypes Members

The following tables list the members exposed by TTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDocumentationCollection (see page 6903)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetSchemaDefs (see page 7015)	This is GetSchemaDefs, a member of class TTypes.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.

R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7014)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.

	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
R	SchemaDefs (see page 7114)	This is SchemaDefs, a member of class ITypes.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.26.2 TTypes Methods

The methods of the TTypes class are listed here.

Protected Methods

	Name	Description
	GetSchemaDefs (see page 7015)	This is GetSchemaDefs, a member of class TTypes.

Public Methods

	Name	Description
V	AfterConstruction (see page 7014)	Responds after the last constructor executes.

76.1.26.2.1 TTypes.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call

it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3358)

System::TInterfacedObject::NewInstance (see page 3358)

System::TInterfacedObject::RefCount (see page 3360)

76.1.26.2.2 TTypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

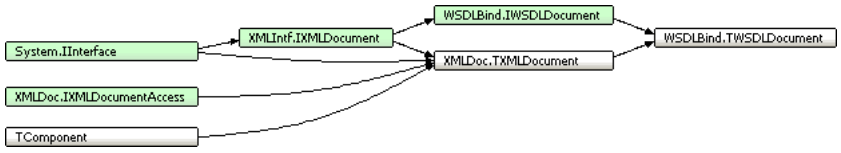
Description

This is GetSchemaDefs, a member of class TTypes.

76.1.27 TWSDLDocument Class

TWSDLDocument represents a WSDL document.

Class Hierarchy



Pascal

```
TWSDLDocument = class(TXMLDocument, IWSDLDocument);
```

C++

```
class TWSDLDocument : public TXMLDocument, public IWSDLDocument;
```

File

WSDLBind

Description

Use TWSDLDocument to represent a WSDL (Web Services Definition (see page 7022) Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements.

TWSDLDocument can read an existing WSDL document from a file, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLDocument uses an external DOM

(Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLDocument uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLDocument instance is automatically freed. When TWSDLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

Tip: Although you can work directly with TWSDLDocument to access a WSDL document, it is often easier to use TWSDLItems, a TWSDLDocument descendant. TWSDLItems implements a number of methods for getting at the definitions in a WSDL document.

See Also

TWSDLItems (see page 7163)

TWSDLView (see page 7223)

76.1.27.1 TWSDLDocument Members

The following tables list the members exposed by TWSDLDocument.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.
	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7313)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDefinition (see page 7021)	Returns the value of the Definition property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.

	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
V	LoadData (see page 7021)	This is LoadData, a member of class TWSDLDocument.
V	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 7327)	Sets the value of the Active property.
	SetDefinition (see page 7021)	Sets the value of the Definition property.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7020)	Initializes the TWSDLDocument (see page 7015) object after the last constructor has executed.
	Create (see page 7303)	This is the overview for the Create constructor overload.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
V	Destroy (see page 7304)	Disposes of a TXMLDocument (see page 7299) object.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.

	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.

	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.
	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
	Definition (see page 7022)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.
	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.
	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

76.1.27.2 TWSDLDocument Methods

The methods of the TWSDLDocument class are listed here.

Protected Methods

	Name	Description
	GetDefinition (see page 7021)	Returns the value of the Definition property.
	LoadData (see page 7021)	This is LoadData, a member of class TWSDLDocument.
	SetDefinition (see page 7021)	Sets the value of the Definition property.

Public Methods

	Name	Description
	AfterConstruction (see page 7020)	Initializes the TWSDLDocument (see page 7015) object after the last constructor has executed.

76.1.27.2.1 TWSDLDocument.AfterConstruction Method

Initializes the TWSDLDocument (see page 7015) object after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

The AfterConstruction method is called automatically after the last constructor has finished executing. TWSDLDocument (see page 7015) overrides this method to register an implementation class for the document element. That class implements the

IDefinition (see page 7045) interface that is the value of the Definition (see page 7022) property.

See Also

RegisterDocBinding (see page 7324)

Definition (see page 7022)

76.1.27.2.2 TWSDLDocument.GetDefinition Method

Returns the value of the Definition property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition property.

See Also

Definition (see page 7118)

76.1.27.2.3 TWSDLDocument.LoadData Method

Pascal

```
procedure LoadData; override;
```

C++

```
virtual __fastcall LoadData();
```

Description

This is LoadData, a member of class TWSDLDocument.

76.1.27.2.4 TWSDLDocument.SetDefinition Method

Sets the value of the Definition property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition property.

See Also

Definition (see page 7118)

76.1.27.3 TWSDLDocument Properties

The properties of the TWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 7022)	Provides access to the definitions contained in the WSDL document.

76.1.27.3.1 TWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 7045) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 7045) interface to access the Bindings, Messages, PortTypes, Services, and Types (see page 4683) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems (a TWSDLDocument (see page 7015) descendant) to access those definitions.

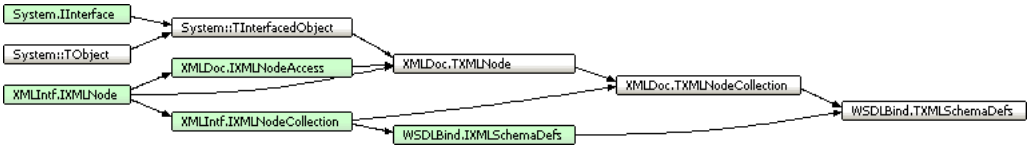
See Also

DocumentElement (see page 7335)

TWSDLItems (see page 7163)

76.1.28 TXMLSchemaDefs Class

Class Hierarchy



Pascal

```
TXMLSchemaDefs = class(TXMLNodeCollection, IXMLSchemaDefs);
```

C++

```
class TXMLSchemaDefs : public TXMLNodeCollection, public IXMLSchemaDefs;
```

File

WSDLBind

Description

This is class WSDLBind.TXMLSchemaDefs.

76.1.28.1 TXMLSchemaDefs Members

The following tables list the members exposed by TXMLSchemaDefs.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	Add (see page 7028)	This is Add, a member of class TXMLSchemaDefs.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.

	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	GetXMLSchemaDef (see page 7028)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.

	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.

	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.

	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.

	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	SchemaDef (see page 7121)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.1.28.2 TXMLSchemaDefs Methods

The methods of the TXMLSchemaDefs class are listed here.

Protected Methods

	Name	Description
	Add (see page 7028)	This is Add, a member of class TXMLSchemaDefs.
	GetXMLSchemaDef (see page 7028)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

76.1.28.2.1 TXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class TXMLSchemaDefs.

76.1.28.2.2 TXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

76.2 Interfaces

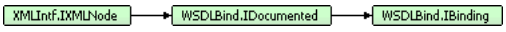
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IBinding (see page 7029)	This is class WSDLBind.IBinding.
	IBindingInput (see page 7033)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 7034)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 7038)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 7041)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 7042)	This is class WSDLBind.IBindings.
	IDefinition (see page 7045)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 7051)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 7052)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 7055)	This is class WSDLBind.IDocumented.
	IFault (see page 7057)	This is class WSDLBind.IFault.
	IFaults (see page 7058)	This is class WSDLBind.IFaults.
	IImport (see page 7061)	This is class WSDLBind.IImport.
	IImports (see page 7064)	This is class WSDLBind.IImports.
	IMessage (see page 7067)	This is class WSDLBind.IMessage.
	IMessages (see page 7071)	This is class WSDLBind.IMessages.
	IOperation (see page 7074)	This is class WSDLBind.IOperation.
	IOperations (see page 7078)	This is class WSDLBind.IOperations.
	IParam (see page 7082)	This is class WSDLBind.IParam.
	IPart (see page 7085)	This is class WSDLBind.IPart.
	IParts (see page 7090)	This is class WSDLBind.IParts.
	IPort (see page 7093)	This is class WSDLBind.IPort.
	IPorts (see page 7096)	This is class WSDLBind.IPorts.
	IPortType (see page 7099)	This is class WSDLBind.IPortType.
	IPortTypes (see page 7103)	This is class WSDLBind.IPortTypes.
	IService (see page 7106)	This is class WSDLBind.IService.
	IServices (see page 7109)	This is class WSDLBind.IServices.
	ITypes (see page 7112)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 7115)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 7119)	This is class WSDLBind.IXMLSchemaDefs.

76.2.1 IBinding Interface

Class Hierarchy



Pascal

```
IBinding = interface(IDocumented);
```

C++

```
__interface IBinding : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBinding.

76.2.1.1 IBinding Members

The following tables list the members exposed by IBinding.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetBindingOperations (see page 7031)	This is GetBindingOperations, a member of class IBinding.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7032)	This is GetName, a member of class IBinding.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetType_ (see page 7032)	This is GetType_, a member of class IBinding.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7032)	This is SetName, a member of class IBinding.

	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	SetType_ (see page 7032)	This is SetType_, a member of class IBinding.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	BindingOperations (see page 7033)	This is BindingOperations, a member of class IBinding.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7033)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
	Type_ (see page 7033)	This is Type_, a member of class IBinding.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.1.2 IBinding Methods

The methods of the IBinding class are listed here.

Public Methods

	Name	Description
	GetBindingOperations (see page 7031)	This is GetBindingOperations, a member of class IBinding.
	GetName (see page 7032)	This is GetName, a member of class IBinding.
	GetType_ (see page 7032)	This is GetType_, a member of class IBinding.
	SetName (see page 7032)	This is SetName, a member of class IBinding.
	SetType_ (see page 7032)	This is SetType_, a member of class IBinding.

76.2.1.2.1 IBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class IBinding.

76.2.1.2.2 IBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBinding.

76.2.1.2.3 IBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IBinding.

76.2.1.2.4 IBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBinding.

76.2.1.2.5 IBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IBinding.

76.2.1.3 IBinding Properties

The properties of the IBinding class are listed here.

Public Properties

	Name	Description
	BindingOperations (see page 7033)	This is BindingOperations, a member of class IBinding.
	Name (see page 7033)	This is Name, a member of class IBinding.
	Type_ (see page 7033)	This is Type_, a member of class IBinding.

76.2.1.3.1 IBinding.BindingOperations Property

Pascal

```
property BindingOperations: IBindingOperations;
```

C++

```
__property IBindingOperations BindingOperations;
```

Description

This is BindingOperations, a member of class IBinding.

76.2.1.3.2 IBinding.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBinding.

76.2.1.3.3 IBinding.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

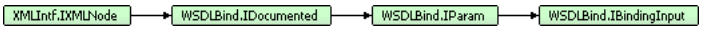
```
__property BSTR Type_;
```

Description

This is Type_, a member of class IBinding.

76.2.2 IBindingInput Interface

Class Hierarchy



Pascal

```
IBindingInput = interface(IParam);
```

C++

```
__interface IBindingInput : public IParam;
```

File

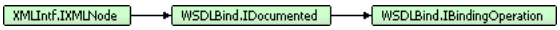
WSDLBind

Description

This is class WSDLBind.IBindingInput.

76.2.3 IBindingOperation Interface

Class Hierarchy



Pascal

```
IBindingOperation = interface(IDocumented);
```

C++

```
__interface IBindingOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperation.

76.2.3.1 IBindingOperation Members

The following tables list the members exposed by IBindingOperation.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetFaults (see page 7036)	This is GetFaults, a member of class IBindingOperation.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetInput (see page 7036)	This is GetInput, a member of class IBindingOperation.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7036)	This is GetName, a member of class IBindingOperation.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.

	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOutput (see page 7036)	This is GetOutput, a member of class IBindingOperation.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7037)	This is SetName, a member of class IBindingOperation.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Faults (see page 7037)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	Input (see page 7037)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7038)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Output (see page 7038)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.3.2 IBindingOperation Methods

The methods of the IBindingOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 7036)	This is GetFaults, a member of class IBindingOperation.
	GetInput (see page 7036)	This is GetInput, a member of class IBindingOperation.
	GetName (see page 7036)	This is GetName, a member of class IBindingOperation.
	GetOutput (see page 7036)	This is GetOutput, a member of class IBindingOperation.
	SetName (see page 7037)	This is SetName, a member of class IBindingOperation.

76.2.3.2.1 IBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IBindingOperation.

76.2.3.2.2 IBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class IBindingOperation.

76.2.3.2.3 IBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBindingOperation.

76.2.3.2.4 IBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class IBindingOperation.

76.2.3.2.5 IBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBindingOperation.

76.2.3.3 IBindingOperation Properties

The properties of the IBindingOperation class are listed here.

Public Properties

	Name	Description
	Faults (see page 7037)	This is Faults, a member of class IBindingOperation.
	Input (see page 7037)	This is Input, a member of class IBindingOperation.
	Name (see page 7038)	This is Name, a member of class IBindingOperation.
	Output (see page 7038)	This is Output, a member of class IBindingOperation.

76.2.3.3.1 IBindingOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IBindingOperation.

76.2.3.3.2 IBindingOperation.Input Property

Pascal

```
property Input: IBindingInput;
```

C++

```
__property IBindingInput Input;
```

Description

This is Input, a member of class IBindingOperation.

76.2.3.3.3 IBindingOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBindingOperation.

76.2.3.3.4 IBindingOperation.Output Property

Pascal

```
property Output: IBindingOutput;
```

C++

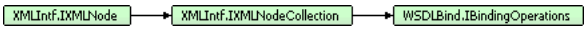
```
__property IBindingOutput Output;
```

Description

This is Output, a member of class IBindingOperation.

76.2.4 IBindingOperations Interface

Class Hierarchy



Pascal

```
IBindingOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IBindingOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperations.

76.2.4.1 IBindingOperations Members

The following tables list the members exposed by IBindingOperations.

Public Methods

	Name	Description
	Add (see page 7040)	This is Add, a member of class IBindingOperations.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.

	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetBindingOperation (see page 7040)	This is GetBindingOperation, a member of class IBindingOperations.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7041)	This is Insert, a member of class IBindingOperations.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	BindingOperation (see page 7041)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.

R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.4.2 IBindingOperations Methods

The methods of the IBindingOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 7040)	This is Add, a member of class IBindingOperations.
	GetBindingOperation (see page 7040)	This is GetBindingOperation, a member of class IBindingOperations.
	Insert (see page 7041)	This is Insert, a member of class IBindingOperations.

76.2.4.2.1 IBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IBindingOperations.

76.2.4.2.2 IBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class IBindingOperations.

76.2.4.2.3 IBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class IBindingOperations.

76.2.4.3 IBindingOperations Properties

The properties of the IBindingOperations class are listed here.

Public Properties

	Name	Description
	BindingOperation (see page 7041)	This is BindingOperation, a member of class IBindingOperations.

76.2.4.3.1 IBindingOperations.BindingOperation Property

Pascal

```
property BindingOperation [Index: Integer]: IBindingOperation;
```

C++

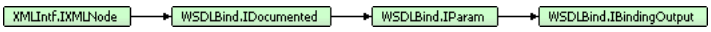
```
__property IBindingOperation BindingOperation[int Index];
```

Description

This is BindingOperation, a member of class IBindingOperations.

76.2.5 IBindingOutput Interface

Class Hierarchy



Pascal

```
IBindingOutput = interface(IParam);
```

C++

```
__interface IBindingOutput : public IParam;
```

File

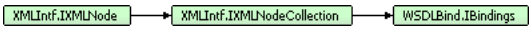
WSDLBind

Description

This is class WSDLBind.IBindingOutput.

76.2.6 IBindings Interface

Class Hierarchy



Pascal

```
IBindings = interface(IXMLNodeCollection);
```

C++

```
__interface IBindings : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindings.

76.2.6.1 IBindings Members

The following tables list the members exposed by IBindings.

Public Methods

	Name	Description
	Add (see page 7044)	This is Add, a member of class IBindings.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetBinding (see page 7044)	This is GetBinding, a member of class IBindings.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.

	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7044)	This is Insert, a member of class IBindings.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	Binding (see page 7045)	This is Binding, a member of class IBindings.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.6.2 IBindings Methods

The methods of the IBindings class are listed here.

Public Methods

	Name	Description
	Add (see page 7044)	This is Add, a member of class IBindings.
	GetBinding (see page 7044)	This is GetBinding, a member of class IBindings.
	Insert (see page 7044)	This is Insert, a member of class IBindings.

76.2.6.2.1 IBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class IBindings.

76.2.6.2.2 IBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class IBindings.

76.2.6.2.3 IBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class IBindings.

76.2.6.3 IBindings Properties

The properties of the IBindings class are listed here.

Public Properties

	Name	Description
R	Binding (see page 7045)	This is Binding, a member of class IBindings.

76.2.6.3.1 IBindings.Binding Property

Pascal

```
property Binding [Index: Integer]: IBinding;
```

C++

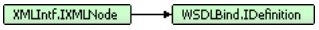
```
__property IBinding Binding[int Index];
```

Description

This is Binding, a member of class IBindings.

76.2.7 IDefinition Interface

Class Hierarchy



Pascal

```
IDefinition = interface(IXMLNode);
```

C++

```
__interface IDefinition : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDefinition.

76.2.7.1 IDefinition Members

The following tables list the members exposed by IDefinition.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetBindings (see page 7047)	This is GetBindings, a member of class IDefinition.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetImports (see page 7047)	This is GetImports, a member of class IDefinition.

	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 7048)	This is GetMessages, a member of class IDefinition.
	GetName (see page 7048)	This is GetName, a member of class IDefinition.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPortTypes (see page 7048)	This is GetPortTypes, a member of class IDefinition.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetServices (see page 7048)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 7048)	This is GetTargetNamespace, a member of class IDefinition.
	GetText (see page 7627)	Returns the text value of the node.
	GetTypes (see page 7049)	This is GetTypes, a member of class IDefinition.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7049)	This is SetName, a member of class IDefinition.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 7049)	This is SetTargetNamespace, a member of class IDefinition.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	Bindings (see page 7050)	This is Bindings, a member of class IDefinition.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	Imports (see page 7050)	This is Imports, a member of class IDefinition.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	Messages (see page 7050)	This is Messages, a member of class IDefinition.
	Name (see page 7050)	This is Name, a member of class IDefinition.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.

	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	PortTypes (see page 7050)	This is PortTypes, a member of class IDefinition.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Services (see page 7051)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7051)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 7643)	Represents the Text of the node.
	Types (see page 7051)	This is Types, a member of class IDefinition.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.7.2 IDefinition Methods

The methods of the IDefinition class are listed here.

Public Methods

	Name	Description
	GetBindings (see page 7047)	This is GetBindings, a member of class IDefinition.
	GetImports (see page 7047)	This is GetImports, a member of class IDefinition.
	GetMessages (see page 7048)	This is GetMessages, a member of class IDefinition.
	GetName (see page 7048)	This is GetName, a member of class IDefinition.
	GetPortTypes (see page 7048)	This is GetPortTypes, a member of class IDefinition.
	GetServices (see page 7048)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 7048)	This is GetTargetNamespace, a member of class IDefinition.
	GetTypes (see page 7049)	This is GetTypes, a member of class IDefinition.
	SetName (see page 7049)	This is SetName, a member of class IDefinition.
	SetTargetNamespace (see page 7049)	This is SetTargetNamespace, a member of class IDefinition.

76.2.7.2.1 IDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class IDefinition.

76.2.7.2.2 IDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class IDefinition.

76.2.7.2.3 IDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class IDefinition.

76.2.7.2.4 IDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IDefinition.

76.2.7.2.5 IDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class IDefinition.

76.2.7.2.6 IDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class IDefinition.

76.2.7.2.7 IDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class IDefinition.

76.2.7.2.8 IDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class IDefinition.

76.2.7.2.9 IDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IDefinition.

76.2.7.2.10 IDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class IDefinition.

76.2.7.3 IDefinition Properties

The properties of the IDefinition class are listed here.

Public Properties

	Name	Description
R	Bindings (see page 7050)	This is Bindings, a member of class IDefinition.
R	Imports (see page 7050)	This is Imports, a member of class IDefinition.
R	Messages (see page 7050)	This is Messages, a member of class IDefinition.
	Name (see page 7050)	This is Name, a member of class IDefinition.
R	PortTypes (see page 7050)	This is PortTypes, a member of class IDefinition.
R	Services (see page 7051)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7051)	This is TargetNamespace, a member of class IDefinition.
R	Types (see page 7051)	This is Types, a member of class IDefinition.

76.2.7.3.1 IDefinition.Bindings Property

Pascal

```
property Bindings: IBindings;
```

C++

```
__property IBindings Bindings;
```

Description

This is Bindings, a member of class IDefinition.

76.2.7.3.2 IDefinition.Imports Property

Pascal

```
property Imports: IImports;
```

C++

```
__property IImports Imports;
```

Description

This is Imports, a member of class IDefinition.

76.2.7.3.3 IDefinition.Messages Property

Pascal

```
property Messages: IMessages;
```

C++

```
__property IMessages Messages;
```

Description

This is Messages, a member of class IDefinition.

76.2.7.3.4 IDefinition.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IDefinition.

76.2.7.3.5 IDefinition.PortTypes Property

Pascal

```
property PortTypes: IPortTypes;
```

C++

```
__property IPortTypes PortTypes;
```

Description

This is PortTypes, a member of class IDefinition.

76.2.7.3.6 IDefinition.Services Property

Pascal

```
property Services: IServices;
```

C++

```
__property IServices Services;
```

Description

This is Services, a member of class IDefinition.

76.2.7.3.7 IDefinition.TargetNamespace Property

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

This is TargetNamespace, a member of class IDefinition.

76.2.7.3.8 IDefinition.Types Property

Pascal

```
property Types: ITypes;
```

C++

```
__property ITypes Types;
```

Description

This is Types, a member of class IDefinition.

76.2.8 IDocumentation Interface

Class Hierarchy

Pascal

```
IDocumentation = interface(IXMLNode);
```

C++

```
__interface IDocumentation : public IXMLNode;
```

File

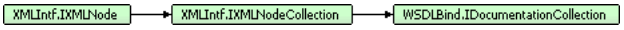
WSDLBind

Description

This is class WSDLBind.IDocumentation.

76.2.9 IDocumentationCollection Interface

Class Hierarchy



Pascal

```
IDocumentationCollection = interface(IXMLNodeCollection);
```

C++

```
__interface IDocumentationCollection : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumentationCollection.

76.2.9.1 IDocumentationCollection Members

The following tables list the members exposed by IDocumentationCollection.

Public Methods

	Name	Description
	Add (see page 7054)	This is Add, a member of class IDocumentationCollection.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDocumentation (see page 7054)	This is GetDocumentation, a member of class IDocumentationCollection.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7054)	This is Insert, a member of class IDocumentationCollection.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	Documentation (see page 7055)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.9.2 IDocumentationCollection Methods

The methods of the IDocumentationCollection class are listed here.

Public Methods

	Name	Description
	Add (see page 7054)	This is Add, a member of class IDocumentationCollection.
	GetDocumentation (see page 7054)	This is GetDocumentation, a member of class IDocumentationCollection.
	Insert (see page 7054)	This is Insert, a member of class IDocumentationCollection.

76.2.9.2.1 IDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class IDocumentationCollection.

76.2.9.2.2 IDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class IDocumentationCollection.

76.2.9.2.3 IDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class IDocumentationCollection.

76.2.9.3 IDocumentationCollection Properties

The properties of the IDocumentationCollection class are listed here.

Public Properties

	Name	Description
R	Documentation (see page 7055)	This is Documentation, a member of class IDocumentationCollection.

76.2.9.3.1 IDocumentationCollection.Documentation Property

Pascal

```
property Documentation [Index: Integer]: IDocumentation;
```

C++

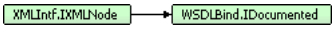
```
__property IDocumentation Documentation[int Index];
```

Description

This is Documentation, a member of class IDocumentationCollection.

76.2.10 IDocumented Interface

Class Hierarchy



Pascal

```
IDocumented = interface(IXMLNode);
```

C++

```
__interface IDocumented : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumented.

76.2.10.1 IDocumented Members

The following tables list the members exposed by IDocumented.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.

	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.10.2 IDocumented Methods

The methods of the IDocumented class are listed here.

Public Methods

	Name	Description
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.

76.2.10.2.1 IDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class IDocumented.

76.2.10.3 IDocumented Properties

The properties of the IDocumented class are listed here.

Public Properties

	Name	Description
	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.

76.2.10.3.1 IDocumented.Documentation Property

Pascal

```
property Documentation: IDocumentationCollection;
```

C++

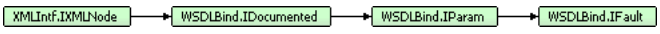
```
__property IDocumentationCollection Documentation;
```

Description

This is Documentation, a member of class IDocumented.

76.2.11 IFault Interface

Class Hierarchy



Pascal

```
IFault = interface(IParam);
```

C++

```
__interface IFault : public IParam;
```

File

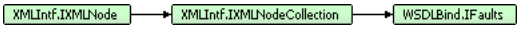
WSDLBind

Description

This is class WSDLBind.IFault.

76.2.12 IFaults Interface

Class Hierarchy



Pascal

```
IFaults = interface(IXMLNodeCollection);
```

C++

```
__interface IFaults : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IFaults.

76.2.12.1 IFaults Members

The following tables list the members exposed by IFaults.

Public Methods

	Name	Description
	Add (see page 7060)	This is Add, a member of class IFaults.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetFault (see page 7060)	This is GetFault, a member of class IFaults.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7060)	This is Insert, a member of class IFaults.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Fault (see page 7061)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.12.2 IFaults Methods

The methods of the IFaults class are listed here.

Public Methods

	Name	Description
	Add (see page 7060)	This is Add, a member of class IFaults.
	GetFault (see page 7060)	This is GetFault, a member of class IFaults.
	Insert (see page 7060)	This is Insert, a member of class IFaults.

76.2.12.2.1 IFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class IFaults.

76.2.12.2.2 IFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class IFaults.

76.2.12.2.3 IFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class IFaults.

76.2.12.3 IFaults Properties

The properties of the IFaults class are listed here.

Public Properties

	Name	Description
	Fault (see page 7061)	This is Fault, a member of class IFaults.

76.2.12.3.1 IFaults.Fault Property

Pascal

```
property Fault [Index: Integer]: IFault;
```

C++

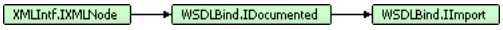
```
__property IFault Fault[int Index];
```

Description

This is Fault, a member of class IFaults.

76.2.13 IImport Interface

Class Hierarchy



Pascal

```
IImport = interface(IDocumented);
```

C++

```
__interface IImport : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IImport.

76.2.13.1 IImport Members

The following tables list the members exposed by IImport.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.

	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 7063)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 7063)	This is GetNamespace, a member of class IImport.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetLocation (see page 7063)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 7063)	This is SetNamespace, a member of class IImport.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Location (see page 7064)	This is Location, a member of class IImport.
	Namespace (see page 7064)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.13.2 IImport Methods

The methods of the IImport class are listed here.

Public Methods

	Name	Description
	GetLocation (see page 7063)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 7063)	This is GetNamespace, a member of class IImport.
	SetLocation (see page 7063)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 7063)	This is SetNamespace, a member of class IImport.

76.2.13.2.1 IImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class IImport.

76.2.13.2.2 IImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class IImport.

76.2.13.2.3 IImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class IImport.

76.2.13.2.4 IImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class IImport.

76.2.13.3 IImport Properties

The properties of the IImport class are listed here.

Public Properties

	Name	Description
	Location (see page 7064)	This is Location, a member of class IImport.
	Namespace (see page 7064)	This is Namespace, a member of class IImport.

76.2.13.3.1 IImport.Location Property

Pascal

```
property Location: WideString;
```

C++

```
__property BSTR Location;
```

Description

This is Location, a member of class IImport.

76.2.13.3.2 IImport.Namespace Property

Pascal

```
property Namespace: WideString;
```

C++

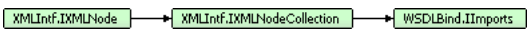
```
__property BSTR Namespace;
```

Description

This is Namespace, a member of class IImport.

76.2.14 IImports Interface

Class Hierarchy



Pascal

```
IImports = interface(IXMLNodeCollection);
```

C++

```
__interface IImports : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IImports.

76.2.14.1 IImports Members

The following tables list the members exposed by IImports.

Public Methods

	Name	Description
	Add (see page 7066)	This is Add, a member of class IImports.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetImport (see page 7067)	This is GetImport, a member of class IImports.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7067)	This is Insert, a member of class IImports.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	Import (see page 7067)	This is Import, a member of class IImports.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.14.2 IImports Methods

The methods of the IImports class are listed here.

Public Methods

	Name	Description
	Add (see page 7066)	This is Add, a member of class IImports.
	GetImport (see page 7067)	This is GetImport, a member of class IImports.
	Insert (see page 7067)	This is Insert, a member of class IImports.

76.2.14.2.1 IImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class IImports.

76.2.14.2.2 IImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class IImports.

76.2.14.2.3 IImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class IImports.

76.2.14.3 IImports Properties

The properties of the IImports class are listed here.

Public Properties

	Name	Description
	Import (see page 7067)	This is Import, a member of class IImports.

76.2.14.3.1 IImports.Import Property

Pascal

```
property Import [Index: Integer]: IImport;
```

C++

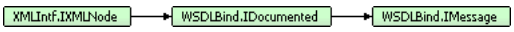
```
__property IImport Import[int Index];
```

Description

This is Import, a member of class IImports.

76.2.15 IMessage Interface

Class Hierarchy



Pascal

```
IMessage = interface(IDocumented);
```

C++

```
__interface IMessage : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IMessage.

76.2.15.1 IMessage Members

The following tables list the members exposed by IMessage.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7069)	This is GetName, a member of class IMessage.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetParts (see page 7070)	This is GetParts, a member of class IMessage.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7070)	This is SetName, a member of class IMessage.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7070)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Parts (see page 7070)	This is Parts, a member of class IMessage.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.15.2 IMessage Methods

The methods of the IMessage class are listed here.

Public Methods

	Name	Description
	GetName (see page 7069)	This is GetName, a member of class IMessage.
	GetParts (see page 7070)	This is GetParts, a member of class IMessage.
	SetName (see page 7070)	This is SetName, a member of class IMessage.

76.2.15.2.1 IMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IMessage.

76.2.15.2.2 IMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class IMessage.

76.2.15.2.3 IMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IMessage.

76.2.15.3 IMessage Properties

The properties of the IMessage class are listed here.

Public Properties

	Name	Description
	Name (see page 7070)	This is Name, a member of class IMessage.
R	Parts (see page 7070)	This is Parts, a member of class IMessage.

76.2.15.3.1 IMessage.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IMessage.

76.2.15.3.2 IMessage.Parts Property

Pascal

```
property Parts: IParts;
```

C++

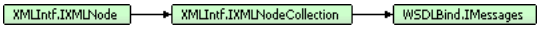
```
__property IParts Parts;
```

Description

This is Parts, a member of class IMessage.

76.2.16 IMessages Interface

Class Hierarchy



Pascal

```
IMessages = interface(IXMLNodeCollection);
```

C++

```
__interface IMessages : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IMessages.

76.2.16.1 IMessages Members

The following tables list the members exposed by IMessages.

Public Methods

	Name	Description
	Add (see page 7073)	This is Add, a member of class IMessages.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 7073)	This is GetMessage, a member of class IMessages.

	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7073)	This is Insert, a member of class IMessages.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	Message (see page 7074)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.16.2 IMessages Methods

The methods of the IMessages class are listed here.

Public Methods

	Name	Description
	Add (see page 7073)	This is Add, a member of class IMessages.
	GetMessage (see page 7073)	This is GetMessage, a member of class IMessages.
	Insert (see page 7073)	This is Insert, a member of class IMessages.

76.2.16.2.1 IMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class IMessages.

76.2.16.2.2 IMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class IMessages.

76.2.16.2.3 IMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class IMessages.

76.2.16.3 IMessages Properties

The properties of the IMessages class are listed here.

Public Properties

	Name	Description
R	Message (see page 7074)	This is Message, a member of class IMessages.

76.2.16.3.1 IMessages.Message Property

Pascal

```
property Message [Index: Integer]: IMessage;
```

C++

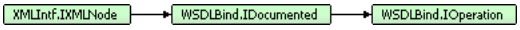
```
__property IMessage Message[int Index];
```

Description

This is Message, a member of class IMessages.

76.2.17 IOperation Interface

Class Hierarchy



Pascal

```
IOperation = interface(IDocumented);
```

C++

```
__interface IOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IOperation.

76.2.17.1 IOperation Members

The following tables list the members exposed by IOperation.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetFaults (see page 7076)	This is GetFaults, a member of class IOperation.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.

	GetInput (see page 7076)	This is GetInput, a member of class IOperation.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7076)	This is GetName, a member of class IOperation.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOutput (see page 7077)	This is GetOutput, a member of class IOperation.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7077)	This is SetName, a member of class IOperation.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	Faults (see page 7077)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	Input (see page 7077)	This is Input, a member of class IOperation.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7078)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Output (see page 7078)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.

	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.17.2 IOperation Methods

The methods of the IOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 7076)	This is GetFaults, a member of class IOperation.
	GetInput (see page 7076)	This is GetInput, a member of class IOperation.
	GetName (see page 7076)	This is GetName, a member of class IOperation.
	GetOutput (see page 7077)	This is GetOutput, a member of class IOperation.
	SetName (see page 7077)	This is SetName, a member of class IOperation.

76.2.17.2.1 IOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IOperation.

76.2.17.2.2 IOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class IOperation.

76.2.17.2.3 IOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IOperation.

76.2.17.2.4 IOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class IOperation.

76.2.17.2.5 IOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IOperation.

76.2.17.3 IOperation Properties

The properties of the IOperation class are listed here.

Public Properties

	Name	Description
R	Faults (see page 7077)	This is Faults, a member of class IOperation.
R	Input (see page 7077)	This is Input, a member of class IOperation.
	Name (see page 7078)	This is Name, a member of class IOperation.
R	Output (see page 7078)	This is Output, a member of class IOperation.

76.2.17.3.1 IOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IOperation.

76.2.17.3.2 IOperation.Input Property

Pascal

```
property Input: IParam;
```

C++

```
__property IParam Input;
```

Description

This is Input, a member of class IOperation.

76.2.17.3.3 IOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IOperation.

76.2.17.3.4 IOperation.Output Property

Pascal

```
property Output: IParam;
```

C++

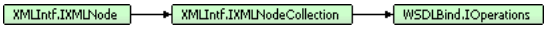
```
__property IParam Output;
```

Description

This is Output, a member of class IOperation.

76.2.18 IOperations Interface

Class Hierarchy



Pascal

```
IOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IOperations.

76.2.18.1 IOperations Members

The following tables list the members exposed by IOperations.

Public Methods

	Name	Description
	Add (see page 7080)	This is the overview for the Add method overload.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOperation (see page 7081)	This is GetOperation, a member of class IOperations.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7081)	This is Insert, a member of class IOperations.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.

	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Operation (see page 7081)	This is Operation, a member of class IOperations.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.18.2 IOperations Methods

The methods of the IOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 7080)	This is the overview for the Add method overload.
	GetOperation (see page 7081)	This is GetOperation, a member of class IOperations.
	Insert (see page 7081)	This is Insert, a member of class IOperations.

76.2.18.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	IOperations.Add (WideString) (see page 7080)	This is Add, a member of class IOperations.
	IOperations.Add (WideString, WideString, WideString, WideString) (see page 7081)	This is Add, a member of class IOperations.

76.2.18.2.1.1 IOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IOperations.

76.2.18.2.1.2 IOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class IOperations.

76.2.18.2.2 IOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class IOperations.

76.2.18.2.3 IOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class IOperations.

76.2.18.3 IOperations Properties

The properties of the IOperations class are listed here.

Public Properties

	Name	Description
R	Operation (see page 7081)	This is Operation, a member of class IOperations.

76.2.18.3.1 IOperations.Operation Property

Pascal

```
property Operation [Index: Integer]: IOperation;
```

C++

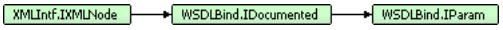
```
__property IOperation Operation[int Index];
```

Description

This is Operation, a member of class IOperations.

76.2.19 IParam Interface

Class Hierarchy



Pascal

```
IParam = interface(IDocumented);
```

C++

```
__interface IParam : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IParam.

76.2.19.1 IParam Members

The following tables list the members exposed by IParam.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 7084)	This is GetMessage, a member of class IParam.
	GetName (see page 7084)	This is GetName, a member of class IParam.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.

	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetMessage (see page 7084)	This is SetMessage, a member of class IParam.
	SetName (see page 7084)	This is SetName, a member of class IParam.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Message (see page 7085)	This is Message, a member of class IParam.
	Name (see page 7085)	This is Name, a member of class IParam.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.19.2 IParam Methods

The methods of the IParam class are listed here.

Public Methods

	Name	Description
	GetMessage (see page 7084)	This is GetMessage, a member of class IParam.
	GetName (see page 7084)	This is GetName, a member of class IParam.
	SetMessage (see page 7084)	This is SetMessage, a member of class IParam.
	SetName (see page 7084)	This is SetName, a member of class IParam.

76.2.19.2.1 IParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class IParam.

76.2.19.2.2 IParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IParam.

76.2.19.2.3 IParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class IParam.

76.2.19.2.4 IParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IParam.

76.2.19.3 IParam Properties

The properties of the IParam class are listed here.

Public Properties

	Name	Description
	Message (see page 7085)	This is Message, a member of class IParam.
	Name (see page 7085)	This is Name, a member of class IParam.

76.2.19.3.1 IParam.Message Property

Pascal

```
property Message: WideString;
```

C++

```
__property BSTR Message;
```

Description

This is Message, a member of class IParam.

76.2.19.3.2 IParam.Name Property

Pascal

```
property Name: WideString;
```

C++

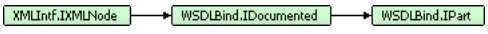
```
__property BSTR Name;
```

Description

This is Name, a member of class IParam.

76.2.20 IPart Interface

Class Hierarchy



Pascal

```
IPart = interface(IDocumented);
```

C++

```
__interface IPart : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPart.

76.2.20.1 IPart Members

The following tables list the members exposed by IPart.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetElement (see page 7087)	This is GetElement, a member of class IPart.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7088)	This is GetName, a member of class IPart.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetType_ (see page 7088)	This is GetType_, a member of class IPart.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetElement (see page 7088)	This is SetElement, a member of class IPart.
	SetName (see page 7088)	This is SetName, a member of class IPart.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	SetType_ (see page 7088)	This is SetType_, a member of class IPart.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	Element (see page 7089)	This is Element, a member of class IPart.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7089)	This is Name, a member of class IPart.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
	Type_ (see page 7089)	This is Type_, a member of class IPart.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.20.2 IPart Methods

The methods of the IPart class are listed here.

Public Methods

	Name	Description
	GetElement (see page 7087)	This is GetElement, a member of class IPart.
	GetName (see page 7088)	This is GetName, a member of class IPart.
	GetType_ (see page 7088)	This is GetType_, a member of class IPart.
	SetElement (see page 7088)	This is SetElement, a member of class IPart.
	SetName (see page 7088)	This is SetName, a member of class IPart.
	SetType_ (see page 7088)	This is SetType_, a member of class IPart.

76.2.20.2.1 IPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class IPart.

76.2.20.2.2 IPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPart.

76.2.20.2.3 IPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IPart.

76.2.20.2.4 IPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class IPart.

76.2.20.2.5 IPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPart.

76.2.20.2.6 IPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IPart.

76.2.20.3 IPart Properties

The properties of the IPart class are listed here.

Public Properties

	Name	Description
	Element (see page 7089)	This is Element, a member of class IPart.
	Name (see page 7089)	This is Name, a member of class IPart.
	Type_ (see page 7089)	This is Type_, a member of class IPart.

76.2.20.3.1 IPart.Element Property

Pascal

```
property Element: WideString;
```

C++

```
__property BSTR Element;
```

Description

This is Element, a member of class IPart.

76.2.20.3.2 IPart.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPart.

76.2.20.3.3 IPart.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

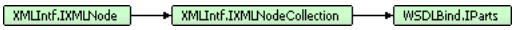
```
__property BSTR Type_;
```

Description

This is Type_, a member of class IPart.

76.2.21 IParts Interface

Class Hierarchy



Pascal

```
IParts = interface(IXMLNodeCollection);
```

C++

```
__interface IParts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IParts.

76.2.21.1 IParts Members

The following tables list the members exposed by IParts.

Public Methods

	Name	Description
	Add (see page 7092)	This is Add, a member of class IParts.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.

	GetPart (see page 7092)	This is GetPart, a member of class IParts.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7092)	This is Insert, a member of class IParts.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Part (see page 7093)	This is Part, a member of class IParts.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.21.2 IParts Methods

The methods of the IParts class are listed here.

Public Methods

	Name	Description
	Add (see page 7092)	This is Add, a member of class IParts.
	GetPart (see page 7092)	This is GetPart, a member of class IParts.
	Insert (see page 7092)	This is Insert, a member of class IParts.

76.2.21.2.1 IParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class IParts.

76.2.21.2.2 IParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class IParts.

76.2.21.2.3 IParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class IParts.

76.2.21.3 IParts Properties

The properties of the IParts class are listed here.

Public Properties

	Name	Description
R	Part (see page 7093)	This is Part, a member of class IParts.

76.2.21.3.1 IParts.Part Property

Pascal

```
property Part [Index: Integer]: IPart;
```

C++

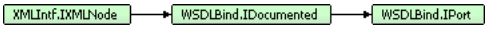
```
__property IPart Part[int Index];
```

Description

This is Part, a member of class IParts.

76.2.22 IPort Interface

Class Hierarchy



Pascal

```
IPort = interface(IDocumented);
```

C++

```
__interface IPort : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPort.

76.2.22.1 IPort Members

The following tables list the members exposed by IPort.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetBinding (see page 7095)	This is GetBinding, a member of class IPort.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.

	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7095)	This is GetName, a member of class IPort.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 7095)	This is SetBinding, a member of class IPort.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7095)	This is SetName, a member of class IPort.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
	Binding (see page 7096)	This is Binding, a member of class IPort.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7096)	This is Name, a member of class IPort.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.22.2 IPort Methods

The methods of the IPort class are listed here.

Public Methods

	Name	Description
	GetBinding (see page 7095)	This is GetBinding, a member of class IPort.
	GetName (see page 7095)	This is GetName, a member of class IPort.
	SetBinding (see page 7095)	This is SetBinding, a member of class IPort.
	SetName (see page 7095)	This is SetName, a member of class IPort.

76.2.22.2.1 IPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class IPort.

76.2.22.2.2 IPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPort.

76.2.22.2.3 IPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class IPort.

76.2.22.2.4 IPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPort.

76.2.22.3 IPort Properties

The properties of the IPort class are listed here.

Public Properties

	Name	Description
	Binding (see page 7096)	This is Binding, a member of class IPort.
	Name (see page 7096)	This is Name, a member of class IPort.

76.2.22.3.1 IPort.Binding Property

Pascal

```
property Binding: WideString;
```

C++

```
__property BSTR Binding;
```

Description

This is Binding, a member of class IPort.

76.2.22.3.2 IPort.Name Property

Pascal

```
property Name: WideString;
```

C++

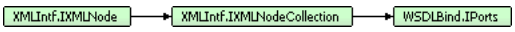
```
__property BSTR Name;
```

Description

This is Name, a member of class IPort.

76.2.23 IPorts Interface

Class Hierarchy



Pascal

```
IPorts = interface(IXMLNodeCollection);
```

C++

```
__interface IPorts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPorts.

76.2.23.1 IPorts Members

The following tables list the members exposed by IPorts.

Public Methods

	Name	Description
	Add (see page 7098)	This is Add, a member of class IPorts.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPort (see page 7099)	This is GetPort, a member of class IPorts.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7099)	This is Insert, a member of class IPorts.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Port (see page 7099)	This is Port, a member of class IPorts.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.23.2 IPorts Methods

The methods of the IPorts class are listed here.

Public Methods

	Name	Description
	Add (see page 7098)	This is Add, a member of class IPorts.
	GetPort (see page 7099)	This is GetPort, a member of class IPorts.
	Insert (see page 7099)	This is Insert, a member of class IPorts.

76.2.23.2.1 IPorts.Add Method

Pascal

```
function Add(const Name: WideString; const ABinding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR ABinding);
```

Description

This is Add, a member of class IPorts.

76.2.23.2.2 IPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class IPorts.

76.2.23.2.3 IPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class IPorts.

76.2.23.3 IPorts Properties

The properties of the IPorts class are listed here.

Public Properties

	Name	Description
	Port (see page 7099)	This is Port, a member of class IPorts.

76.2.23.3.1 IPorts.Port Property

Pascal

```
property Port [Index: Integer]: IPort;
```

C++

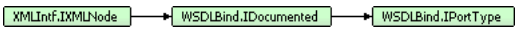
```
__property IPort Port[int Index];
```

Description

This is Port, a member of class IPorts.

76.2.24 IPortType Interface

Class Hierarchy



Pascal

```
IPortType = interface(IDocumented);
```

C++

```
__interface IPortType : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPortType.

76.2.24.1 IPortType Members

The following tables list the members exposed by IPortType.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7101)	This is GetName, a member of class IPortType.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOperations (see page 7102)	This is GetOperations, a member of class IPortType.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7102)	This is SetName, a member of class IPortType.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7102)	This is Name, a member of class IPortType.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Operations (see page 7102)	This is Operations, a member of class IPortType.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.24.2 IPortType Methods

The methods of the IPortType class are listed here.

Public Methods

	Name	Description
	GetName (see page 7101)	This is GetName, a member of class IPortType.
	GetOperations (see page 7102)	This is GetOperations, a member of class IPortType.
	SetName (see page 7102)	This is SetName, a member of class IPortType.

76.2.24.2.1 IPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPortType.

76.2.24.2.2 IPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class IPortType.

76.2.24.2.3 IPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPortType.

76.2.24.3 IPortType Properties

The properties of the IPortType class are listed here.

Public Properties

	Name	Description
	Name (see page 7102)	This is Name, a member of class IPortType.
R	Operations (see page 7102)	This is Operations, a member of class IPortType.

76.2.24.3.1 IPortType.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPortType.

76.2.24.3.2 IPortType.Operations Property

Pascal

```
property Operations: IOperations;
```

C++

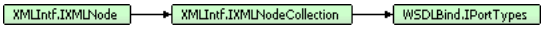
```
__property IOperations Operations;
```

Description

This is Operations, a member of class IPortType.

76.2.25 IPortTypes Interface

Class Hierarchy



Pascal

```
IPortTypes = interface(IXMLNodeCollection);
```

C++

```
__interface IPortTypes : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPortTypes.

76.2.25.1 IPortTypes Members

The following tables list the members exposed by IPortTypes.

Public Methods

	Name	Description
	Add (see page 7105)	This is Add, a member of class IPortTypes.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.

	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPortType (see page 7105)	This is GetPortType, a member of class IPortTypes.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7105)	This is Insert, a member of class IPortTypes.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	PortType (see page 7106)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.25.2 IPortTypes Methods

The methods of the IPortTypes class are listed here.

Public Methods

	Name	Description
	Add (see page 7105)	This is Add, a member of class IPortTypes.
	GetPortType (see page 7105)	This is GetPortType, a member of class IPortTypes.
	Insert (see page 7105)	This is Insert, a member of class IPortTypes.

76.2.25.2.1 IPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class IPortTypes.

76.2.25.2.2 IPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class IPortTypes.

76.2.25.2.3 IPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class IPortTypes.

76.2.25.3 IPortTypes Properties

The properties of the IPortTypes class are listed here.

Public Properties

	Name	Description
R	PortType (see page 7106)	This is PortType, a member of class IPortTypes.

76.2.25.3.1 IPortTypes.PortType Property

Pascal

```
property PortType [Index: Integer]: IPortType;
```

C++

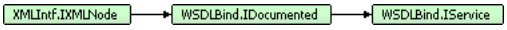
```
__property IPortType PortType[int Index];
```

Description

This is PortType, a member of class IPortTypes.

76.2.26 IService Interface

Class Hierarchy



Pascal

```
IService = interface(IDocumented);
```

C++

```
__interface IService : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IService.

76.2.26.1 IService Members

The following tables list the members exposed by IService.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.

	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetName (see page 7108)	This is GetName, a member of class IService.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPorts (see page 7108)	This is GetPorts, a member of class IService.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetName (see page 7108)	This is SetName, a member of class IService.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	Name (see page 7109)	This is Name, a member of class IService.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Ports (see page 7109)	This is Ports, a member of class IService.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.26.2 IService Methods

The methods of the IService class are listed here.

Public Methods

	Name	Description
	GetName (see page 7108)	This is GetName, a member of class IService.
	GetPorts (see page 7108)	This is GetPorts, a member of class IService.
	SetName (see page 7108)	This is SetName, a member of class IService.

76.2.26.2.1 IService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IService.

76.2.26.2.2 IService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class IService.

76.2.26.2.3 IService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IService.

76.2.26.3 IService Properties

The properties of the IService class are listed here.

Public Properties

	Name	Description
	Name (see page 7109)	This is Name, a member of class IService.

	Ports (see page 7109)	This is Ports, a member of class IService.

76.2.26.3.1 IService.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IService.

76.2.26.3.2 IService.Ports Property

Pascal

```
property Ports: IPorts;
```

C++

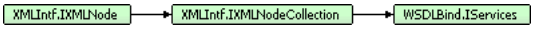
```
__property IPorts Ports;
```

Description

This is Ports, a member of class IService.

76.2.27 IServices Interface

Class Hierarchy



Pascal

```
IServices = interface(IXMLNodeCollection);
```

C++

```
__interface IServices : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IServices.

76.2.27.1 IServices Members

The following tables list the members exposed by IServices.

Public Methods

	Name	Description
	Add (see page 7111)	This is Add, a member of class IServices.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.

	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetService (see page 7111)	This is GetService, a member of class IServices.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	Insert (see page 7112)	This is Insert, a member of class IServices.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.

	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Service (see page 7112)	This is Service, a member of class IServices.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.27.2 IServices Methods

The methods of the IServices class are listed here.

Public Methods

	Name	Description
	Add (see page 7111)	This is Add, a member of class IServices.
	GetService (see page 7111)	This is GetService, a member of class IServices.
	Insert (see page 7112)	This is Insert, a member of class IServices.

76.2.27.2.1 IServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class IServices.

76.2.27.2.2 IServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class IServices.

76.2.27.2.3 IServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class IServices.

76.2.27.3 IServices Properties

The properties of the IServices class are listed here.

Public Properties

	Name	Description
	Service (see page 7112)	This is Service, a member of class IServices.

76.2.27.3.1 IServices.Service Property

Pascal

```
property Service [Index: Integer]: IService;
```

C++

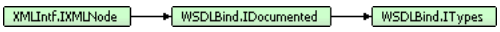
```
__property IService Service[int Index];
```

Description

This is Service, a member of class IServices.

76.2.28 ITypes Interface

Class Hierarchy



Pascal

```
ITypes = interface(IDocumented);
```

C++

```
__interface ITypes : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.ITypes.

76.2.28.1 ITypes Members

The following tables list the members exposed by ITypes.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDocumentationCollection (see page 7057)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetSchemaDefs (see page 7114)	This is GetSchemaDefs, a member of class ITypes.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.

	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Documentation (see page 7057)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	SchemaDefs (see page 7114)	This is SchemaDefs, a member of class ITypes.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.28.2 ITypes Methods

The methods of the ITypes class are listed here.

Public Methods

	Name	Description
	GetSchemaDefs (see page 7114)	This is GetSchemaDefs, a member of class ITypes.

76.2.28.2.1 ITypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

Description

This is GetSchemaDefs, a member of class ITypes.

76.2.28.3 ITypes Properties

The properties of the ITypes class are listed here.

Public Properties

	Name	Description
R	SchemaDefs (see page 7114)	This is SchemaDefs, a member of class ITypes.

76.2.28.3.1 ITypes.SchemaDefs Property

Pascal

```
property SchemaDefs: IXMLSchemaDefs;
```

C++

```
__property IXMLSchemaDefs SchemaDefs;
```

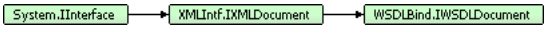
Description

This is SchemaDefs, a member of class ITypes.

76.2.29 IWSDLDocument Interface

IWSDLDocument provides low-level access to the definitions in a WSDL document.

Class Hierarchy



Pascal

```
IWSDLDocument = interface(IXMLDocument);
```

C++

```
__interface IWSDLDocument : public IXMLDocument;
```

File

WSDLBind

Description

IWSDLDocument is an interface to the TWSDLDocument (see page 7015) class. It is primarily for internal use, because it only provides access to the Definitions property, which is used for low-level access to the definitions in a WSDL document. It is generally easier to work with a TWSDLItems object. You can often obtain this from an IWSDLDocument interface by using the global Supports function (Delphi) or the Supports method (C++) to obtain an IWSDLItems interface from this one, and then obtaining a TWSDLItems object from that interface.

Note: C++ method declarations that use IWSDLDocument use the _di_IWSDLDocument type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWSDLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLDocument (see page 7015)

IWSDLItems (see page 7208)

TWSDLItems (see page 7163)

76.2.29.1 IWSDLDocument Members

The following tables list the members exposed by IWSDLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.

	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDefinition (see page 7117)	Returns the value of the Definition (see page 7118) property.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDefinition (see page 7117)	Sets the value of the Definition (see page 7118) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.
	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.

R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
	Definition (see page 7118)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.
	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

76.2.29.2 IWSDLDocument Methods

The methods of the IWSDLDocument class are listed here.

Public Methods

	Name	Description
	GetDefinition (see page 7117)	Returns the value of the Definition (see page 7118) property.
	SetDefinition (see page 7117)	Sets the value of the Definition (see page 7118) property.

76.2.29.2.1 IWSDLDocument.GetDefinition Method

Returns the value of the Definition (see page 7118) property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition (see page 7118) property.

See Also

Definition (see page 7118)

76.2.29.2.2 IWSDLDocument.SetDefinition Method

Sets the value of the Definition (see page 7118) property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition (see page 7118) property.

See Also

Definition (see page 7118)

76.2.29.3 IWSDLDocument Properties

The properties of the IWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 7118)	Provides access to the definitions contained in the WSDL document.

76.2.29.3.1 IWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 7045) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 7045) interface to access the Bindings, Messages, PortTypes, Services, and Types (see page 4683) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems. To obtain a TWSDLItems instance for this document,

1.Cast this interface to IWSDLItems using the global Supports function (C++) or the inherited Supports method (C++).

2.Use the GetWSDLItems method of that interface.

See Also

DocumentElement (see page 7609)

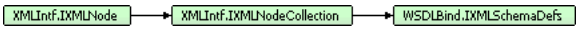
TWSDLItems (see page 7163)

Supports

Supports (see page 4335)

76.2.30 IXMLSchemaDefs Interface

Class Hierarchy



Pascal

```
IXMLSchemaDefs = interface(IXMLNodeCollection);
```

C++

```
__interface IXMLSchemaDefs : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IXMLSchemaDefs.

76.2.30.1 IXMLSchemaDefs Members

The following tables list the members exposed by IXMLSchemaDefs.

Public Methods

	Name	Description
	Add (see page 7121)	This is Add, a member of class IXMLSchemaDefs.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.

	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	GetXMLSchemaDef (see page 7121)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	SchemaDef (see page 7121)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

76.2.30.2 IXMLSchemaDefs Methods

The methods of the IXMLSchemaDefs class are listed here.

Public Methods

	Name	Description
	Add (see page 7121)	This is Add, a member of class IXMLSchemaDefs.

	GetXMLSchemaDef (see page 7121)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

76.2.30.2.1 IXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class IXMLSchemaDefs.

76.2.30.2.2 IXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

76.2.30.3 IXMLSchemaDefs Properties

The properties of the IXMLSchemaDefs class are listed here.

Public Properties

	Name	Description
	SchemaDef (see page 7121)	This is SchemaDef, a member of class IXMLSchemaDefs.

76.2.30.3.1 IXMLSchemaDefs.SchemaDef Property

Pascal

```
property SchemaDef [Index: Integer]: IXMLSchemaDef;
```

C++

```
__property IXMLSchemaDef SchemaDef[int Index];
```

Description

This is SchemaDef, a member of class IXMLSchemaDefs.

76.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetDefinition (see page 7122)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 7122)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 7122)	This is function WSDLBind.NewWSDLDoc.

76.3.1 WSDLBind.GetDefinition Function

Pascal

```
function GetDefinition(Doc: TXMLDocument): IDefinition;
```

C++

```
IDefinition GetDefinition(TXMLDocument Doc);
```

File

WSDLBind

Description

This is function WSDLBind.GetDefinition.

76.3.2 WSDLBind.LoadWSDLDoc Function

Pascal

```
function LoadWSDLDoc(const FileName: WideString): IWSDLDocument;
```

C++

```
IWSDLDocument LoadWSDLDoc(const BSTR FileName);
```

File

WSDLBind

Description

This is function WSDLBind.LoadWSDLDoc.

76.3.3 WSDLBind.NewWSDLDoc Function

Pascal

```
function NewWSDLDoc: IWSDLDocument;
```

C++

```
IWSDLDocument NewWSDLDoc();
```

File

WSDLBind

Description

This is function WSDLBind.NewWSDLDoc.

76.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SBinding (see page 7124)	This is constant WSDLBind.SBinding.
SDefinitions (see page 7124)	This is constant WSDLBind.SDefinitions.
SFault (see page 7125)	This is constant WSDLBind.SFault.
SImport (see page 7125)	This is constant WSDLBind.SImport.
SInput (see page 7125)	This is constant WSDLBind.SInput.
SLocation (see page 7126)	This is constant WSDLBind.SLocation.
SMessage (see page 7126)	This is constant WSDLBind.SMessage.
SNamespace (see page 7126)	This is constant WSDLBind.SNamespace.
SOperation (see page 7127)	This is constant WSDLBind.SOperation.
SOutput (see page 7127)	This is constant WSDLBind.SOutput.
SPart (see page 7127)	This is constant WSDLBind.SPart.
SPort (see page 7128)	This is constant WSDLBind.SPort.
SPortType (see page 7128)	This is constant WSDLBind.SPortType.
SService (see page 7128)	This is constant WSDLBind.SService.
SType (see page 7129)	This is constant WSDLBind.SType.
STypes (see page 7129)	This is constant WSDLBind.STypes.

76.4.1 WSDLBind.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLBind

Description

This is constant WSDLBind.SBinding.

76.4.2 WSDLBind.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLBind

Description

This is constant WSDLBind.SDefinitions.

76.4.3 WSDLBind.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLBind

Description

This is constant WSDLBind.SFault.

76.4.4 WSDLBind.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLBind

Description

This is constant WSDLBind.SImport.

76.4.5 WSDLBind.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLBind

Description

This is constant WSDLBind.SInput.

76.4.6 WSDLBind.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLBind

Description

This is constant WSDLBind.SLocation.

76.4.7 WSDLBind.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLBind

Description

This is constant WSDLBind.SMessage.

76.4.8 WSDLBind.SNamespace Constant

Pascal

```
SNamespace = 'namespace';
```

C++

```
SNamespace = 'namespace';
```

File

WSDLBind

Description

This is constant WSDLBind.SNamespace.

76.4.9 WSDLBind.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLBind

Description

This is constant WSDLBind.SOperation.

76.4.10 WSDLBind.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLBind

Description

This is constant WSDLBind.SOutput.

76.4.11 WSDLBind.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLBind

Description

This is constant WSDLBind.SPart.

76.4.12 WSDLBind.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLBind

Description

This is constant WSDLBind.SPort.

76.4.13 WSDLBind.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLBind

Description

This is constant WSDLBind.SPortType.

76.4.14 WSDLBind.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLBind

Description

This is constant WSDLBind.SService.

76.4.15 WSDLBind.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLBind

Description

This is constant WSDLBind.SType.

76.4.16 WSDLBind.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLBind

Description

This is constant WSDLBind.STypes.

77 WSDLIntf Namespace

This is namespace WSDLIntf.

Topics

Name	Description
Classes (see page 7133)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 7137)	The following table lists structs, records, enums in this documentation.
Types (see page 7139)	The following table lists types in this documentation.
Constants (see page 7140)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDOMStrings (see page 7133)	This is class WSDLIntf.TDOMStrings.

Constants

Name	Description
Directives (see page 7141)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 7141)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 7142)	This is constant WSDLIntf.Httpns.
Operators (see page 7142)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 7142)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 7143)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 7143)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 7144)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 7144)	This is constant WSDLIntf.SAnyType.
SArray (see page 7144)	This is constant WSDLIntf.SArray.
SArrayOf (see page 7145)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 7145)	This is constant WSDLIntf.SArrayType.
SBinding (see page 7145)	This is constant WSDLIntf.SBinding.
SBody (see page 7146)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 7146)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 7146)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 7147)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 7147)	This is constant WSDLIntf.SDynArray.
SElement (see page 7147)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 7148)	This is constant WSDLIntf.SEncodingStyle.

SFault (see page 7148)	This is constant WSDLIntf.SFault.
SHeader (see page 7148)	This is constant WSDLIntf.SHeader.
SImport (see page 7149)	This is constant WSDLIntf.SImport.
SInOut (see page 7149)	This is constant WSDLIntf.SInOut.
SInput (see page 7149)	This is constant WSDLIntf.SInput.
SLocation (see page 7150)	This is constant WSDLIntf.SLocation.
SMessage (see page 7150)	This is constant WSDLIntf.SMessage.
SName (see page 7150)	This is constant WSDLIntf.SName.
SNameSpace (see page 7151)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 7151)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 7151)	This is constant WSDLIntf.Soapns.
SOperation (see page 7152)	This is constant WSDLIntf.SOperation.
SOutput (see page 7152)	This is constant WSDLIntf.SOutput.
SPart (see page 7152)	This is constant WSDLIntf.SPart.
SParts (see page 7153)	This is constant WSDLIntf.SParts.
SPort (see page 7153)	This is constant WSDLIntf.SPort.
SPortType (see page 7153)	This is constant WSDLIntf.SPortType.
SRequest (see page 7154)	This is constant WSDLIntf.SRequest.
SResponse (see page 7154)	This is constant WSDLIntf.SResponse.
SReturn (see page 7154)	This is constant WSDLIntf.SReturn.
SSchema (see page 7155)	This is constant WSDLIntf.SSchema.
SService (see page 7155)	This is constant WSDLIntf.SService.
SSoapAction (see page 7155)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 7156)	This is constant WSDLIntf.SStyle.
STns (see page 7156)	This is constant WSDLIntf.STns.
STransport (see page 7156)	This is constant WSDLIntf.STransport.
SType (see page 7157)	This is constant WSDLIntf.SType.
STypes (see page 7157)	This is constant WSDLIntf.STypes.
SUnknown (see page 7157)	This is constant WSDLIntf.SUnknown.
SUse (see page 7158)	This is constant WSDLIntf.SUse.
SVerb (see page 7158)	This is constant WSDLIntf.SVerb.
tns (see page 7158)	This is constant WSDLIntf.tns.
Wsdlns (see page 7159)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 7159)	This is constant WSDLIntf.xsdns.

Structs, Records, Enums

	Name	Description
	TWebServiceBindingType (see page 7137)	This is record WSDLIntf.TWebServiceBindingType.
	TWSDLElement (see page 7137)	This is record WSDLIntf.TWSDLElement.
	TWString (see page 7138)	This is record WSDLIntf.TWString.

Types

Name	Description
TWSDLElements (see page 7139)	This is type WSDLIntf.TWSDLElements.

77.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDOMStrings (see page 7133)	This is class WSDLIntf.TDOMStrings.

77.1.1 TDOMStrings Class

Class Hierarchy

Pascal

```
TDOMStrings = class;
```

C++

```
class TDOMStrings;
```

File

WSDLIntf

Description

This is class WSDLIntf.TDOMStrings.

77.1.1.1 TDOMStrings Members

The following tables list the members exposed by TDOMStrings.

Public Methods

	Name	Description
	Add (see page 7134)	This is Add, a member of class TDOMStrings.
	Clear (see page 7135)	This is Clear, a member of class TDOMStrings.
	Count (see page 7135)	This is Count, a member of class TDOMStrings.
	Create (see page 7134)	This is Create, a member of class TDOMStrings.
	Destroy (see page 7134)	This is Destroy, a member of class TDOMStrings.
	IndexOf (see page 7135)	This is IndexOf, a member of class TDOMStrings.
	IndexOfIgnoreCase (see page 7135)	This is IndexOfIgnoreCase, a member of class TDOMStrings.
	Insert (see page 7135)	This is Insert, a member of class TDOMStrings.

Public Properties

	Name	Description
	Strings (see page 7136)	This is Strings, a member of class TDOMStrings.

77.1.1.2 TDOMStrings.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDOMStrings();
```

Description

This is Create, a member of class TDOMStrings.

77.1.1.3 TDOMStrings.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDOMStrings();
```

Description

This is Destroy, a member of class TDOMStrings.

77.1.1.4 TDOMStrings Methods

The methods of the TDOMStrings class are listed here.

Public Methods

	Name	Description
	Add (see page 7134)	This is Add, a member of class TDOMStrings.
	Clear (see page 7135)	This is Clear, a member of class TDOMStrings.
	Count (see page 7135)	This is Count, a member of class TDOMStrings.
	IndexOf (see page 7135)	This is IndexOf, a member of class TDOMStrings.
	IndexOfIgnoreCase (see page 7135)	This is IndexOfIgnoreCase, a member of class TDOMStrings.
	Insert (see page 7135)	This is Insert, a member of class TDOMStrings.

77.1.1.4.1 TDOMStrings.Add Method

Pascal

```
function Add(const S: DOMString): Integer;
```

C++

```
__fastcall int Add(const DOMString S);
```

Description

This is Add, a member of class TDOMStrings.

77.1.1.4.2 TDOMStrings.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TDOMStrings.

77.1.1.4.3 TDOMStrings.Count Method

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class TDOMStrings.

77.1.1.4.4 TDOMStrings.IndexOf Method

Pascal

```
function IndexOf(const S: DOMString): Integer;
```

C++

```
__fastcall int IndexOf(const DOMString S);
```

Description

This is IndexOf, a member of class TDOMStrings.

77.1.1.4.5 TDOMStrings.IndexOfIgnoreCase Method

Pascal

```
function IndexOfIgnoreCase(const S: DOMString): Integer;
```

C++

```
__fastcall int IndexOfIgnoreCase(const DOMString S);
```

Description

This is IndexOfIgnoreCase, a member of class TDOMStrings.

77.1.1.4.6 TDOMStrings.Insert Method

Pascal

```
procedure Insert(Index: Integer; const S: DOMString);
```

C++

```
__fastcall Insert(int Index, const DOMString S);
```

Description

This is Insert, a member of class TDOMStrings.

77.1.1.5 TDOMStrings Properties

The properties of the TDOMStrings class are listed here.

Public Properties

	Name	Description
	Strings (see page 7136)	This is Strings, a member of class TDOMStrings.

77.1.1.5.1 TDOMStrings.Strings Property

Pascal

```
property Strings [Index: Integer]: DOMString;
```

C++

```
__property DOMString Strings[int Index];
```

Description

This is Strings, a member of class TDOMStrings.

77.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebServiceBindingType (see page 7137)	This is record WSDLIntf.TWebServiceBindingType.
	TWSDLElement (see page 7137)	This is record WSDLIntf.TWSDLElement.

Records

	Name	Description
	TWString (see page 7138)	This is record WSDLIntf.TWString.

77.2.1 WSDLIntf.TWebServiceBindingType Enumeration

Pascal

```
TWebServiceBindingType = (
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
);
```

C++

```
enum TWebServiceBindingType {
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
};
```

File

WSDLIntf

Description

This is record WSDLIntf.TWebServiceBindingType.

77.2.2 WSDLIntf.TWSDLElement Enumeration

Pascal

```
TWSDLElement = (
  weServiceIntf,
  weServiceImpl,
  weMessage,
  wePortType,
  weBinding,
```

```
    weService,
    weTypes,
    WeImport,
    weOperation,
    wePart
  );
```

C++

```
  enum TWSDLElement {
    weServiceIntf,
    weServiceImpl,
    weMessage,
    wePortType,
    weBinding,
    weService,
    weTypes,
    WeImport,
    weOperation,
    wePart
  };
```

File

WSDLIntf

Description

This is record WSDLIntf.TWSDLElement.

77.2.3 WSDLIntf.TWString Record

Pascal

```
  TWString = record
    WString: DOMString;
  end;
```

C++

```
  struct TWString {
    DOMString WString;
  };
```

File

WSDLIntf

Description

This is record WSDLIntf.TWString.

77.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWSDLElements (see page 7139)	This is type WSDLIntf.TWSDLElements.

77.3.1 WSDLIntf.TWSDLElements Type

Pascal

```
TWSDLElements = set of TWSDLElement;
```

C++

```
TWSDLElement TWSDLElements;
```

File

WSDLIntf

Description

This is type WSDLIntf.TWSDLElements.

77.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Directives (see page 7141)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 7141)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 7142)	This is constant WSDLIntf.Httpns.
Operators (see page 7142)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 7142)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 7143)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 7143)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 7144)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 7144)	This is constant WSDLIntf.SAnyType.
SArray (see page 7144)	This is constant WSDLIntf.SArray.
SArrayOf (see page 7145)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 7145)	This is constant WSDLIntf.SArrayType.
SBinding (see page 7145)	This is constant WSDLIntf.SBinding.
SBody (see page 7146)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 7146)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 7146)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 7147)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 7147)	This is constant WSDLIntf.SDynArray.
SElement (see page 7147)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 7148)	This is constant WSDLIntf.SEncodingStyle.
SFault (see page 7148)	This is constant WSDLIntf.SFault.
SHeader (see page 7148)	This is constant WSDLIntf.SHeader.
SImport (see page 7149)	This is constant WSDLIntf.SImport.
SInOut (see page 7149)	This is constant WSDLIntf.SInOut.
SInput (see page 7149)	This is constant WSDLIntf.SInput.
SLocation (see page 7150)	This is constant WSDLIntf.SLocation.
SMessage (see page 7150)	This is constant WSDLIntf.SMessage.
SName (see page 7150)	This is constant WSDLIntf.SName.
SNameSpace (see page 7151)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 7151)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 7151)	This is constant WSDLIntf.Soapns.
SOperation (see page 7152)	This is constant WSDLIntf.SOperation.
SOutput (see page 7152)	This is constant WSDLIntf.SOutput.
SPart (see page 7152)	This is constant WSDLIntf.SPart.
SParts (see page 7153)	This is constant WSDLIntf.SParts.
SPort (see page 7153)	This is constant WSDLIntf.SPort.
SPortType (see page 7153)	This is constant WSDLIntf.SPortType.
SRequest (see page 7154)	This is constant WSDLIntf.SRequest.
SResponse (see page 7154)	This is constant WSDLIntf.SResponse.
SReturn (see page 7154)	This is constant WSDLIntf.SReturn.
SSchema (see page 7155)	This is constant WSDLIntf.SSchema.
SService (see page 7155)	This is constant WSDLIntf.SService.
SSoapAction (see page 7155)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 7156)	This is constant WSDLIntf.SStyle.

STns (see page 7156)	This is constant WSDLIntf.STns.
STransport (see page 7156)	This is constant WSDLIntf.STransport.
SType (see page 7157)	This is constant WSDLIntf.SType.
STypes (see page 7157)	This is constant WSDLIntf.STypes.
SUnknown (see page 7157)	This is constant WSDLIntf.SUnknown.
SUse (see page 7158)	This is constant WSDLIntf.SUse.
SVerb (see page 7158)	This is constant WSDLIntf.SVerb.
tns (see page 7158)	This is constant WSDLIntf.tns.
Wsdlns (see page 7159)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 7159)	This is constant WSDLIntf.xsdns.

77.4.1 WSDLIntf.Directives Constant

Pascal

```
Directives: array[0..43] of string =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

C++

```
array[0..43] of string Directives =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Directives.

77.4.2 WSDLIntf.DynArrayTypes Constant

Pascal

```
DynArrayTypes: array[0..12] of string =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

C++

```
array[0..12] of string DynArrayTypes =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
```

```
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

File

WSDLIntf

Description

This is constant WSDLIntf.DynArrayTypes.

77.4.3 WSDLIntf.Httpns Constant

Pascal

```
Httpns = Wsdlns +'http/';
```

C++

```
Httpns = Wsdlns +'http/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Httpns.

77.4.4 WSDLIntf.Operators Constant

Pascal

```
Operators: array[0..12] of string = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

C++

```
array[0..12] of string Operators = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Operators.

77.4.5 WSDLIntf.OtherReservedWords Constant

Pascal

```
OtherReservedWords: array[0..0] of string = ('Application');
```

C++

```
array[0..0] of string OtherReservedWords = ('Application');
```

File

WSDLIntf

Description

This is constant WSDLIntf.OtherReservedWords.

77.4.6 WSDLIntf.ReservedWords Constant

Pascal

```
ReservedWords: array[0..66] of string = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

C++

```
array[0..66] of string ReservedWords = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ReservedWords.

77.4.7 WSDLIntf.SAddress Constant

Pascal

```
SAddress = 'address';
```

C++

```
SAddress = 'address';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAddress.

77.4.8 WSDLIntf.SAnySimpleType Constant

Pascal

```
SAnySimpleType = 'anySimpleType';
```

C++

```
SAnySimpleType = 'anySimpleType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnySimpleType.

77.4.9 WSDLIntf.SAnyType Constant

Pascal

```
SAnyType = 'anyType';
```

C++

```
SAnyType = 'anyType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnyType.

77.4.10 WSDLIntf.SArray Constant

Pascal

```
SArray = 'Array';
```

C++

```
SArray = 'Array';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArray.

77.4.11 WSDLIntf.SArrayOf Constant

Pascal

```
SArrayOf = 'ArrayOf';
```

C++

```
SArrayOf = 'ArrayOf';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayOf.

77.4.12 WSDLIntf.SArrayType Constant

Pascal

```
SArrayType = 'arrayType';
```

C++

```
SArrayType = 'arrayType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayType.

77.4.13 WSDLIntf.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBinding.

77.4.14 WSDLIntf.SBody Constant

Pascal

```
SBody = 'body';
```

C++

```
SBody = 'body';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBody.

77.4.15 WSDLIntf.ScalarArrayTypes Constant

Pascal

```
ScalarArrayTypes: array[0..12] of string =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

C++

```
array[0..12] of string ScalarArrayTypes =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ScalarArrayTypes.

77.4.16 WSDLIntf.SComplexType Constant

Pascal

```
SComplexType = 'complexType';
```

C++

```
SComplexType = 'complexType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SComplexType.

77.4.17 WSDLIntf.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDefinitions.

77.4.18 WSDLIntf.SDynArray Constant

Pascal

```
SDynArray = 'array of ';
```

C++

```
SDynArray = 'array of ';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDynArray.

77.4.19 WSDLIntf.SElement Constant

Pascal

```
SElement = 'element';
```

C++

```
SElement = 'element';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SElement.

77.4.20 WSDLIntf.SEncodingStyle Constant

Pascal

```
SEncodingStyle = 'encodingStyle';
```

C++

```
SEncodingStyle = 'encodingStyle';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SEncodingStyle.

77.4.21 WSDLIntf.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SFault.

77.4.22 WSDLIntf.SHeader Constant

Pascal

```
SHeader = 'header';
```

C++

```
SHeader = 'header';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SHeader.

77.4.23 WSDLIntf.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SImport.

77.4.24 WSDLIntf.SInOut Constant

Pascal

```
SInOut = 'inout';
```

C++

```
SInOut = 'inout';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInOut.

77.4.25 WSDLIntf.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInput.

77.4.26 WSDLIntf.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SLocation.

77.4.27 WSDLIntf.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SMessage.

77.4.28 WSDLIntf.SName Constant

Pascal

```
SName = 'name';
```

C++

```
SName = 'name';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SName.

77.4.29 WSDLIntf.SNameSpace Constant

Pascal

```
SNameSpace = 'namespace';
```

C++

```
SNameSpace = 'namespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNameSpace.

77.4.30 WSDLIntf.SNsPrefix Constant

Pascal

```
SNsPrefix = 'ns';
```

C++

```
SNsPrefix = 'ns';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNsPrefix.

77.4.31 WSDLIntf.Soapns Constant

Pascal

```
Soapns = Wsdlns +'soap/';
```

C++

```
Soapns = Wsdlns +'soap/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Soapns.

77.4.32 WSDLIntf.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOperation.

77.4.33 WSDLIntf.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOutput.

77.4.34 WSDLIntf.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPart.

77.4.35 WSDLIntf.SParts Constant

Pascal

```
SParts = 'parts';
```

C++

```
SParts = 'parts';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SParts.

77.4.36 WSDLIntf.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPort.

77.4.37 WSDLIntf.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPortType.

77.4.38 WSDLIntf.SRequest Constant

Pascal

```
SRequest = 'Request';
```

C++

```
SRequest = 'Request';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SRequest.

77.4.39 WSDLIntf.SResponse Constant

Pascal

```
SResponse = 'Response';
```

C++

```
SResponse = 'Response';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SResponse.

77.4.40 WSDLIntf.SReturn Constant

Pascal

```
SReturn = 'return';
```

C++

```
SReturn = 'return';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SReturn.

77.4.41 WSDLIntf.SSchema Constant

Pascal

```
SSchema = 'schema';
```

C++

```
SSchema = 'schema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSchema.

77.4.42 WSDLIntf.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SService.

77.4.43 WSDLIntf.SSoapAction Constant

Pascal

```
SSoapAction = 'soapAction';
```

C++

```
SSoapAction = 'soapAction';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSoapAction.

77.4.44 WSDLIntf.SStyle Constant

Pascal

```
SStyle = 'style';
```

C++

```
SStyle = 'style';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SStyle.

77.4.45 WSDLIntf.STns Constant

Pascal

```
STns = 'targetNamespace';
```

C++

```
STns = 'targetNamespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STns.

77.4.46 WSDLIntf.STransport Constant

Pascal

```
STransport = 'transport';
```

C++

```
STransport = 'transport';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STransport.

77.4.47 WSDLIntf.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SType.

77.4.48 WSDLIntf.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STypes.

77.4.49 WSDLIntf.SUnknown Constant

Pascal

```
SUnknown = 'Unknown';
```

C++

```
SUnknown = 'Unknown';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUnknown.

77.4.50 WSDLIntf.SUse Constant

Pascal

```
SUse = 'use';
```

C++

```
SUse = 'use';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUse.

77.4.51 WSDLIntf.SVerb Constant

Pascal

```
SVerb = 'verb';
```

C++

```
SVerb = 'verb';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SVerb.

77.4.52 WSDLIntf.tns Constant

Pascal

```
tns = 'http:/'+'/tempuri.org/';
```

C++

```
tns = 'http:/'+'/tempuri.org/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.tns.

77.4.53 WSDLIntf.Wsdlns Constant

Pascal

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

C++

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Wsdlns.

77.4.54 WSDLIntf.xsdns Constant

Pascal

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

C++

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.xsdns.

78 WSDLItems Namespace

This is namespace WSDLItems.

Topics

Name	Description
Classes (see page 7163)	The following table lists classes in this documentation.
Interfaces (see page 7199)	The following table lists interfaces in this documentation.
Functions (see page 7211)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7213)	The following table lists structs, records, enums in this documentation.
Types (see page 7214)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLItems (see page 7163)	TWSDLItems represents the definitions in a WSDL document.

Functions

	Name	Description
	HasDefinition (see page 7211)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 7211)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 7211)	This is function WSDLItems.NewQualifiedName.

Interfaces

	Name	Description
	IHeaderInfo (see page 7199)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 7204)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 7208)	IWSDLItems provides access to a WSDL document.

Structs, Records, Enums

	Name	Description
	IterateImportOptions (see page 7213)	This is record WSDLItems.IterateImportOptions.

Types

Name	Description
IHeaderInfoArray (see page 7214)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 7214)	This is type WSDLItems.IQualifiedNameArray.

TWSDLIterateProc (see page 7214)	This is type WSDLItems.TWSDLIterateProc.

78.1 Classes

The following table lists classes in this documentation.

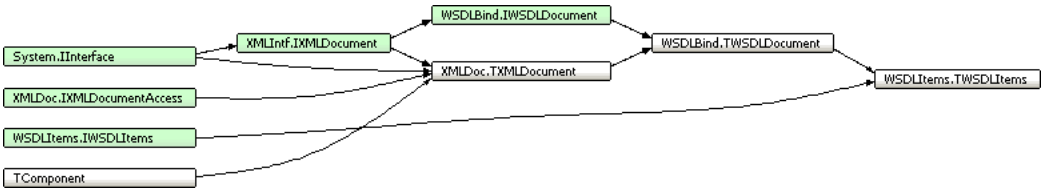
Classes

	Name	Description
	TWSDLItems (see page 7163)	TWSDLItems represents the definitions in a WSDL document.

78.1.1 TWSDLItems Class

TWSDLItems represents the definitions in a WSDL document.

Class Hierarchy



Pascal

```
TWSDLItems = class(TWSDLDocument, IWSDLItems);
```

C++

```
class TWSDLItems : public TWSDLDocument, public IWSDLItems;
```

File

WSDLItems

Description

Use TWSDLItems to represent a WSDL (Web Services Definition Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements. TWSDLItems augments its immediate ancestor (TWSDLDocument (see page 7015)) by introducing methods that provide easier access to the definitions in the WSDL document.

TWSDLItems can read an existing WSDL document from a file or server, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLItems uses an external DOM (Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLItems uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLItems is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLItems instance is automatically freed. When TWSDLItems is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

See Also

TWSDLView (see page 7223)

78.1.1.1 TWSDLItems Members

The following tables list the members exposed by TWSDLItems.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.
	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
	GetActive (see page 7313)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDefinition (see page 7021)	Returns the value of the Definition property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetName (see page 7177)	Returns the name of the definitions node in the WSDL document.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetTargetNamespace (see page 7194)	Returns the value of the TargetNamespace (see page 7198) property.
	GetTargetNamespacePrefix (see page 7195)	Returns the prefix string that represents the target namespace of the WSDL document.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7021)	This is LoadData, a member of class TWSDLDocument.
	LoadData (see page 7320)	This is LoadData, a member of class TXMLDocument.
	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.

	SetActive (see page 7327)	Sets the value of the Active property.
	SetDefinition (see page 7021)	Sets the value of the Definition property.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 7020)	Initializes the TWSDLDocument (see page 7015) object after the last constructor has executed.
	AfterConstruction (see page 7307)	Responds after the last constructor has executed.
	CompareName (see page 7171)	Indicates whether two node names refer to the same element.
	Create (see page 7169)	This is the overview for the Create constructor overload.
	Create (see page 7303)	This is the overview for the Create constructor overload.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
	Destroy (see page 7304)	Disposes of a TXMLDocument (see page 7299) object.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetBindingForServicePort (see page 7172)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 7173)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 7173)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 7174)	Fills a WideString (see page 3476) list with the names of all bindings defined in the WSDL document.

	GetBindingType (see page 7174)	Returns the port type to which a specified binding applies.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDefinition (see page 7117)	Returns the value of the Definition (see page 7118) property.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetHttpBindingAttribute (see page 7175)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 7175)	Fills a WideString (see page 3476) list with the namespaces of all documents imported by this one.
	GetLocationForImport (see page 7176)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 7176)	Returns the interface for the node that describes a message.
	GetMessages (see page 7177)	Fills a WideString (see page 3476) list with the names of all messages defined in the WSDL document.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOperationNode (see page 7177)	Returns the interface for the node that describes an operation.
	GetOperations (see page 7178)	Fills a WideString (see page 3476) list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 7178)	This is GetOperationsForBinding, a member of class TWSDLItems.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetPartNode (see page 7179)	Returns the interface for the node that describes a part of a message.
	GetParts (see page 7179)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 7180)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 7180)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 7181)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 7181)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 7182)	Returns the interface for the schema node that defines a type.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetSchemas (see page 7182)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 7183)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 7183)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 7184)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 7184)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 7185)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 7185)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 7186)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 7186)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 7187)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 7188)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 7188)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.
	GetSoapBodyInputAttribute (see page 7189)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.

	GetSoapBodyNamespace (see page 7190)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 7190)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 7191)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 7192)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 7192)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 7193)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 7193)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetWSDLDoc (see page 7195)	Returns the interface for this TWSDLItems (see page 7163) instance.
	GetWSDLItems (see page 7195)	Returns a reference to this TWSDLItems (see page 7163) instance.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	HasTypesNode (see page 7196)	Indicates whether the WSDL document includes any type definitions.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	IsPortTypeHTTPBound (see page 7196)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 7196)	Executes a callback for every WSDL document that this one imports.
	Load (see page 7197)	Loads a WSDL document and activates it.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDefinition (see page 7117)	Sets the value of the Definition (see page 7118) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.

	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
	Definition (see page 7118)	Provides access to the definitions contained in the WSDL document.
	Definition (see page 7022)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
R	StreamLoader (see page 7197)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 7198)	Provides access to an interface for loading the WSDL document.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.

	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.
	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

78.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWSDLItems.Create (TComponent) (see page 7169)	Creates and initializes a TWSDLItems (see page 7163) object.
	TWSDLItems.Create (TWSDLItems, IStreamLoader) (see page 7169)	Creates and initializes a TWSDLItems (see page 7163) object.

78.1.1.2.1 TWSDLItems.Create Constructor (TComponent)

Creates and initializes a TWSDLItems (see page 7163) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const WSDLItems: TWSDLItems; const StreamLoader: IStreamLoader); virtual;
overload;
```

C++

```
virtual __fastcall TWSDLItems(TComponent * AOwner);
virtual __fastcall TWSDLItems(const TWSDLItems WSDLItems, const IStreamLoader StreamLoader);
```

Description

Call Create to instantiate a TWSDLItems (see page 7163) component at runtime. Typically, TWSDLItems (see page 7163) instances are created by other components, such as THTTPRIO, that use them.

AOwner is another component, typically a form or data module, that is responsible for freeing the TWSDLItems (see page 7163) instance. It becomes the value of the Owner property.

WSDLItems (see page 7161) is another TWSDLItems (see page 7163) object that the new instance can use to extract a stream loader if none is specified using the StreamLoader (see page 7197) property,

StreamLoader (see page 7197) is an interface to use for loading the WSDL document from a server. It includes properties for specifying the user name, password, and proxy server needed to connect to the server that hosts the WSDL document.

If you do not specify a stream loader, the TWSDLItems (see page 7163) constructor creates one as follows: Using the first syntax, it creates a default stream loader object. Using the second syntax, it uses the stream loader from the TWSDLItems (see page 7163) instance that is the first parameter. Thus, when using the second syntax, you must supply either the WSDLItems (see page 7161) or StreamLoader (see page 7197) argument.

See Also

THTTPRio

Owner

78.1.1.3 TWSDLItems Methods

The methods of the TWSDLItems class are listed here.

Protected Methods

	Name	Description
	GetName (see page 7177)	Returns the name of the definitions node in the WSDL document.
	GetTargetNamespace (see page 7194)	Returns the value of the TargetNamespace (see page 7198) property.
	GetTargetNamespacePrefix (see page 7195)	Returns the prefix string that represents the target namespace of the WSDL document.

Public Methods

	Name	Description
	CompareName (see page 7171)	Indicates whether two node names refer to the same element.
	GetBindingForServicePort (see page 7172)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 7173)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 7173)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 7174)	Fills a WideString (see page 3476) list with the names of all bindings defined in the WSDL document.
	GetBindingType (see page 7174)	Returns the port type to which a specified binding applies.
	GetHttpBindingAttribute (see page 7175)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 7175)	Fills a WideString (see page 3476) list with the namespaces of all documents imported by this one.
	GetLocationForImport (see page 7176)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 7176)	Returns the interface for the node that describes a message.
	GetMessages (see page 7177)	Fills a WideString (see page 3476) list with the names of all messages defined in the WSDL document.
	GetOperationNode (see page 7177)	Returns the interface for the node that describes an operation.
	GetOperations (see page 7178)	Fills a WideString (see page 3476) list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 7178)	This is GetOperationsForBinding, a member of class TWSDLItems.
	GetPartNode (see page 7179)	Returns the interface for the node that describes a part of a message.

	GetParts (see page 7179)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 7180)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 7180)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 7181)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 7181)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 7182)	Returns the interface for the schema node that defines a type.
	GetSchemas (see page 7182)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 7183)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 7183)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 7184)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 7184)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 7185)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 7185)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 7186)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 7186)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 7187)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 7188)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 7188)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.
	GetSoapBodyInputAttribute (see page 7189)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.
	GetSoapBodyNamespace (see page 7190)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 7190)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 7191)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 7192)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 7192)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 7193)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 7193)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetWSDLDoc (see page 7195)	Returns the interface for this TWSDLItems (see page 7163) instance.
	GetWSDLItems (see page 7195)	Returns a reference to this TWSDLItems (see page 7163) instance.
	HasTypesNode (see page 7196)	Indicates whether the WSDL document includes any type definitions.
	IsPortTypeHTTPBound (see page 7196)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 7196)	Executes a callback for every WSDL document that this one imports.
	Load (see page 7197)	Loads a WSDL document and activates it.

78.1.1.3.1 TWSDLItems.CompareName Method

Indicates whether two node names refer to the same element.

Pascal

```
function CompareName(const NodeName: DOMString; const OtherName: DOMString; const TNS:
DOMString = ''): boolean;
```

C++

```
__fastcall boolean CompareName(const DOMString NodeName, const DOMString OtherName, const
DOMString TNS = '');
```

Description

Call CompareName to compare the names of two nodes to determine whether they match. CompareName compares the node names, taking into account any namespace prefix added to the node names. CompareName uses the namespace prefix to look up the corresponding namespace URI. If either node name does not have a namespace prefix, CompareName uses the TargetNamespace (see page 7198) property for the first node name, and the supplied namespace for the second. CompareName can find a match between two node names that have different namespace prefixes, as long as both prefixes map to the same namespace URI.

NodeName and OtherName are the two node names to compare.

TNS is the target namespace for OtherName. If this is a nonempty string, it is used to replace the namespace of OtherName. If it is an empty string, OtherName uses the namespace associated with its namespace prefix, or the TargetNamespace (see page 7198) property if it has no namespace prefix.

CompareName returns true if the two names match, false if they differ (either in the local name, or in the namespace to which the names belong).

See Also

NodeName (see page 7639)

TargetNamespace (see page 7198)

78.1.1.3.2 TWSDLItems.GetBindingForServicePort Method

Returns the name of the binding associated with a specified service and port.

Pascal

```
function GetBindingForServicePort(const ServiceName: DOMString; const PortName: DOMString):
IQualifiedName;
```

C++

```
__fastcall IQualifiedName GetBindingForServicePort(const DOMString ServiceName, const
DOMString PortName);
```

Description

Call GetBindingForServicePort to retrieve the name of a binding that is associated with the specified service and port. The binding is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

ServiceName is the name of the service to which the desired binding belongs.

PortName is the name of a port within the specified service that uses the binding.

See Also

GetBindings (see page 7174)

GetBindingType (see page 7174)

GetPortsForService (see page 7180)

GetServices (see page 7184)

GetBindingOfType (see page 7173)

GetServiceAndPortOfBinding (see page 7183)

78.1.1.3.3 TWSDLItems.GetBindingOfType Method

Returns the names of all bindings associated with a specified port type.

Pascal

```
function GetBindingOfType(const BindingTypeName: DOMString; const TNS: DOMString = ''):
IQualifiedNameArray; overload;
```

C++

```
__fastcall IQualifiedNameArray GetBindingOfType(const DOMString BindingTypeName, const
DOMString TNS = '');
```

Description

Call GetBindingOfType to retrieve the names of the bindings that are associated with the specified port type. Each bindings is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingTypeName is the type of the binding. This is the name of a port type for which the binding describes an encoding format and communications protocol.

TNS is the namespace in which BindingTypeName is defined. If this parameter is omitted, GetBindingOfType assumes the value of the TargetNamespace (see page 7198) property.

GetBindingOfType returns an array of qualified names for each binding defined for the specified type.

See Also

GetBindings (see page 7174)

GetPortTypes (see page 7181)

GetBindingType (see page 7174)

TargetNamespace (see page 7198)

78.1.1.3.4 TWSDLItems.GetBindingOperationNode Method

Returns the interface for a specified operation on a specified binding.

Pascal

```
function GetBindingOperationNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperationNode(const DOMString BindingName, const
DOMString Operation, int OverloadIndex);
```

Description

Call GetBindingOperationNode to retrieve the operation interface for a SOAP binding. GetBindingOperationNode locates a binding with the specified name, and searches its child nodes for an operation node with the specified name. If there is more than one operation with the specified name (that is, if the method is overloaded), GetBindingOperationNode locates the desired overload based on its position.

BindingName is the name of the binding. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that binding node. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node to choose if there are overloaded methods. 0 indicates the first operation node with the specified name, 1 indicates the second operation node with that name, and so on.

GetBindingOperationNode returns an IBindingOperation (see page 7034) interface, which can be used to access the name, inputs, outputs, and faults associated with the operation. If GetBindingOperationNode is unable to locate an operation node for the specified binding and operation names, it returns nil (Delphi) or NULL (C++).

See Also

GetOperationNode (see page 7177)

78.1.1.3.5 TWSDLItems.GetBindings Method

Fills a WideString (see page 3476) list with the names of all bindings defined in the WSDL document.

Pascal

```
procedure GetBindings(BindingNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetBindings(TDOMStrings BindingNames, Boolean QualifiedNames = False);
```

Description

Call GetBindings to retrieve the names of all bindings defined in the WSDL document. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingNames is an existing TWideStrings object to which the names of any defined bindings are added.

QualifiedNames indicates whether the binding names should be qualified using the target namespace prefix.

Note: GetBindings does not clear the WideString (see page 3476) list before adding the names of all defined bindings.

See Also

GetBindingForServicePort (see page 7172)

GetBindingType (see page 7174)

GetPortTypes (see page 7181)

78.1.1.3.6 TWSDLItems.GetBindingType Method

Returns the port type to which a specified binding applies.

Pascal

```
function GetBindingType(const BindingName: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetBindingType(const DOMString BindingName);
```

Description

Call GetBindingType to retrieve the names of the port type to which a specified binding applies.

BindingName is the name of a binding defined in the WSDL document.

See Also

GetBindingForServicePort (see page 7172)

GetBindings (see page 7174)

GetPortTypes (see page 7181)

GetBindingOfType (see page 7173)

78.1.1.3.7 GetHttpBindingAttribute Method

This is the overview for the GetHttpBindingAttribute method overload.

Overload List

	Name	Description
	TWSDLItems.GetHttpBindingAttribute (DOMString, DOMString) (see page 7175)	Returns the value of a specified attribute of a binding's <http:binding> node.
	TWSDLItems.GetHttpBindingAttribute (IQualifiedName, DOMString) (see page 7175)	Returns the value of a specified attribute of a binding's <http:binding> node.

78.1.1.3.7.1 TWSDLItems.GetHttpBindingAttribute Method (DOMString, DOMString)

Returns the value of a specified attribute of a binding's <http:binding> node.

Pascal

```
function GetHttpBindingAttribute(const BindingName: DOMString; Attribute: DOMString):
DOMString; overload;
function GetHttpBindingAttribute(const QualifiedName: IQualifiedName; Attribute: DOMString):
DOMString; overload;
```

C++

```
__fastcall DOMString GetHttpBindingAttribute(const DOMString BindingName, DOMString Attribute);
__fastcall DOMString GetHttpBindingAttribute(const IQualifiedName QualifiedName, DOMString
Attribute);
```

Description

Call GetHttpBindingAttribute to retrieve value of an attribute on the http binding tag of a specified binding. The http binding tag can come from the WSDL document or another document that it imports.

BindingName or QualifiedName is the name of the binding that includes the <http:binding> tag.

Attribute is the name of the desired attribute on the <http:binding> tag.

See Also

GetSoapBindingAttribute (see page 7186)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 7192)

GetSoapBodyAttribute (see page 7188)

78.1.1.3.8 TWSDLItems.GetImports Method

Fills a WideString (see page 3476) list with the namespaces of all documents imported by this one.

Pascal

```
procedure GetImports(ImportNames: TDOMStrings);
```

C++

```
__fastcall GetImports(TDOMStrings ImportNames);
```

Description

Call GetImports to retrieve the namespaces for of all documents that are imported into this one using an <import> tag. The location of these imported documents can be retrieved from the namespace using the GetLocationForImport (see page 7176) method.

ImportNames is an existing TWideStrings object to which the namespaces for any imports are added.

Note: GetImports does not clear the WideString (see page 3476) list before adding the namespaces for all imported documents.

See Also

GetLocationForImport (see page 7176)

78.1.1.3.9 TWSDLItems.GetLocationForImport Method

Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.

Pascal

```
function GetLocationForImport(const ImportNameSpace: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetLocationForImport(const DOMString ImportNameSpace);
```

Description

Call GetLocationForImport to retrieve the URI that indicates the location of an imported document, given its namespace.

ImportNameSpace is the namespace that identifies another document that is imported into this WSDL document. It is one of the strings retrieved by the GetImports (see page 7175) method.

See Also

GetImports (see page 7175)

78.1.1.3.10 TWSDLItems.GetMessageNode Method

Returns the interface for the node that describes a message.

Pascal

```
function GetMessageNode(const MessageName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetMessageNode(const DOMString MessageName);
```

Description

Call GetMessageNode to obtain information about a message defined in the WSDL document. A message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageName is the name of the message whose node is returned.

See Also

GetMessages (see page 7177)

GetParts (see page 7179)

78.1.1.3.11 TWSDLItems.GetMessages Method

Fills a WideString (see page 3476) list with the names of all messages defined in the WSDL document.

Pascal

```
procedure GetMessages(MessageNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetMessages(TDOMStrings MessageNames, Boolean QualifiedNames = False);
```

Description

Call GetMessages to retrieve the names of all messages defined in the WSDL document. Each message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageNames is an existing TWideStrings object to which the names of all messages are added.

QualifiedNames indicates whether the message names added to the list should be qualified using the target namespace prefix.

Note: GetMessages does not clear the WideString (see page 3476) list before adding the message names defined in the WSDL document.

See Also

GetMessageNode (see page 7176)

GetParts (see page 7179)

78.1.1.3.12 TWSDLItems.GetName Method

Returns the name of the definitions node in the WSDL document.

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

The root node of a WSDL document is always a definitions node. This node gives a name to the set of definitions in the WSDL document. Call GetName to retrieve that name.

See Also

Definition (see page 7022)

GetTargetNamespace (see page 7194)

78.1.1.3.13 TWSDLItems.GetOperationNode Method

Returns the interface for the node that describes an operation.

Pascal

```
function GetOperationNode(const PortTypeName: DOMString; const OperationName: DOMString):
IXMLNode;
```

C++

```
__fastcall IXMLNode GetOperationNode(const DOMString PortTypeName, const DOMString
OperationName);
```

Description

Call GetOperationNode to obtain information about an operation defined in the WSDL document. An operation corresponds to a method on the interface that corresponds to a specified port type.

PortTypeName is the name of the port type that contains the specified operation.

OperationName is the name of the operation whose node is returned.

See Also

GetOperations (see page 7178)

getoperationsforbinding (see page 7178)

GetPortTypes (see page 7181)

78.1.1.3.14 TWSDLItems.GetOperations Method

Fills a WideString (see page 3476) list with the names of all operations defined for a specified port type.

Pascal

```
procedure GetOperations(const PortTypeName: DOMString; OperationNames: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperations(const DOMString PortTypeName, TDOMStrings OperationNames, Boolean
QualifiedNames = False);
```

Description

Call GetOperations to retrieve the names of all operations defined in for the port type specified by PortTypeName. An operation corresponds to a method on the interface that corresponds to the specified port type.

PortTypeName is the name of the port type whose operations are retrieved.

OperationNames is an existing TWideStrings object to which the names of all operations on PortTypeName are added.

QualifiedNames indicates whether the operation names added to the list should be qualified using the target namespace prefix.

Note: GetOperations does not clear the WideString (see page 3476) list before adding the names of the operations on the specified port type.

See Also

GetOperationNode (see page 7177)

getoperationsforbinding (see page 7178)

GetPortTypes (see page 7181)

78.1.1.3.15 TWSDLItems.GetOperationsForBinding Method

Pascal

```
procedure GetOperationsForBinding(const BindingName: DOMString; OperationNames: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperationsForBinding(const DOMString BindingName, TDOMStrings OperationNames,
Boolean QualifiedNames = False);
```

Description

This is GetOperationsForBinding, a member of class TWSDLItems.

78.1.1.3.16 TWSDLItems.GetPartNode Method

Returns the interface for the node that describes a part of a message.

Pascal

```
function GetPartNode(const MessageName: DOMString; const PartName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPartNode(const DOMString MessageName, const DOMString PartName);
```

Description

Call GetPartNode to obtain information about a part that is defined for a message. Every message consists of one or more parts, each of which is associated with a definition from some type system.

MessageName is the name of the message that is defined using the part.

PartName is the name of the part whose node is returned.

See Also

GetParts (see page 7179)

GetMessages (see page 7177)

GetPartsForOperation (see page 7180)

78.1.1.3.17 TWSDLItems.GetParts Method

Fills a list with the names of all parts defined for a specified message.

Pascal

```
procedure GetParts(const MessageName: DOMString; PartNames: TDOMStrings; QualifiedNames:
Boolean = False);
```

C++

```
__fastcall GetParts(const DOMString MessageName, TDOMStrings PartNames, Boolean QualifiedNames
= False);
```

Description

Call GetParts to retrieve the names of all parts that make up a specified message. A part corresponds to definition from some type system.

MessageName is the name of the message whose parts are retrieved.

PartNames is an existing TWideStrings object to which the names of all parts on MessageName are added.

QualifiedNames indicates whether the part names added to the list should be qualified using the target namespace prefix.

Note: GetParts does not clear the WideString (see page 3476) list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 7179)

GetMessages (see page 7177)

GetPartsForOperation (see page 7180)

78.1.1.3.18 TWSDLItems.GetPartsForOperation Method

Fills a list with the names of all parts defined for input messages of a specified operation.

Pascal

```
function GetPartsForOperation(const PortTypeName: DOMString; const OperationName: DOMString;
OperationIndex: Integer; PartNames: TDOMStrings): Boolean;
```

C++

```
__fastcall Boolean GetPartsForOperation(const DOMString PortTypeName, const DOMString
OperationName, int OperationIndex, TDOMStrings PartNames);
```

Description

Call GetPartsForOperation to retrieve the names of all parts that make up input messages on a specified operation. The part definition must be in the WSDL document or a document that it imports. A part corresponds to definition from some type system.

PortTypeName is the name of the port type that contains the operation. A port type corresponds to an invokable interface in a Web Service application.

OperationName is the name of the operation from whose input messages the parts are retrieved. An operation corresponds to a method on the interface that corresponds to PortTypeName.

PartNames is an existing TWideStrings object to which the names of all parts on input messages (input parameters) of the operation specified by OperationName.

GetPartsForOperation returns true if it adds any part names to PartNames, false if there are no parts to add.

Note: GetPartsForOperation does not clear the WideString (see page 3476) list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 7179)

GetParts (see page 7179)

GetOperations (see page 7178)

78.1.1.3.19 TWSDLItems.GetPortsForService Method

Fills a list with the names of all ports defined within a specified service.

Pascal

```
procedure GetPortsForService(const ServiceName: DOMString; PortNames: TDOMStrings;
SkipHttpBindings: Boolean = True; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortsForService(const DOMString ServiceName, TDOMStrings PortNames, Boolean
SkipHttpBindings = True, Boolean QualifiedNames = False);
```

Description

Call GetPortsForService to retrieve the names of all ports defined for the specified service. A port represents a binding, plus the address where that binding can be found.

ServiceName is the name of the service whose ports are listed.

PortNames is an existing TWideStrings object to which the names of all ports in the specified service are added.

SkipHttpBindings indicates whether GetPortsForService should include port types whose definitions include an <http:binding> tag. When false, GetPortsForService retrieves all port type definitions. When true, GetPortsForService ignores port type definitions with simple HTTP bindings.

QualifiedNames indicates whether the port names added to the list should be qualified using the target namespace prefix.

Note: GetPortsForService does not clear the WideString (see page 3476) list before adding the names of the ports.

See Also

GetBindingForServicePort (see page 7172)

GetServices (see page 7184)

78.1.1.3.20 TWSDLItems.GetPortTypeNode Method

Returns the interface for the node that describes a port type defined in the WSDL document.

Pascal

```
function GetPortTypeNode(const PortTypeName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPortTypeNode(const DOMString PortTypeName);
```

Description

Call GetPortTypeNode to obtain information about a port type that is defined in the WSDL document or a document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeName is the name of the port type whose node is returned.

See Also

GetPortTypes (see page 7181)

78.1.1.3.21 TWSDLItems.GetPortTypes Method

Fills a list with the names of all port types defined in the WSDL document.

Pascal

```
procedure GetPortTypes(PortTypeNames: TDOMStrings; SkipHttpBindings: Boolean = True;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortTypes(TDOMStrings PortTypeNames, Boolean SkipHttpBindings = True, Boolean
QualifiedNames = False);
```

Description

Call GetPortTypes to retrieve the names of all port types defined in the WSDL document or any document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeNames is an existing TWideStrings object to which the names of all port types are added.

SkipHttpBindings indicates whether GetPortTypes should include port types whose definitions include an <http:binding> tag. When false, GetPortTypes retrieves all port type definitions. When true, GetPortTypes ignores port type definitions with simple HTTP bindings.

QualifiedNames indicates whether the port type names added to the list should be qualified using the target namespace prefix.

Note: GetPortTypes does not clear the WideString (see page 3476) list before adding the names of the port types.

See Also

GetPortTypeNode (see page 7181)

GetOperations (see page 7178)

78.1.1.3.22 TWSDLItems.GetSchemaNode Method

Returns the interface for the schema node that defines a type.

Pascal

```
function GetSchemaNode(const SchemaTns: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSchemaNode(const DOMString SchemaTns);
```

Description

Call GetSchemaNode to obtain information about a schema definition for a type that is defined in the WSDL document.

SchemaTns is the target namespace of the schema definition for a type.

GetSchemaNode returns the corresponding node.

See Also

GetSchemas (see page 7182)

HasTypesNode (see page 7196)

78.1.1.3.23 TWSDLItems.GetSchemas Method

Fills a list with the namespaces for all schema definitions of types in the WSDL document.

Pascal

```
procedure GetSchemas(SchemaNames: TDOMStrings);
```

C++

```
__fastcall GetSchemas(TDOMStrings SchemaNames);
```

Description

Call GetSchemaTypes to retrieve the namespaces that correspond to schema definitions of types.

SchemaNames is an existing TWideStrings object to which the namespaces for the schema definitions are added.

Note: GetSchemas does not clear the WideString (see page 3476) list before adding the namespaces of the schema definitions.

See Also

GetSchemaNode (see page 7182)

HasTypesNode (see page 7196)

78.1.1.3.24 TWSDLItems.GetServiceAndPortOfBinding Method

Retrieves the names of the service and port associated with a specified binding.

Pascal

```
function GetServiceAndPortOfBinding(const BindingName: IQualifiedName; var ServiceName:
DOMString; var PortName: DOMString): Boolean;
```

C++

```
__fastcall Boolean GetServiceAndPortOfBinding(const IQualifiedName BindingName, DOMString
ServiceName, DOMString PortName);
```

Description

Call GetServiceAndPortOfBinding to retrieve the name of the service and port associated with a specified binding. GetServiceAndPortOfBinding performs the converse lookup from the GetBindingForServicePort (see page 7172) method.

BindingName specifies the qualified name of the binding for which you want to look up the service and port.

ServiceName returns the name of the service to which the binding belongs.

PortName returns the name of a port within the service that uses the binding.

GetServiceAndPortOfBinding returns true if it locates the specified binding and returns a service and port. It returns false if the specified binding is not defined in the WSDL document.

See Also

GetServices (see page 7184)

GetServiceNode (see page 7183)

GetBindingForServicePort (see page 7172)

GetPortsForService (see page 7180)

GetBindingOfType (see page 7173)

GetBindings (see page 7174)

78.1.1.3.25 TWSDLItems.GetServiceNode Method

Returns the interface for the node that describes a service defined in the WSDL document.

Pascal

```
function GetServiceNode(const ServiceName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetServiceNode(const DOMString ServiceName);
```

Description

Call GetServiceNode to obtain information about a service that is defined in the WSDL document. A service is a collection of related ports.

ServiceName is the name of the service whose node is returned.

See Also

GetServices (see page 7184)

GetPortsForService (see page 7180)

GetServiceAndPortOfBinding (see page 7183)

78.1.1.3.26 TWSDLItems.GetServices Method

Fills a list with the names of all services defined in the WSDL document.

Pascal

```
procedure GetServices(ServiceNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetServices(TDOMStrings ServiceNames, Boolean QualifiedNames = False);
```

Description

Call GetServices to retrieve the names of all services defined in the WSDL document or any document that it imports. Each service is a collection of related ports.

ServiceNames is an existing TWideStrings object to which the names of all services are added.

QualifiedNames indicates whether the service names added to the list should be qualified using the target namespace prefix.

Note: GetServices does not clear the WideString (see page 3476) list before adding the names of the services.

See Also

GetServiceNode (see page 7183)

GetPortsForService (see page 7180)

GetServiceAndPortOfBinding (see page 7183)

78.1.1.3.27 TWSDLItems.GetSoapAction Method

Returns the value of the soapAction attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapAction(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapAction(const DOMString BindingName, const DOMString Operation, int
OverloadIndex);
```

Description

Call GetSoapAction to retrieve the value of the soapAction attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation is desired when the operation is overloaded.

See Also

GetSoapActionList (see page 7185)

GetBindings (see page 7174)

getoperationsforbinding (see page 7178)

GetSoapOperationAttribute (see page 7192)

78.1.1.3.28 TWSDLItems.GetSoapActionList Method

Fills a list with the soapAction attributes for all operations on a specified SOAP binding.

Pascal

```
procedure GetSoapActionList(const BindingName: DOMString; ActionList: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetSoapActionList(const DOMString BindingName, TDOMStrings ActionList, Boolean
QualifiedNames = False);
```

Description

Call GetSoapActionList to retrieve the value of the soapAction attribute of each operation of a specified SOAP binding. Each soapAction is added as a string with the format

```
OperationName = soapActionName
```

where OperationName is the name of the operation with which the soapAction attribute is associated.

BindingName is the name of the binding associated with the port type whose operations are retrieved along with their SOAP actions.

OperationNames is an existing TWideStrings object to which the names of all operations on the binding's port type are added.

QualifiedNames specifies whether the operation names should be qualified using the target namespace prefix.

Note: GetSoapActionList does not clear the WideString (see page 3476) list before adding the names of the operations and their corresponding SOAP actions.

See Also

GetSoapAction (see page 7184)

GetBindings (see page 7174)

getoperationsforbinding (see page 7178)

GetSoapOperationAttribute (see page 7192)

78.1.1.3.29 TWSDLItems.GetSoapAddressForServicePort Method

Returns the address of a specified port from a specified service.

Pascal

```
function GetSoapAddressForServicePort(const ServiceName: DOMString; const PortName:
DOMString): DOMString;
```

C++

```
__fastcall DOMString GetSoapAddressForServicePort(const DOMString ServiceName, const DOMString
PortName);
```

Description

Call GetSoapAddressForServicePort to retrieve the address to which a SOAP request should be sent if it wants to access a specified port from a specified service.

ServiceName is the name of the service that contains the desired port. It is defined in the WSDL document or another document that it imports.

PortName is the name of the port whose address is returned.

See Also

GetBindingForServicePort (see page 7172)

GetPortsForService (see page 7180)

78.1.1.3.30 TWSDLItems.GetSoapBindingAttribute Method

Returns the value of a specified attribute of a specified SOAP binding.

Pascal

```
function GetSoapBindingAttribute(const BindingName: DOMString; Attribute: DOMString):
DOMString;
```

C++

```
__fastcall DOMString GetSoapBindingAttribute(const DOMString BindingName, DOMString Attribute);
```

Description

Call GetSoapBindingAttribute to retrieve value of an attribute on a SOAP binding. GetSoapBindingAttribute locates a binding with the specified name that includes a <soap:binding> tag, and returns the specified attribute from that <soap:binding> tag.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Attribute is the name of the desired attribute on the <soap:binding> tag.

If BindingName does not identify a binding that includes a <soap:binding> tag that has the specified attribute, GetSoapBindingAttribute returns an empty string.

See Also

GetSoapBodyAttribute (see page 7188)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 7192)

GetBindingForServicePort (see page 7172)

78.1.1.3.31 TWSDLItems.GetSoapBindingInputNode Method

Returns the interface for the input node on a specified binding and operation.

Pascal

```
function GetSoapBindingInputNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingInputNode(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapBindingInputNode to retrieve the node for input messages (parameters) on an operation of a SOAP binding. GetSoapBindingInputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the input node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport

protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingInputNode is unable to locate an input node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 7188)

GetSoapBodyAttribute (see page 7188)

GetSoapBindingIONode (see page 7187)

GetBindingForServicePort (see page 7172)

78.1.1.3.32 TWSDLItems.GetSoapBindingIONode Method

Returns the interface for a node on a specified binding and operation.

Pascal

```
function GetSoapBindingIONode(const BindingName: DOMString; const Operation: DOMString; Input:
Boolean; OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingIONode(const DOMString BindingName, const DOMString
Operation, Boolean Input, int OverloadIndex);
```

Description

Call GetSoapBindingIONode to retrieve a specified node on an operation of a SOAP binding. GetSoapBindingIONode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the child of that node with a specified name. This method provides the underlying implementation for the GetSoapBindingInputNode (see page 7186) and GetSoapBindingOutputNode (see page 7188) methods.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input (see page 3895) indicates whether the child node is an input node (true) or an output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingIONode is unable to locate a node for the specified name, binding, and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 7188)

GetSoapBodyAttribute (see page 7188)

GetSoapBindingInputNode (see page 7186)

GetBindingForServicePort (see page 7172)

78.1.1.3.33 TWSDLItems.GetSoapBindingOutputNode Method

Returns the interface for the output node on a specified binding and operation.

Pascal

```
function GetSoapBindingOutputNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingOutputNode(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapBindingOutputNode to retrieve the node for output messages (output parameters or return value) on an operation of a SOAP binding. GetSoapBindingOutputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the output node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingOutputNode is unable to locate an output node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingInputNode (see page 7186)

GetSoapBodyAttribute (see page 7188)

GetSoapBindingIONode (see page 7187)

GetBindingForServicePort (see page 7172)

78.1.1.3.34 TWSDLItems.GetSoapBodyAttribute Method

Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyAttribute(const BindingName: DOMString; const Operation: DOMString; const
IOType: DOMString; const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString IOType, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyAttribute to retrieve value of an attribute on the body tag for a message type that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the message.

IOType indicates the type of the message (input or output).

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 7186)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 7192)

GetBindingForServicePort (see page 7172)

getoperationsforbinding (see page 7178)

GetInOutMessagesForOperation

GetSoapBodyInputAttribute (see page 7189)

GetSoapBodyOutputAttribute (see page 7190)

78.1.1.3.35 TWSDLItems.GetSoapBodyInputAttribute Method

Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyInputAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyInputAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyInputAttribute to retrieve value of an attribute on the body tag for an input message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the input message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyInputAttribute returns the same value as GetSoapBodyAttribute (see page 7188) with an IOType of 'input'.

See Also

GetSoapBodyAttribute (see page 7188)

GetSoapBodyOutputAttribute (see page 7190)

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 7192)

GetBindingForServicePort (see page 7172)

getoperationsforbinding (see page 7178)

GetInOutMessagesForOperation

78.1.1.3.36 TWSDLItems.GetSoapBodyNamespace Method

Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.

Pascal

```
function GetSoapBodyNamespace(const BindingPortType: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyNamespace(const DOMString BindingPortType);
```

Description

GetSoapBodyNamespace searches the bindings in the WSDL document and any documents it imports for one that has a port type that matches BindingPortType. It then searches the operations of that binding to find one whose messages include a SOAP body node. It returns the namespace attribute of that SOAP body node.

PortTypeName is the name of the port type (interface) that must appear in the binding.

See Also

GetSoapBodyAttribute (see page 7188)

GetPortTypes (see page 7181)

78.1.1.3.37 TWSDLItems.GetSoapBodyOutputAttribute Method

Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyOutputAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyOutputAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyOutputAttribute to retrieve value of an attribute on the body tag for an output message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the output message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyOutputAttribute returns the same value as GetSoapBodyAttribute (see page 7188) with an IOType of 'output'.

See Also

GetSoapBodyAttribute (see page 7188)

GetSoapBodyInputAttribute (see page 7189)

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 7192)

GetBindingForServicePort (see page 7172)

getoperationsforbinding (see page 7178)

GetInOutMessagesForOperation

78.1.1.3.38 TWSDLItems.GetSoapHeaders Method

Returns an interface for accessing a specified set of header nodes.

Pascal

```
function GetSoapHeaders(BindingName: IQualifiedName; const Operation: DOMString; Input:
Boolean; OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapHeaders(IQualifiedName BindingName, const DOMString
Operation, Boolean Input, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapHeaders to retrieve a dynamic array of interfaces for accessing information about the header nodes of a specified operation on a specified binding. This method provides the underlying implementation for the GetSoapInputHeaders (see page 7192) and GetSoapOutputHeaders (see page 7193) methods.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input (see page 3895) indicates whether the header node should belong to the operation's input node (true) or output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapHeaders does not check the message with which the header is associated.

If GetSoapHeaders is unable to locate a header node for the specified name, binding, and IOType, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 7193)

GetSoapBindingIONode (see page 7187)

GetSoapInputHeaders (see page 7192)

GetBindingForServicePort (see page 7172)

GetPartNode (see page 7179)

78.1.1.3.39 TWSDLItems.GetSoapInputHeaders Method

Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.

Pascal

```
function GetSoapInputHeaders(BindingName: IQualifiedName; const Operation: DOMString;
OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapInputHeaders(IQualifiedName BindingName, const DOMString
Operation, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapInputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the input node on a specified binding. GetSoapInputHeaders locates the specified input node using the GetSoapBindingIONode (see page 7187) method, and extracts all of its child header nodes. It then checks the message and part attributes of that input node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapInputHeaders does not check the message with which the header is associated.

If GetSoapInputHeaders is unable to locate an input header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 7193)

GetSoapBindingIONode (see page 7187)

GetSoapHeaders (see page 7191)

GetBindingForServicePort (see page 7172)

GetPartNode (see page 7179)

78.1.1.3.40 TWSDLItems.GetSoapOperationAttribute Method

Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.

Pascal

```
function GetSoapOperationAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapOperationAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapOperationAttribute to retrieve value of an attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

Attribute is the name of the desired attribute on the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 7186)

GetSoapBodyAttribute (see page 7188)

GetSoapHeaderAttribute

GetSoapOperationAttribute

GetSoapFaultsForOperation

78.1.1.3.41 TWSDLItems.GetSoapOperationStyle Method

Returns the value of the style attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapOperationStyle(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapOperationStyle(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapOperationStyle to retrieve the value of the style attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetBindings (see page 7174)

getoperationsforbinding (see page 7178)

GetSoapOperationAttribute (see page 7192)

78.1.1.3.42 TWSDLItems.GetSoapOutputHeaders Method

Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.

Pascal

```
function GetSoapOutputHeaders(BindingName: IQualifiedName; const Operation: DOMString;
OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapOutputHeaders(IQualifiedName BindingName, const DOMString
Operation, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapOutputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the output node on a specified binding. GetSoapOutputHeaders locates the specified output node using the GetSoapBindingIONode (see page 7187) method, and extracts all of its child header nodes. It then checks the message and part attributes of that output node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapOutputHeaders does not check the message with which the header is associated.

If GetSoapOutputHeaders is unable to locate an output header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapInputHeaders (see page 7192)

GetSoapBindingIONode (see page 7187)

GetSoapHeaders (see page 7191)

GetBindingForServicePort (see page 7172)

GetPartNode (see page 7179)

78.1.1.3.43 TWSDLItems.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 7198) property.

Pascal

```
function GetTargetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetTargetNamespace();
```

Description

GetTargetNamespace is the read access method for the TargetNamespace (see page 7198) property.

See Also

Definition (see page 7022)

GetTargetNamespacePrefix (see page 7195)

TargetNamespace (see page 7198)

78.1.1.3.44 TWSDLItems.GetTargetNamespacePrefix Method

Returns the prefix string that represents the target namespace of the WSDL document.

Pascal

```
function GetTargetNamespacePrefix: DOMString;
```

C++

```
__fastcall DOMString GetTargetNamespacePrefix();
```

Description

Call GetTargetNamespacePrefix to retrieve the prefix string that should be added to node names so that they explicitly use the target namespace of the WSDL document. Node names that include this prefix are defined in the namespace identified by the URL that is the value of the TargetNamespace (see page 7198) property.

See Also

TargetNamespace (see page 7198)

78.1.1.3.45 TWSDLItems.GetWSDLDoc Method

Returns the interface for this TWSDLItems (see page 7163) instance.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns the IWSDLDocument (see page 7115) for this TWSDLItems (see page 7163) instance. This interface can be used to access the Definition property, which supplies uninterpreted access to the definitions in the WSDL document.

See Also

Definition (see page 7022)

78.1.1.3.46 TWSDLItems.GetWSDLItems Method

Returns a reference to this TWSDLItems (see page 7163) instance.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the TWSDLItems (see page 7163) instance. This method is mostly used by code that holds an IWSDLItems (see page 7208) interface and needs to use the methods of the object itself.

See Also

IWSDLItems (see page 7208)

78.1.1.3.47 TWSDLItems.HasTypesNode Method

Indicates whether the WSDL document includes any type definitions.

Pascal

```
function HasTypesNode: Boolean;
```

C++

```
__fastcall Boolean HasTypesNode();
```

Description

HasTypesNode returns true if the WSDL document includes any type definitions, false otherwise.

78.1.1.3.48 TWSDLItems.IsPortTypeHTTPBound Method

Indicates whether a specified port type has an HTTP binding.

Pascal

```
function IsPortTypeHTTPBound(const PortType: DOMString): Boolean;
```

C++

```
__fastcall Boolean IsPortTypeHTTPBound(const DOMString PortType);
```

Description

Call IsPortTypeHTTPBound to determine whether there is an HTTP binding for a specified port type. A port type corresponds to an invokable interface.

PortType is the port type you want to check.

IsPortTypeHTTPBound returns true if the specified port type has an HTTP binding, false otherwise.

See Also

GetPortTypes (see page 7181)

GetPortTypeNode (see page 7181)

GetBindingOfType (see page 7173)

78.1.1.3.49 TWSDLItems.IterateImports Method

Executes a callback for every WSDL document that this one imports.

Pascal

```
procedure IterateImports(IterateProc: TWSDLIterateProc);
```

C++

```
__fastcall IterateImports(TWSDLIterateProc IterateProc);
```

Description

IterateImports is used to execute a callback before and after loading each WSDL document that this one imports, including those documents that are imported by documents that this one imports.

78.1.1.3.50 TWSDLItems.Load Method

Loads a WSDL document and activates it.

Pascal

```
procedure Load(const WSDLFileName: DOMString);
```

C++

```
__fastcall Load(const DOMString WSDLFileName);
```

Description

Load uses the stream loader specified by the StreamLoader (see page 7197) property to load a WSDL document into memory. It then sets itself to represent that WSDL document and activates it.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

See Also

StreamLoader (see page 7197)

LoadFromFile (see page 7321)

LoadFromStream (see page 7321)

LoadFromXML (see page 7322)

78.1.1.4 TWSDLItems Properties

The properties of the TWSDLItems class are listed here.

Public Properties

	Name	Description
R	StreamLoader (see page 7197)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 7198)	Provides access to an interface for loading the WSDL document.

78.1.1.4.1 TWSDLItems.StreamLoader Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property StreamLoader: IStreamLoader;
```

C++

```
__property IStreamLoader StreamLoader;
```

Description

StreamLoader is set by the TWSDLItems (see page 7163) constructor. It is an interface that specifies basic authentication information that can be used to load a WSDL document that is on another server. The IStreamLoader interface specifies the password, user name, and proxy server that can be used to connect to the server that hosts the WSDL document. TWSDLItems (see page 7163) uses this to load the WSDL document.

See Also

Create (see page 7169)

78.1.1.4.2 TWSDLItems.TargetNamespace Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property TargetNamespace: DOMString;
```

C++

```
__property DOMString TargetNamespace;
```

Description

The root node of a WSDL document is always a definitions node. This node includes a TargetNamespace attribute that uniquely identifies the set of definitions in the WSDL document. Read (see page 3727) TargetNamespace to retrieve that namespace.

See Also

Definition (see page 7022)

GetTargetNamespacePrefix (see page 7195)

78.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHeaderInfo (see page 7199)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 7204)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 7208)	IWSDLItems provides access to a WSDL document.

78.2.1 IHeaderInfo Interface

Class Hierarchy

WSDLItems.IHeaderInfo

Pascal

```
IHeaderInfo = interface;
```

C++

```
__interface IHeaderInfo;
```

File

WSDLItems

Description

This is class WSDLItems.IHeaderInfo.

78.2.1.1 IHeaderInfo Members

The following tables list the members exposed by IHeaderInfo.

Public Methods

	Name	Description
	GetEncodingStyle (see page 7200)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 7200)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 7201)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 7201)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 7201)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 7201)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 7201)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 7202)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 7202)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 7202)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 7202)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 7203)	This is SetUse, a member of class IHeaderInfo.

Public Properties

	Name	Description
	EncodingStyle (see page 7203)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 7203)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 7203)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 7204)	This is Part, a member of class IHeaderInfo.
	Required (see page 7204)	This is Required, a member of class IHeaderInfo.
	Use (see page 7204)	This is Use, a member of class IHeaderInfo.

78.2.1.2 IHeaderInfo Methods

The methods of the IHeaderInfo class are listed here.

Public Methods

	Name	Description
	GetEncodingStyle (see page 7200)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 7200)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 7201)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 7201)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 7201)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 7201)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 7201)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 7202)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 7202)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 7202)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 7202)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 7203)	This is SetUse, a member of class IHeaderInfo.

78.2.1.2.1 IHeaderInfo.GetEncodingStyle Method

Pascal

```
function GetEncodingStyle: DOMString;
```

C++

```
__fastcall DOMString GetEncodingStyle();
```

Description

This is GetEncodingStyle, a member of class IHeaderInfo.

78.2.1.2.2 IHeaderInfo.GetName Method

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

This is GetName, a member of class IHeaderInfo.

78.2.1.2.3 IHeaderInfo.GetNamespace Method

Pascal

```
function GetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetNamespace();
```

Description

This is GetNamespace, a member of class IHeaderInfo.

78.2.1.2.4 IHeaderInfo.GetPart Method

Pascal

```
function GetPart: IPart;
```

C++

```
__fastcall IPart GetPart();
```

Description

This is GetPart, a member of class IHeaderInfo.

78.2.1.2.5 IHeaderInfo.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IHeaderInfo.

78.2.1.2.6 IHeaderInfo.GetUse Method

Pascal

```
function GetUse: DOMString;
```

C++

```
__fastcall DOMString GetUse();
```

Description

This is GetUse, a member of class IHeaderInfo.

78.2.1.2.7 IHeaderInfo.SetEncodingStyle Method

Pascal

```
procedure SetEncodingStyle(const EncStyl: DOMString);
```

C++

```
__fastcall SetEncodingStyle(const DOMString EncStyl);
```

Description

This is SetEncodingStyle, a member of class IHeaderInfo.

78.2.1.2.8 IHeaderInfo.SetName Method

Pascal

```
procedure SetName(const N: DOMString);
```

C++

```
__fastcall SetName(const DOMString N);
```

Description

This is SetName, a member of class IHeaderInfo.

78.2.1.2.9 IHeaderInfo.SetNamespace Method

Pascal

```
procedure SetNamespace(const N: DOMString);
```

C++

```
__fastcall SetNamespace(const DOMString N);
```

Description

This is SetNamespace, a member of class IHeaderInfo.

78.2.1.2.10 IHeaderInfo.SetPart Method

Pascal

```
procedure SetPart(const P: IPart);
```

C++

```
__fastcall SetPart(const IPart P);
```

Description

This is SetPart, a member of class IHeaderInfo.

78.2.1.2.11 IHeaderInfo.SetRequired Method

Pascal

```
procedure SetRequired(Req: Boolean);
```

C++

```
__fastcall SetRequired(Boolean Req);
```

Description

This is SetRequired, a member of class IHeaderInfo.

78.2.1.2.12 IHeaderInfo.SetUse Method

Pascal

```
procedure SetUse(const U: DOMString);
```

C++

```
__fastcall SetUse(const DOMString U);
```

Description

This is SetUse, a member of class IHeaderInfo.

78.2.1.3 IHeaderInfo Properties

The properties of the IHeaderInfo class are listed here.

Public Properties

	Name	Description
	EncodingStyle (see page 7203)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 7203)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 7203)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 7204)	This is Part, a member of class IHeaderInfo.
	Required (see page 7204)	This is Required, a member of class IHeaderInfo.
	Use (see page 7204)	This is Use, a member of class IHeaderInfo.

78.2.1.3.1 IHeaderInfo.EncodingStyle Property

Pascal

```
property EncodingStyle: DOMString;
```

C++

```
__property DOMString EncodingStyle;
```

Description

This is EncodingStyle, a member of class IHeaderInfo.

78.2.1.3.2 IHeaderInfo.Name Property

Pascal

```
property Name: DOMString;
```

C++

```
__property DOMString Name;
```

Description

This is Name, a member of class IHeaderInfo.

78.2.1.3.3 IHeaderInfo.Namespace Property

Pascal

```
property Namespace: DOMString;
```

C++

```
__property DOMString Namespace;
```

Description

This is Namespace, a member of class IHeaderInfo.

78.2.1.3.4 IHeaderInfo.Part Property

Pascal

```
property Part: IPart;
```

C++

```
__property IPart Part;
```

Description

This is Part, a member of class IHeaderInfo.

78.2.1.3.5 IHeaderInfo.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IHeaderInfo.

78.2.1.3.6 IHeaderInfo.Use Property

Pascal

```
property Use: DOMString;
```

C++

```
__property DOMString Use;
```

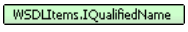
Description

This is Use, a member of class IHeaderInfo.

78.2.2 IQualifiedName Interface

IQualifiedName is the interface for obtaining an information about a name in an XML document.

Class Hierarchy



Pascal

```
IQualifiedName = interface;
```

C++

```
__interface IQualifiedName;
```

File

WSDLItems

Description

IQualifiedName introduces two properties, Name (see page 7207) and Namespace (see page 7207), which represent the local name and namespace URI for a name that is used in an XML document. Using IQualifiedName makes it easier to work with names where it is important to know the namespace in which the name is defined.

Note: C++ method declarations that use IQualifiedName use the _di_IQualifiedName type instead. This type is a DelphiInterface (see page 3263) wrapper around the IQualifiedName interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IQualifiedName >  _di_IQualifiedName;
```

See Also

TWSDLItems (see page 7163)

TMemoryStream

78.2.2.1 IQualifiedName Members

The following tables list the members exposed by IQualifiedName.

Public Methods

	Name	Description
	GetName (see page 7205)	Returns the value of the Name (see page 7207) property.
	GetNamespace (see page 7206)	Returns the value of the Namespace (see page 7207) property.
	SetName (see page 7206)	Sets the value of the Name (see page 7207) property.
	SetNamespace (see page 7206)	Sets the value of the Namespace (see page 7207) property.

Public Properties

	Name	Description
	Name (see page 7207)	Specifies the local name.
	Namespace (see page 7207)	Specifies the namespace in which the name is defined.

78.2.2.2 IQualifiedName Methods

The methods of the IQualifiedName class are listed here.

Public Methods

	Name	Description
	GetName (see page 7205)	Returns the value of the Name (see page 7207) property.
	GetNamespace (see page 7206)	Returns the value of the Namespace (see page 7207) property.
	SetName (see page 7206)	Sets the value of the Name (see page 7207) property.
	SetNamespace (see page 7206)	Sets the value of the Namespace (see page 7207) property.

78.2.2.2.1 IQualifiedName.GetName Method

Returns the value of the Name (see page 7207) property.

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

GetName is the read implementation for the Name (see page 7207) property.

See Also

Name (see page 7207)

78.2.2.2.2 IQualifiedName.GetNamespace Method

Returns the value of the Namespace (see page 7207) property.

Pascal

```
function GetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetNamespace();
```

Description

GetNamespace is the read implementation for the Namespace (see page 7207) property.

See Also

Namespace (see page 7207)

78.2.2.2.3 IQualifiedName.SetName Method

Sets the value of the Name (see page 7207) property.

Pascal

```
procedure SetName(const name: DOMString);
```

C++

```
__fastcall SetName(const DOMString name);
```

Description

SetName is the write implementation for the Name (see page 7207) property.

Name (see page 7207) is the new name.

See Also

Name (see page 7207)

78.2.2.2.4 IQualifiedName.SetNamespace Method

Sets the value of the Namespace (see page 7207) property.

Pascal

```
procedure SetNamespace(const ns: DOMString);
```

C++

```
__fastcall SetNamespace(const DOMString ns);
```

Description

SetNamespace is the write implementation for the Namespace (see page 7207) property.

Namespace (see page 7207) is the new namespace in which to interpret Name (see page 7207).

See Also

Namespace (see page 7207)

78.2.2.3 IQualifiedName Properties

The properties of the IQualifiedName class are listed here.

Public Properties

	Name	Description
	Name (see page 7207)	Specifies the local name.
	Namespace (see page 7207)	Specifies the namespace in which the name is defined.

78.2.2.3.1 IQualifiedName.Name Property

Specifies the local name.

Pascal

```
property Name: DOMString;
```

C++

```
__property DOMString Name;
```

Description

Name specifies the local name portion of the qualified name. This name is defined within the context of the namespace that the Namespace (see page 7207) property specifies.

See Also

Namespace (see page 7207)

78.2.2.3.2 IQualifiedName.Namespace Property

Specifies the namespace in which the name is defined.

Pascal

```
property Namespace: DOMString;
```

C++

```
__property DOMString Namespace;
```

Description

Namespace specifies the namespace that is used to interpret the value of the Name (see page 7207) property.

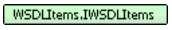
See Also

Name (see page 7207)

78.2.3 IWSDLItems Interface

IWSDLItems provides access to a WSDL document.

Class Hierarchy



Pascal

```
IWSDLItems = interface;
```

C++

```
__interface IWSDLItems;
```

File

WSDLItems

Description

IWSDLItems is an interface to the TWSDLItems (see page 7163) class. You can use this interface to load the WSDL document from a file or server, obtain access to the TWSDLItems (see page 7163) object whose methods provide information about the definitions in that document, and obtain an IWSDLDocument (see page 7115) interface for low-level access to the definitions in the document.

Note: C++ method declarations that use IWSDLItems use the _di_IWSDLItems type instead. This type is a DelphiInterface (see page 3263) wrapper around the IWSDLItems interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLItems (see page 7163)

IWSDLItems

78.2.3.1 IWSDLItems Members

The following tables list the members exposed by IWSDLItems.

Public Methods

	Name	Description
	GetWSDLDoc (see page 7209)	Returns the IWSDLDocument (see page 7115) interface for the WSDL document.
	GetWSDLItems (see page 7209)	Returns a reference to the TWSDLItems (see page 7163) instance that implements this interface.
	Load (see page 7209)	Loads a WSDL document so that its definitions are available.

78.2.3.2 IWSDLItems Methods

The methods of the IWSDLItems class are listed here.

Public Methods

	Name	Description
	GetWSDLDoc (see page 7209)	Returns the IWSDLDocument (see page 7115) interface for the WSDL document.
	GetWSDLItems (see page 7209)	Returns a reference to the TWSDLItems (see page 7163) instance that implements this interface.
	Load (see page 7209)	Loads a WSDL document so that its definitions are available.

78.2.3.2.1 IWSDLItems.GetWSDLDoc Method

Returns the IWSDLDocument (see page 7115) interface for the WSDL document.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns a IWSDLDocument (see page 7115) interface, which supplies uninterpreted access to the definitions in the WSDL document. To access these definitions more easily, use the GetWSDLItems (see page 7209) method, which returns a reference to the underlying TWSDLItems (see page 7163) object.

See Also

GetWSDLItems (see page 7209)

78.2.3.2.2 IWSDLItems.GetWSDLItems Method

Returns a reference to the TWSDLItems (see page 7163) instance that implements this interface.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the underlying TWSDLItems (see page 7163) instance. This object has numerous methods that let you access the definitions in the WSDL document.

78.2.3.2.3 IWSDLItems.Load Method

Loads a WSDL document so that its definitions are available.

Pascal

```
procedure Load(const WSDLFileName: DOMString);
```

C++

```
__fastcall Load(const DOMString WSDLFileName);
```

Description

Load reads and interprets a specified WSDL document. This method should be called before you attempt to access the definitions in the WSDL document.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

78.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	HasDefinition (see page 7211)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 7211)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 7211)	This is function WSDLItems.NewQualifiedName.

78.3.1 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLDoc: IWSDLDocument): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLDocument WSDLDoc);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

78.3.2 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLItems: IWSDLItems): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLItems WSDLItems);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

78.3.3 WSDLItems.NewQualifiedName Function

Pascal

```
function NewQualifiedName(const Name: DOMString = ''; const Namespace: DOMString = ''):
```

```
  IQualifiedName;
```

C++

```
  IQualifiedName NewQualifiedName(const DOMString Name = '', const DOMString Namespace = '');
```

File

WSDLItems

Description

This is function WSDLItems.NewQualifiedName.

78.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	IterateImportOptions (see page 7213)	This is record WSDLItems.IterateImportOptions.

78.4.1 WSDLItems.IterateImportOptions Enumeration

Pascal

```
IterateImportOptions = (
  ioBeforeLoad,
  ioLoaded
);
```

C++

```
enum IterateImportOptions {
  ioBeforeLoad,
  ioLoaded
};
```

File

WSDLItems

Description

This is record WSDLItems.IterateImportOptions.

78.5 Types

The following table lists types in this documentation.

Types

Name	Description
IHeaderInfoArray (see page 7214)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 7214)	This is type WSDLItems.IQualifiedNameArray.
TWSDLIterateProc (see page 7214)	This is type WSDLItems.TWSDLIterateProc.

78.5.1 WSDLItems.IHeaderInfoArray Type

Pascal

```
IHeaderInfoArray = array of IHeaderInfo;
```

C++

```
array of IHeaderInfo IHeaderInfoArray;
```

File

WSDLItems

Description

This is type WSDLItems.IHeaderInfoArray.

78.5.2 WSDLItems.IQualifiedNameArray Type

Pascal

```
IQualifiedNameArray = array of IQualifiedName;
```

C++

```
array of IQualifiedName IQualifiedNameArray;
```

File

WSDLItems

Description

This is type WSDLItems.IQualifiedNameArray.

78.5.3 WSDLItems.TWSDLIterateProc Type

Pascal

```
TWSDLIterateProc = procedure (const Options: IterateImportOptions; const WSDLItems:
```

```
IWSDLItems; const XMLSchemaDoc: IXMLSchemaDoc; const Name: DOMString) of object;
```

C++

```
(const Options: IterateImportOptions; const WSDLItems: IWSDLItems; const XMLSchemaDoc:
IXMLSchemaDoc; const Name: DOMString) ( TWSDLIterateProc)();
```

File

WSDLItems

Description

This is type WSDLItems.TWSDLIterateProc.

79 WSDLNode Namespace

This is namespace WSDLNode.

Topics

Name	Description
Classes (see page 7218)	The following table lists classes in this documentation.
Functions (see page 7229)	The following table lists functions in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 7218)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 7218)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 7223)	TWSDLView represents a specific binding and operation in a WSDL (see page 7228) document.

Functions

	Name	Description
	ActivateWSDL (see page 7229)	This is function WSDLNode.ActivateWSDL.

79.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 7218)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 7218)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 7223)	TWSDLView represents a specific binding and operation in a WSDL (see page 7228) document.

79.1.1 EWSDLLoadException Class

Class Hierarchy

Pascal

```
EWSDLLoadException = class(Exception);
```

C++

```
class EWSDLLoadException : public Exception;
```

File

WSDLNode

Description

This is class WSDLNode.EWSDLLoadException.

79.1.2 TWSDLClientNode Class

Class Hierarchy

Pascal

```
TWSDLClientNode = class(TComponent, IWebNode);
```

C++

```
class TWSDLClientNode : public TComponent, public IWebNode;
```

File

WSDLNode

Description

This is class WSDLNode.TWSDLClientNode.

79.1.2.1 TWSDLClientNode Members

The following tables list the members exposed by TWSDLClientNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 7221)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 7222)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	BeforeExecute (see page 7219)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6584)	This is the overview for the Execute method overload.
	Execute (see page 7220)	This is the overview for the Execute method overload.
	Notification (see page 7221)	Forwards notification messages to all owned components.

Public Properties

	Name	Description
	MimeBoundary (see page 6585)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Published Properties

	Name	Description
	TransportNode (see page 7222)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 7223)	This is WSDLView, a member of class TWSDLClientNode.

79.1.2.2 TWSDLClientNode Methods

The methods of the TWSDLClientNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 7221)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 7222)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	BeforeExecute (see page 7219)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 7220)	This is the overview for the Execute method overload.
	Notification (see page 7221)	Forwards notification messages to all owned components.

79.1.2.2.1 TWSDLClientNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 6584)

OnBeforeExecute

79.1.2.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TWSDLClientNode.Execute (String, TStream) (see page 7220)	Executes an encoded method call and returns the encoded results in a stream.
	TWSDLClientNode.Execute (TStream) (see page 7221)	This is Execute, a member of class TWSDLClientNode.
	TWSDLClientNode.Execute (TStream, TStream) (see page 7221)	This is Execute, a member of class TWSDLClientNode.

79.1.2.2.2.1 TWSDLClientNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); overload;
```

C++

```
__fastcall Execute(const UnicodeString DataMsg, TStream Resp);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert

BeforeExecute (see page 6583)

79.1.2.2.2.2 TWSDLClientNode.Execute Method (TStream)

Pascal

```
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall TStream Execute(const TStream Request);
```

Description

This is Execute, a member of class TWSDLClientNode.

79.1.2.2.2.3 TWSDLClientNode.Execute Method (TStream, TStream)

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); overload;
```

C++

```
__fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TWSDLClientNode.

79.1.2.2.3 TWSDLClientNode.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 6585)

79.1.2.2.4 TWSDLClientNode.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

79.1.2.2.5 TWSDLClientNode.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

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

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 6585)

79.1.2.3 TWSDLClientNode Properties

The properties of the TWSDLClientNode class are listed here.

Published Properties

	Name	Description
	TransportNode (see page 7222)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 7223)	This is WSDLView, a member of class TWSDLClientNode.

79.1.2.3.1 TWSDLClientNode.TransportNode Property

Pascal

```
property TransportNode: IWebNode;
```

C++

```
__property IWebNode TransportNode;
```

Description

This is TransportNode, a member of class TWSDLClientNode.

79.1.2.3.2 TWSDLClientNode.WSDLView Property

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

This is WSDLView, a member of class TWSDLClientNode.

79.1.3 TWSDLView Class

TWSDLView represents a specific binding and operation in a WSDL (see page 7228) document.

Class Hierarchy

Pascal

```
TWSDLView = class(TComponent);
```

C++

```
class TWSDLView : public TComponent;
```

File

WSDLNode

Description

TWSDLView is used by components that work with WSDL (see page 7228) documents to represent a specific port and/or operation that is defined in a WSDL (see page 7228) document. The WSDL (see page 7228) property provides access to the WSDL (see page 7228) document itself. The other properties give the names of the entities of interest that are defined within that document.

See Also

THTTPReqResp (see page 1214)

TOPToSoapDomConvert

TWSDLItems (see page 7163)

79.1.3.1 TWSDLView Members

The following tables list the members exposed by TWSDLView.

Public Fields

	Name	Description
	IntfInfo (see page 7224)	This is IntfInfo, a member of class TWSDLView.

Public Methods

	Name	Description
	Activate (see page 7225)	This is Activate, a member of class TWSDLView.
	SetDesignState (see page 7225)	This is SetDesignState, a member of class TWSDLView.

Published Properties

	Name	Description
	Operation (see page 7225)	Identifies the target operation that is defined in the WSDL (see page 7228) document.
	Password (see page 7226)	This is Password, a member of class TWSDLView.
	Port (see page 7226)	Specifies the target port that is defined in the WSDL (see page 7228) document.
	PortType (see page 7226)	Specifies the target port type that is defined in the WSDL (see page 7228) document.
	Proxy (see page 7227)	This is Proxy, a member of class TWSDLView.
	Service (see page 7227)	Specifies the target service that is defined in the WSDL (see page 7228) document.
	UserName (see page 7227)	This is UserName, a member of class TWSDLView.
	WSDL (see page 7228)	Provides access to the definitions in the WSDL document.

79.1.3.2 TWSDLView Fields

The fields of the TWSDLView class are listed here.

Public Fields

	Name	Description
	IntfInfo (see page 7224)	This is IntfInfo, a member of class TWSDLView.

79.1.3.2.1 TWSDLView.IntfInfo Field

Pascal

```
IntfInfo: PTypeInfo;
```

C++

```
PTypeInfo IntfInfo;
```

Description

This is IntfInfo, a member of class TWSDLView.

79.1.3.3 TWSDLView Methods

The methods of the TWSDLView class are listed here.

Public Methods

	Name	Description
	Activate (see page 7225)	This is Activate, a member of class TWSDLView.
	SetDesignState (see page 7225)	This is SetDesignState, a member of class TWSDLView.

79.1.3.3.1 TWSDLView.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TWSDLView.

79.1.3.3.2 TWSDLView.SetDesignState Method

Pascal

```
procedure SetDesignState(Designing: Boolean);
```

C++

```
__fastcall SetDesignState(Boolean Designing);
```

Description

This is SetDesignState, a member of class TWSDLView.

79.1.3.4 TWSDLView Properties

The properties of the TWSDLView class are listed here.

Published Properties

	Name	Description
	Operation (see page 7225)	Identifies the target operation that is defined in the WSDL (see page 7228) document.
	Password (see page 7226)	This is Password, a member of class TWSDLView.
	Port (see page 7226)	Specifies the target port that is defined in the WSDL (see page 7228) document.
	PortType (see page 7226)	Specifies the target port type that is defined in the WSDL (see page 7228) document.
	Proxy (see page 7227)	This is Proxy, a member of class TWSDLView.
	Service (see page 7227)	Specifies the target service that is defined in the WSDL (see page 7228) document.
	UserName (see page 7227)	This is UserName, a member of class TWSDLView.
	WSDL (see page 7228)	Provides access to the definitions in the WSDL document.

79.1.3.4.1 TWSDLView.Operation Property

Identifies the target operation that is defined in the WSDL (see page 7228) document.

Pascal

```
property Operation: string;
```

C++

```
__property UnicodeString Operation;
```

Description

Operation is the name of an operation that is defined in the WSDL (see page 7228) document identified by the WSDL (see

page 7228) property. Operation maps to a method on the interface that corresponds to the PortType (see page 7226) property.

See Also

PortType (see page 7226)

WSDL (see page 7228)

79.1.3.4.2 TWSDLView.Password Property

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

This is Password, a member of class TWSDLView.

79.1.3.4.3 TWSDLView.Port Property

Specifies the target port that is defined in the WSDL (see page 7228) document.

Pascal

```
property Port: string;
```

C++

```
__property UnicodeString Port;
```

Description

Port is the name of a port that is defined in the WSDL (see page 7228) document identified by the WSDL (see page 7228) property. A Port represents the address of a Web Service (see page 7227) application, a transport protocol that is used for requests to that application, the data format to use for encoding requests, and the port type that defines the operations contained in those requests.

See Also

PortType (see page 7226)

WSDL (see page 7228)

Service (see page 7227)

79.1.3.4.4 TWSDLView.PortType Property

Specifies the target port type that is defined in the WSDL (see page 7228) document.

Pascal

```
property PortType: string;
```

C++

```
__property UnicodeString PortType;
```

Description

Port (see page 7226) is the name of a port type that is defined in the WSDL (see page 7228) document identified by the WSDL (see page 7228) property. A port type corresponds to an interface that the Web Service (see page 7227) application

implements.

The target method on that interface is identified by the Operation (see page 7225) property. The address and protocols to use when calling this interface are identified by the Port (see page 7226) property.

See Also

WSDL (see page 7228)

Port (see page 7226)

Operation (see page 7225)

79.1.3.4.5 TWSDLView.Proxy Property

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

This is Proxy, a member of class TWSDLView.

79.1.3.4.6 TWSDLView.Service Property

Specifies the target service that is defined in the WSDL (see page 7228) document.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Service is the name of the service of interest that is defined in the WSDL (see page 7228) document identified by the WSDL (see page 7228) property. A service is a collection of related ports, including the one specified by the Port (see page 7226) property.

See Also

Port (see page 7226)

WSDL (see page 7228)

79.1.3.4.7 TWSDLView.UserName Property

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

This is UserName, a member of class TWSDLView.

79.1.3.4.8 TWSDLView.WSDL Property

Provides access to the definitions in the WSDL document.

Pascal

```
property WSDL: TWSDLItems;
```

C++

```
__property TWSDLItems WSDL;
```

Description

WSDLItems (see page 7161) is a helper object whose methods return the definitions included in the WSDL document. All the other TWSDLView (see page 7223) properties refer to definitions contained in this WSDL document

79.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ActivateWSDL (see page 7229)	This is function WSDLNode.ActivateWSDL.

79.2.1 WSDLNode.ActivateWSDL Function

Pascal

```
function ActivateWSDL(WSDL: TWSDLItems; const Name: string; const Password: string; const
Proxy: string): Boolean;
```

C++

```
Boolean ActivateWSDL(TWSDLItems WSDL, const UnicodeString Name, const UnicodeString Password,
const UnicodeString Proxy);
```

File

WSDLNode

Description

This is function WSDLNode.ActivateWSDL.

80 WSDLPub Namespace

This is namespace WSDLPub.

Topics

Name	Description
Classes (see page 7232)	The following table lists classes in this documentation.
Interfaces (see page 7247)	The following table lists interfaces in this documentation.
Functions (see page 7249)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7250)	The following table lists structs, records, enums in this documentation.
Types (see page 7251)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 7232)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 7242)	This is class WSDLPub.TWSDLPublish.

Functions

	Name	Description
	WSDLPubFactory (see page 7249)	This is function WSDLPub.WSDLPubFactory.

Interfaces

	Name	Description
	IWSDLPublish (see page 7247)	This is class WSDLPub.IWSDLPublish.

Structs, Records, Enums

	Name	Description
	SOAPPublishOptions (see page 7250)	This is record WSDLPub.SOAPPublishOptions.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 7251)	This is type WSDLPub.TBeforePublishingWSDLEvent.
TSOAPPublishOptions (see page 7251)	This is type WSDLPub.TSOAPPublishOptions.

80.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 7232)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 7242)	This is class WSDLPub.TWSDLPublish.

80.1.1 TWSDLHTMLPublish Class

TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.

Class Hierarchy

Pascal

```
TWSDLHTMLPublish = class(TComponent, IWebDispatch);
```

C++

```
class TWSDLHTMLPublish : public TComponent, public IWebDispatch;
```

File

WSDLPub

Description

Add TWSDLHTMLPublish to the Web module of a Web Service application to publish a list of WSDL documents that describe how to call that Web Service application. By publishing WSDL documents for your Web Service application, you allow arbitrary clients to call on your Web Services.

TWSDLHTMLPublish publishes definitions it derives from all the invokable interfaces that are registered with the invocation registry (InvRegistry), as well as from all remotable classes and types that are registered with the remotable type registry (RemTypeRegistry). By default, it uses the address where you install the application that contains the TWSDLHTMLPublish component as the address where those services are available. If you deploy the implementation of those Web Services elsewhere, you can enable the WSDL administrator (using the AdminEnabled (see page 7239) property), and then launch the WSDL administrator from a Web browser to change the addresses specified in the generated WSDL documents.

TWSDLHTMLPublish automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the WSDL publisher without requiring the use of Web action items. Those messages include requests for the list of WSDL documents and requests for the WSDL administrator.

See Also

TWebModule

InvRegistry

RemTypeRegistry

Generating WSDL Documents for a Web Service Application

Using the SOAP Application Wizard

Writing Servers that Support Web Services

80.1.1.1 TWSDLHTMLPublish Members

The following tables list the members exposed by TWSDLHTMLPublish.

Protected Methods

	Name	Description
	AddInterfaceList (see page 7235)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 7235)	This is AddPortList, a member of class TWSDLHTMLPublish.
	GetHostScriptBaseURL (see page 7237)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	UpdatePortList (see page 7238)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	WSILInfo (see page 7238)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	Create (see page 7233)	Creates an instance of TWSDLHTMLPublish (see page 7232).
	Destroy (see page 7234)	Destroys the instance of a TWSDLHTMLPublish (see page 7232) object.
	DispatchEnabled (see page 7235)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 7235)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 7236)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 7236)	Responds to an incoming message and generates the appropriate response message.
	GetTargetNamespace (see page 7237)	Returns the value of the TargetNamespace (see page 7240) property.
	HandleRequest (see page 7237)	Generates the response for an incoming request message.
	ServiceInfo (see page 7238)	Responds to requests that are otherwise not handled by the application.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 7241)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 7241)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 7242)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 7242)	Occurs before the publisher generates the schema for a given type.

Published Properties

	Name	Description
	AdminEnabled (see page 7239)	Enables or disables the WSDL administrator.
	PublishOptions (see page 7240)	Controls the definitions TWSDLHTMLPublish (see page 7232) creates when it generates its WSDL documents.
	TargetNamespace (see page 7240)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 7240)	This is WebDispatch, a member of class TWSDLHTMLPublish.

80.1.1.2 TWSDLHTMLPublish.Create Constructor

Creates an instance of TWSDLHTMLPublish (see page 7232).

Pascal

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

C++

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

Description

Call Create to instantiate a TWSDLHTMLPublish (see page 7232) object at runtime. TWSDLHTMLPublish (see page 7232) components added to web modules at design time are created automatically.

The AOwner parameter specifies another component (usually the web module) that is responsible for freeing the TWSDLHTMLPublish (see page 7232) instance. It becomes the value of the Owner property.

See Also

Owner

80.1.1.3 TWSDLHTMLPublish.Destroy Destructor

Destroys the instance of a TWSDLHTMLPublish (see page 7232) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWSDLHTMLPublish();
```

Description

Use the Free method to free a TWSDLHTMLPublish (see page 7232) instance. Unlike the Destroy method, Free guards against a nil reference.

See Also

System::TObject::Free (see page 3382)

80.1.1.4 TWSDLHTMLPublish Methods

The methods of the TWSDLHTMLPublish class are listed here.

Protected Methods

	Name	Description
	AddInterfaceList (see page 7235)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 7235)	This is AddPortList, a member of class TWSDLHTMLPublish.
	GetHostScriptBaseURL (see page 7237)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	UpdatePortList (see page 7238)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	WSILInfo (see page 7238)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	DispatchEnabled (see page 7235)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 7235)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 7236)	Returns the method type of all HTTP messages to which this component responds.

	DispatchRequest (see page 7236)	Responds to an incoming message and generates the appropriate response message.
	GetTargetNamespace (see page 7237)	Returns the value of the TargetNamespace (see page 7240) property.
	HandleRequest (see page 7237)	Generates the response for an incoming request message.
	ServiceInfo (see page 7238)	Responds to requests that are otherwise not handled by the application.

80.1.1.4.1 TWSDLHTMLPublish.AddInterfaceList Method

Pascal

```
procedure AddInterfaceList(htmldoc: TStringList; const WSDLBaseURL: String);
```

C++

```
__fastcall AddInterfaceList(TStringList htmldoc, const UnicodeString WSDLBaseURL);
```

Description

This is AddInterfaceList, a member of class TWSDLHTMLPublish.

80.1.1.4.2 TWSDLHTMLPublish.AddPortList Method

Pascal

```
procedure AddPortList(htmldoc: TStringList; const PortType: String);
```

C++

```
__fastcall AddPortList(TStringList htmldoc, const UnicodeString PortType);
```

Description

This is AddPortList, a member of class TWSDLHTMLPublish.

80.1.1.4.3 TWSDLHTMLPublish.DispatchEnabled Method

Indicates whether the WSDL publisher is enabled to respond to incoming messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In TWSDLHTMLPublish (see page 7232), DispatchEnabled always returns true.

See Also

webdispatch (see page 7240)

80.1.1.4.4 TWSDLHTMLPublish.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns a Mask derived from the Mask property of the WebDispatch (see page 7240) property.

See Also

webdispatch (see page 7240)

Mask

80.1.1.4.5 TWSDLHTMLPublish.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to this component. It returns the MethodType properties of WebDispatch (see page 7240).

See Also

MethodType

webdispatch (see page 7240)

80.1.1.4.6 TWSDLHTMLPublish.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of the WebDispatch (see page 7240) property. DispatchRequest responds by generating an HTML page that lists the WSDL documents describing the Web Service, and including a button for the WSDL administrator if the AdminEnabled (see page 7239) property is true.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully generates a response. It returns false if it could not handle the request.

See Also

webdispatch (see page 7240)

AdminEnabled (see page 7239)

80.1.1.4.7 TWSDLHTMLPublish.GetHostScriptBaseURL Method

Pascal

```
function GetHostScriptBaseURL(Request: TWebRequest): String;
```

C++

```
__fastcall UnicodeString GetHostScriptBaseURL(TWebRequest Request);
```

Description

This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.

80.1.1.4.8 TWSDLHTMLPublish.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 7240) property.

Pascal

```
function GetTargetNamespace: InvString;
```

C++

```
__fastcall InvString GetTargetNamespace();
```

Description

GetTargetNamespace is the read implementation for the TargetNamespace (see page 7240) property.

See Also

TargetNamespace (see page 7240)

80.1.1.4.9 TWSDLHTMLPublish.HandleRequest Method

Generates the response for an incoming request message.

Pascal

```
function HandleRequest(Resp: TStringList; const Path: String; const HostScriptBaseURL: String;
var ContentType: String; Request: TWebRequest): integer;
```

C++

```
__fastcall int HandleRequest(TStringList Resp, const UnicodeString Path, const UnicodeString
HostScriptBaseURL, UnicodeString ContentType, TWebRequest Request);
```

Description

The DispatchRequest (see page 7236) method calls HandleRequest to generate the content of the HTTP response message it fills out. Typically, applications should not need to call this method.

Resp is a string list to which HandleRequest writes the content of the response.

Path is the path information taken from the incoming request message.

HostScriptBaseURL is the URL for the script name to which the incoming request is addressed.

ContentType returns the content type of the response that is written to Resp. This becomes the value of a ContentType header on the response message.

Request is the request message for which this method generates a response.

HandleRequest returns the StatusCode for the generated response. This becomes the StatusCode property of the response message.

See Also

DispatchRequest (see page 7236)

StatusCode

80.1.1.4.10 TWSDLHTMLPublish.ServiceInfo Method

Responds to requests that are otherwise not handled by the application.

Pascal

```
procedure ServiceInfo(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var
Handled: Boolean);
```

C++

```
__fastcall ServiceInfo(TObject * Sender, TWebRequest Request, TWebResponse Response, Boolean
Handled);
```

Description

The Web Module (or Web Dispatcher) calls ServiceInfo when there are no other components in the application to handle an incoming request message. ServiceInfo responds by generating a default Web page that includes a link to the WSDL documents.

Sender is not used in the ServiceInfo method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that ServiceInfo successfully generated a response.

See Also

DispatchRequest (see page 7236)

80.1.1.4.11 TWSDLHTMLPublish.UpdatePortList Method

Pascal

```
procedure UpdatePortList(PortList: TStrings; const PortType: String; const Command: String);
```

C++

```
__fastcall UpdatePortList(TStrings PortList, const UnicodeString PortType, const UnicodeString
Command);
```

Description

This is UpdatePortList, a member of class TWSDLHTMLPublish.

80.1.1.4.12 TWSDLHTMLPublish.WSILInfo Method

Returns the WS inspection document that describes the Web Service.

Pascal

```
procedure WSILInfo(const HostScriptBaseURL: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean);
```

C++

```
__fastcall WSILInfo(const UnicodeString HostScriptBaseURL, TWebRequest Request, TWebResponse
Response, Boolean Handled);
```

Description

The Web Module (or Web Dispatcher) calls WSILInfo when a request is made for the WS inspection document for the Web Service (when the path portion of the request ends in the string 'inspection.wisl'.. WSILInfo responds by generating the WSIL description.

HostScriptBaseURL is the base path for accessing the WSDL documents of the Web Service.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that WSILInfo successfully generated a response.

See Also

DispatchRequest (see page 7236)

ServiceInfo (see page 7238)

80.1.1.5 TWSDLHTMLPublish Properties

The properties of the TWSDLHTMLPublish class are listed here.

Published Properties

	Name	Description
	AdminEnabled (see page 7239)	Enables or disables the WSDL administrator.
	PublishOptions (see page 7240)	Controls the definitions TWSDLHTMLPublish (see page 7232) creates when it generates its WSDL documents.
	TargetNamespace (see page 7240)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 7240)	This is WebDispatch, a member of class TWSDLHTMLPublish.

80.1.1.5.1 TWSDLHTMLPublish.AdminEnabled Property

Enables or disables the WSDL administrator.

Pascal

```
property AdminEnabled: Boolean;
```

C++

```
__property Boolean AdminEnabled;
```

Description

Use AdminEnabled to indicate whether this application implements the WSDL administrator. When AdminEnabled is true, pointing a Web browser at the URL that addresses this application using the path info specified by WebDispatch (see page 7240) launches a Web page that includes a button for the WSDL administrator. The WSDL administrator can then adjust the published WSDL documents to indicate an address other than the one where this application is installed as the location or locations where the Web Services are deployed.

When AdminEnabled is false, the URL that includes WebDispatch (see page 7240) publishes a list of WSDL documents but

does not provide access to the WSDL administrator.

Tip: You can set AdminEnabled to true, update the URLs in the resulting WSDL documents, and then rebuild your application, setting AdminEnabled to false. The WSDL administrator stores your changes in a .ini file so that they appear in the list of WSDL documents even after you have changed AdminEnabled.

See Also

webdispatch (see page 7240)

TargetNamespace (see page 7240)

Generating WSDL Documents for a Web Service Application

80.1.1.5.2 TWSDLHTMLPublish.PublishOptions Property

Controls the definitions TWSDLHTMLPublish (see page 7232) creates when it generates its WSDL documents.

Pascal

```
property PublishOptions: TSOAPPublishOptions;
```

C++

```
__property TSOAPPublishOptions PublishOptions;
```

Description

PublishOptions is a set of flags that customize the generated WSDL document.

See Also

TargetNamespace (see page 7240)

80.1.1.5.3 TWSDLHTMLPublish.TargetNamespace Property

Specifies the target namespace URL of the WSDL document.

Pascal

```
property TargetNamespace: InvString;
```

C++

```
__property InvString TargetNamespace;
```

Description

Use TargetNamespace to specify a namespace for the WSDL documents this component publishes. This is the namespace URL that the WSDL document uses for the definitions it publishes.

See Also

AdminEnabled (see page 7239)

80.1.1.5.4 TWSDLHTMLPublish.WebDispatch Property

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

This is WebDispatch, a member of class TWSDLHTMLPublish.

80.1.1.6 TWSDLHTMLPublish Events

The events of the TWSDLHTMLPublish class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 7241)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 7241)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 7242)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 7242)	Occurs before the publisher generates the schema for a given type.

80.1.1.6.1 TWSDLHTMLPublish.OnAfterPublishingWSDL Event

Occurs after the publisher generates a WSDL document for the Web Service application.

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

Use OnAfterPublishingWSDL to perform special actions after the publisher has generated a WSDL document. For example, you can use this event to cache the generated WSDL document and then re-use the cached document in an OnBeforePublishingWSDL (see page 7242) event rather than requiring the publisher to generate a new document for every WSDL request it receives.

See Also

OnBeforePublishingWSDL (see page 7242)

80.1.1.6.2 TWSDLHTMLPublish.OnBeforePublishingTypes Event

Occurs before the publisher starts adding type definitions to the WSDL document.

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

Use OnBeforePublishingTypes to add your own type definitions to the WSDL document before the publisher adds definitions for the types that the invokable interfaces use and types that are registered with the remotable type registry. If the OnBeforePublishingTypes event handler adds a schema definition for a type, the publisher uses that definition rather than generating its own version.

See Also

OnBeforePublishingWSDL (see page 7242)

OnPublishingType (see page 7242)

80.1.1.6.3 TWSDLHTMLPublish.OnBeforePublishingWSDL Event

Occurs before the publisher generates its WSDL document.

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

Use OnBeforePublishingWSDL to provide your own WSDL document rather than letting the publisher generate one dynamically. For example, the event handler can supply a cached version of the WSDL document that you saved previously from an OnAfterPublishingWSDL (see page 7241) event handler.

See Also

OnAfterPublishingWSDL (see page 7241)

OnBeforePublishingTypes (see page 7241)

80.1.1.6.4 TWSDLHTMLPublish.OnPublishingType Event

Occurs before the publisher generates the schema for a given type.

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

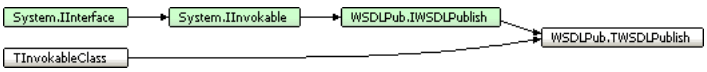
Use OnPublishingType to perform specific actions immediately before the publisher adds a type definition to the emerging WSDL document.

See Also

OnBeforePublishingTypes (see page 7241)

80.1.2 TWSDLPublish Class

Class Hierarchy



Pascal

```
TWSDLPublish = class(TInvokableClass, IWSDLPublish);
```

C++

```
class TWSDLPublish : public TInvokableClass, public IWSDLPublish;
```

File

WSDLPub

Description

This is class WSDLPub.TWSDLPublish.

80.1.2.1 TWSDLPublish Members

The following tables list the members exposed by TWSDLPublish.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 7245)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 7245)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 7246)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.
	OnPublishingType (see page 7246)	This is OnPublishingType, a member of class TWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	GetPortTypeEntries (see page 7243)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 7244)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 7244)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 7244)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 7244)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

Public Properties

	Name	Description
	TargetNamespace (see page 7245)	This is TargetNamespace, a member of class TWSDLPublish.

80.1.2.2 TWSDLPublish Methods

The methods of the TWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeEntries (see page 7243)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 7244)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 7244)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 7244)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 7244)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.

80.1.2.2.1 TWSDLPublish.GetPortTypeEntries Method

Pascal

```
procedure GetPortTypeEntries(var Entries: TInvRegIntfEntryArray);
```

C++

```
__fastcall GetPortTypeEntries(TInvRegIntfEntryArray Entries);
```

Description

This is GetPortTypeEntries, a member of class TWSDLPublish.

80.1.2.2.2 TWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class TWSDLPublish.

80.1.2.2.3 TWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class TWSDLPublish.

80.1.2.2.4 TWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetWSDLForPortType(const InvString PortType);
```

Description

This is GetWSDLForPortType, a member of class TWSDLPublish.

80.1.2.2.5 TWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetXSDForTypeSystem(const InvString TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class TWSDLPublish.

80.1.2.3 TWSDLPublish Properties

The properties of the TWSDLPublish class are listed here.

Public Properties

	Name	Description
	TargetNamespace (see page 7245)	This is TargetNamespace, a member of class TWSDLPublish.

80.1.2.3.1 TWSDLPublish.TargetNamespace Property

Pascal

```
property TargetNamespace: InvString;
```

C++

```
__property InvString TargetNamespace;
```

Description

This is TargetNamespace, a member of class TWSDLPublish.

80.1.2.4 TWSDLPublish Events

The events of the TWSDLPublish class are listed here.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 7245)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 7245)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 7246)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.
	OnPublishingType (see page 7246)	This is OnPublishingType, a member of class TWSDLPublish.

80.1.2.4.1 TWSDLPublish.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWSDLPublish.

80.1.2.4.2 TWSDLPublish.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWSDLPublish.

80.1.2.4.3 TWSDLPublish.OnBeforePublishingWSDL Event

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

This is OnBeforePublishingWSDL, a member of class TWSDLPublish.

80.1.2.4.4 TWSDLPublish.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWSDLPublish.

80.2 Interfaces

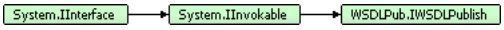
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWSDLPublish (see page 7247)	This is class WSDLPub.IWSDLPublish.

80.2.1 IWSDLPublish Interface

Class Hierarchy



Pascal

```
IWSDLPublish = interface(IInvokable);
```

C++

```
__interface IWSDLPublish : public IInvokable;
```

File

WSDLPub

Description

This is class WSDLPub.IWSDLPublish.

80.2.1.1 IWSDLPublish Members

The following tables list the members exposed by IWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	GetPortTypeList (see page 7248)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 7248)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 7248)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 7248)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.

80.2.1.2 IWSDLPublish Methods

The methods of the IWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeList (see page 7248)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 7248)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 7248)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 7248)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.

80.2.1.2.1 IWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class IWSDLPublish.

80.2.1.2.2 IWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class IWSDLPublish.

80.2.1.2.3 IWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetWSDLForPortType(const InvString PortType);
```

Description

This is GetWSDLForPortType, a member of class IWSDLPublish.

80.2.1.2.4 IWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetXSDForTypeSystem(const InvString TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class IWSDLPublish.

80.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WSDLPubFactory (see page 7249)	This is function WSDLPub.WSDLPubFactory.

80.3.1 WSDLPub.WSDLPubFactory Function

Pascal

```
procedure WSDLPubFactory(out obj: TObject);
```

C++

```
WSDLPubFactory(TObject * obj);
```

File

WSDLPub

Description

This is function WSDLPub.WSDLPubFactory.

80.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPPublishOptions (see page 7250)	This is record WSDLPub.SOAPPublishOptions.

80.4.1 WSDLPub.SOAPPublishOptions Enumeration

Pascal

```
SOAPPublishOptions = (
  poDefault,
  poPublishLocationAsSecure,
  poUTF8ContentType
);
```

C++

```
enum SOAPPublishOptions {
  poDefault,
  poPublishLocationAsSecure,
  poUTF8ContentType
};
```

File

WSDLPub

Description

This is record WSDLPub.SOAPPublishOptions.

80.5 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 7251)	This is type WSDLPub.TBeforePublishingWSDLEvent.
TSOAPPublishOptions (see page 7251)	This is type WSDLPub.TSOAPPublishOptions.

80.5.1 WSDLPub.TBeforePublishingWSDLEvent Type

Pascal

```
TBeforePublishingWSDLEvent = procedure (const IntfName: InvString; var WSDL: InvString; var
Handled: Boolean) of object;
```

C++

```
(const IntfName: InvString; var WSDL: InvString; var Handled: Boolean) (
TBeforePublishingWSDLEvent)();
```

File

WSDLPub

Description

This is type WSDLPub.TBeforePublishingWSDLEvent.

80.5.2 WSDLPub.TSOAPPublishOptions Type

Pascal

```
TSOAPPublishOptions = set of SOAPPublishOptions;
```

C++

```
SOAPPublishOptions TSOAPPublishOptions;
```

File

WSDLPub

Description

This is type WSDLPub.TSOAPPublishOptions.

81 XMLBrokr Namespace

This is namespace XMLBrokr.

Topics

Name	Description
Classes (see page 7255)	The following table lists classes in this documentation.
Interfaces (see page 7289)	The following table lists interfaces in this documentation.
Functions (see page 7291)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7292)	The following table lists structs, records, enums in this documentation.
Types (see page 7293)	The following table lists types in this documentation.
Constants (see page 7295)	The following table lists constants in this documentation.

Classes

	Name	Description
	TXMLBroker (see page 7255)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 7285)	TXMLParams manages a list of parameters for an XML broker.

Constants

Name	Description
sPostDelta (see page 7295)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 7295)	This is constant XMLBrokr.sProducer.
sRedirect (see page 7295)	This is constant XMLBrokr.sRedirect.

Functions

	Name	Description
	FormatXML (see page 7291)	This is function XMLBrokr.FormatXML.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 7289)	This is class XMLBrokr.INotifyConnectionChange.

Structs, Records, Enums

	Name	Description
	TXMLOption (see page 7292)	This is record XMLBrokr.TXMLOption.

Types

Name	Description
TGetErrorResponseEvent (see page 7293)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 7293)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 7294)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 7294)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 7294)	This is type XMLBrokr.TXMLOptions.

81.1 Classes

The following table lists classes in this documentation.

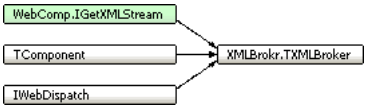
Classes

	Name	Description
	TXMLBroker (see page 7255)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 7285)	TXMLParams manages a list of parameters for an XML broker.

81.1.1 TXMLBroker Class

TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.

Class Hierarchy



Pascal

```
TXMLBroker = class(TComponent, IWebDispatch, IGetXMLStream);
```

C++

```
class TXMLBroker : public TComponent, public IWebDispatch, public IGetXMLStream;
```

File

XMLBrokr

Description

Use TXMLBroker in an application that is simultaneously the client of a multi-tiered database application and a Web server application. This class serves two major functions:

It fetches XML data packets from a provider using the IAppServer interface.

It receives HTTP messages from Web browsers that contain XML delta packets and applies them to the application server.

Add a TXMLBroker object to a Web module to make database information supplied by a provider on the application server available as XML. An InternetExpress page producer (TInetXPageProducer) in the same Web module uses this XML to build HTML pages that represent that database information.

The XML broker automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the XML broker without requiring the use of Web action items.

These incoming messages are assumed to be update messages created by the browser in response to HTML generated by a TApplyUpdatesButton component. The XML broker automatically applies the delta packet (the content of the HTTP request message) to the application server, and passes any update errors to a separate content producer, which can generate a response message.

See Also

TRemoteDataModule

IAppServer

TInetXPageProducer

TApplyUpdatesButton

Building an InternetExpress Application

81.1.1.1 TXMLBroker Members

The following tables list the members exposed by TXMLBroker.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 7260)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 7261)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 7261)	This is AS_GetRecords, a member of class TXMLBroker.
	CheckInactive (see page 7261)	This is CheckInactive, a member of class TXMLBroker.
	DispatchEnabled (see page 7261)	
	DispatchMask (see page 7261)	
	DispatchMethodType (see page 7262)	
	DispatchRequest (see page 7262)	
	GetAppServer (see page 7262)	This is GetAppServer, a member of class TXMLBroker.
	GetConnected (see page 7263)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 7263)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 7264)	This is GetHasAppServer, a member of class TXMLBroker.
	GetNotify (see page 7264)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 7265)	This is GetNotifyCount, a member of class TXMLBroker.
	GetProducerName (see page 7265)	This is GetProducerName, a member of class TXMLBroker.
	GetRedirect (see page 7265)	This is GetRedirect, a member of class TXMLBroker.
	GetResponse (see page 7265)	This is GetResponse, a member of class TXMLBroker.
	GetXMLStream (see page 7266)	This is GetXMLStream, a member of class TXMLBroker.
	Notification (see page 7267)	Forwards notification messages to all owned components.
	SendConnectEvent (see page 7271)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAppServer (see page 7271)	This is SetAppServer, a member of class TXMLBroker.
	SetConnected (see page 7271)	This is SetConnected, a member of class TXMLBroker.
	SetParams (see page 7271)	This is SetParams, a member of class TXMLBroker.
	SetProviderName (see page 7272)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 7272)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 7272)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 7273)	This is SetWebDispatch, a member of class TXMLBroker.

Public Methods

	Name	Description
	AddNotify (see page 7259)	Adds an object to the Notify (see page 7276) property list.
	ApplyXMLUpdates (see page 7260)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
	Create (see page 7257)	Creates a new TXMLBroker (see page 7255) object.
	Destroy (see page 7258)	Frees the memory associated with the TXMLBroker (see page 7255) object.
	FetchParams (see page 7262)	Retrieves parameter values from the application server.
	GetDelta (see page 7263)	Extracts an XML delta packet from an HTTP request message.

	GetErrorCount (see page 7263)	This is GetErrorCount, a member of class TXMLBroker.
	GetErrors (see page 7264)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetXMLRecords (see page 7265)	Retrieves an XML data packet from the application server.
	HTMLSubmitFormName (see page 7266)	Returns the name of the HTML form that submits updates to the XML broker.
	MasterRowSetVarName (see page 7267)	Returns the name of the javascript variable for the XML broker's dataset.
	RemoveNotify (see page 7268)	Removes an object to the Notify (see page 7276) property list.
V	RequestRecords (see page 7268)	Returns an XML data packet from the application server.
V	RequestUpdate (see page 7269)	Responds to an HTTP update request.
	RowSetVarName (see page 7270)	Returns the name of the javascript variable for the XML broker's dataset.
	SetProvider (see page 7272)	Associates the XML broker with a provider in the same process space.
	SubmitFormVarName (see page 7273)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7266).

Public Properties

	Name	Description
	AppServer (see page 7274)	Provides access to the interface the XML broker uses to communicate with the application server.
R	HasAppServer (see page 7275)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.
R	Notify (see page 7276)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
R	NotifyCount (see page 7277)	Indicates the number of objects in the Notify (see page 7276) property list.

Published Events

	Name	Description
	AfterDispatch (see page 7281)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 7282)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 7282)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 7283)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 7284)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 7284)	Occurs immediately before the XML broker applies updates to the application server.

Published Properties

	Name	Description
	Connected (see page 7274)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 7275)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 7276)	Specifies the maximum number of records that can appear in XML data packets.
	Params (see page 7277)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 7278)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 7279)	Specifies the producer that generates response message content when an update generates errors.
	RemoteServer (see page 7280)	Specifies the connection component that this XML broker uses to connect to an application server.
	WebDispatch (see page 7280)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

81.1.1.2 TXMLBroker.Create Constructor

Creates a new TXMLBroker (see page 7255) object.

Pascal

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

C++

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

Description

Most applications do not need to call the XML broker constructor. Typically the XML broker is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the XML broker.

See Also

Owner

81.1.1.3 TXMLBroker.Destroy Destructor

Frees the memory associated with the TXMLBroker (see page 7255) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLBroker();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the XML broker is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3382)

81.1.1.4 TXMLBroker Methods

The methods of the TXMLBroker class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 7260)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 7261)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 7261)	This is AS_GetRecords, a member of class TXMLBroker.
	CheckInactive (see page 7261)	This is CheckInactive, a member of class TXMLBroker.
	DispatchEnabled (see page 7261)	
	DispatchMask (see page 7261)	
	DispatchMethodType (see page 7262)	
	DispatchRequest (see page 7262)	
	GetAppServer (see page 7262)	This is GetAppServer, a member of class TXMLBroker.
	GetConnected (see page 7263)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 7263)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 7264)	This is GetHasAppServer, a member of class TXMLBroker.
	GetNotify (see page 7264)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 7265)	This is GetNotifyCount, a member of class TXMLBroker.
	GetProducerName (see page 7265)	This is GetProducerName, a member of class TXMLBroker.

	GetRedirect (see page 7265)	This is GetRedirect, a member of class TXMLBroker.
	GetResponse (see page 7265)	This is GetResponse, a member of class TXMLBroker.
	GetXMLStream (see page 7266)	This is GetXMLStream, a member of class TXMLBroker.
V	Notification (see page 7267)	Forwards notification messages to all owned components.
	SendConnectEvent (see page 7271)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAppServer (see page 7271)	This is SetAppServer, a member of class TXMLBroker.
	SetConnected (see page 7271)	This is SetConnected, a member of class TXMLBroker.
	SetParams (see page 7271)	This is SetParams, a member of class TXMLBroker.
	SetProviderName (see page 7272)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 7272)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 7272)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 7273)	This is SetWebDispatch, a member of class TXMLBroker.

Public Methods

	Name	Description
	AddNotify (see page 7259)	Adds an object to the Notify (see page 7276) property list.
	ApplyXMLUpdates (see page 7260)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
V	FetchParams (see page 7262)	Retrieves parameter values from the application server.
	GetDelta (see page 7263)	Extracts an XML delta packet from an HTTP request message.
	GetErrorCount (see page 7263)	This is GetErrorCount, a member of class TXMLBroker.
	GetErrors (see page 7264)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetXMLRecords (see page 7265)	Retrieves an XML data packet from the application server.
	HTMLSubmitFormName (see page 7266)	Returns the name of the HTML form that submits updates to the XML broker.
	MasterRowSetVarName (see page 7267)	Returns the name of the javascript variable for the XML broker's dataset.
	RemoveNotify (see page 7268)	Removes an object to the Notify (see page 7276) property list.
V	RequestRecords (see page 7268)	Returns an XML data packet from the application server.
V	RequestUpdate (see page 7269)	Responds to an HTTP update request.
	RowSetVarName (see page 7270)	Returns the name of the javascript variable for the XML broker's dataset.
	SetProvider (see page 7272)	Associates the XML broker with a provider in the same process space.
	SubmitFormVarName (see page 7273)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7266).

81.1.1.4.1 TXMLBroker.AddNotify Method

Adds an object to the Notify (see page 7276) property list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the list maintained by the Notify (see page 7276) property. Each object in this list is notified every time a property that affects the metadata of the XML broker's data packets changes. These include changes to the RemoteServer (see page 7280), AppServer (see page 7274), and ProviderName (see page 7278) properties.

ANotify specifies the object that wants to be notified of property changes. This object must support the INotifyConnectionChange (see page 7289) interface, which the XML broker uses to notify it of changes.

See Also

NotifyCount (see page 7277)

Notify (see page 7276)

RemoveNotify (see page 7268)

RemoteServer (see page 7280)

AppServer (see page 7274)

ProviderName (see page 7278)

81.1.1.4.2 TXMLBroker.ApplyXMLUpdates Method

Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.

Pascal

```
function ApplyXMLUpdates(const Delta: string; out ErrorCount: Integer): string;
```

C++

```
__fastcall UnicodeString ApplyXMLUpdates(const UnicodeString Delta, int ErrorCount);
```

Description

Call ApplyXMLUpdates to send the updates in the XML delta packet specified by Delta to the application server. ApplyXMLUpdates is used internally (by the RequestUpdate (see page 7269) method) to apply updates in an XML delta packet and retrieve a delta packet containing any failed updates.

Delta is the XML delta packet that contains the insertions, deletions and modifications from a Web browser. Typically, it is retrieved from an HTTP request message using the GetDelta (see page 7263) method.

ErrorCount returns the number of errors encountered on the application server before it terminated the update operation. If ErrorCount is 0, all updates were applied successfully. ErrorCount can not exceed the value of MaxErrors (see page 7275).

ApplyXMLUpdates returns an XML delta packet containing all updates that were not successfully applied. If ErrorCount is 0, this is an empty string.

See Also

RequestUpdate (see page 7269)

AS_ApplyUpdates

MaxErrors (see page 7275)

GetDelta (see page 7263)

81.1.1.4.3 TXMLBroker.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is AS_ApplyUpdates, a member of class TXMLBroker.

81.1.1.4.4 TXMLBroker.AS_FetchParams Method

Pascal

```
procedure AS_FetchParams;
```

C++

```
__fastcall AS_FetchParams();
```

Description

This is AS_FetchParams, a member of class TXMLBroker.

81.1.1.4.5 TXMLBroker.AS_GetRecords Method

Pascal

```
function AS_GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; Params: OleVariant; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class TXMLBroker.

81.1.1.4.6 TXMLBroker.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TXMLBroker.

81.1.1.4.7 TXMLBroker.DispatchEnabled Method

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

See Also

Enabled

81.1.1.4.8 TXMLBroker.DispatchMask Method

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

See Also

Mask

81.1.1.4.9 TXMLBroker.DispatchMethodType Method

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

See Also

MethodType

81.1.1.4.10 TXMLBroker.DispatchRequest Method

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

81.1.1.4.11 TXMLBroker.FetchParams Method

Retrieves parameter values from the application server.

Pascal

```
procedure FetchParams; virtual;
```

C++

```
virtual __fastcall FetchParams();
```

Description

Call FetchParams to initialize the Params (see page 7277) property with the current parameter values on the application server. FetchParams does nothing unless the provider component on the application server can supply parameter values.

Warning: if the XML broker uses a stateless application server, other clients can change the values returned by FetchParams at any time.

See Also

RowRequest

Params (see page 7277)

81.1.1.4.12 TXMLBroker.GetAppServer Method

Pascal

```
function GetAppServer: IAppServer;
```

C++

```
__fastcall IAppServer GetAppServer();
```

Description

This is GetAppServer, a member of class TXMLBroker.

81.1.1.4.13 TXMLBroker.GetConnected Method

Pascal

```
function GetConnected: Boolean;
```

C++

```
__fastcall Boolean GetConnected();
```

Description

This is GetConnected, a member of class TXMLBroker.

81.1.1.4.14 TXMLBroker.GetDelta Method

Extracts an XML delta packet from an HTTP request message.

Pascal

```
function GetDelta(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetDelta(TWebRequest Request);
```

Description

Call GetDelta to obtain the XML delta packet contained in the content of the request message specified by Request. This value can be used to apply the updates from the request message by calling ApplyXMLUpdates (see page 7260).

See Also

ApplyXMLUpdates (see page 7260)

81.1.1.4.15 TXMLBroker.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

This is GetErrorCount, a member of class TXMLBroker.

81.1.1.4.16 TXMLBroker.GetErrorResponse Method

Pascal

```
function GetErrorResponse(ErrorCount: Integer; XMLErrors: string; Request: TWebRequest;
Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetErrorResponse(int ErrorCount, UnicodeString XMLErrors, TWebRequest
Request, TWebResponse Response);
```

Description

This is GetErrorResponse, a member of class TXMLBroker.

81.1.1.4.17 TXMLBroker.GetErrors Method

Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.

Pascal

```
function GetErrors: string;
```

C++

```
__fastcall UnicodeString GetErrors();
```

Description

Call GetErrors to obtain the XML delta packet returned by the application server when an update operation included some errors.

Note: See the template err.html (included in the WebMidas directory) for one possible use for the XML delta packet of errors. In err.html, the string returned by GetErrors should replace the <#XMLEMBED_VAR_ERR> tag.

See Also

OnGetErrorResponse (see page 7282)

ApplyXMLUpdates (see page 7260)

GetErrorcount (see page 7263)

81.1.1.4.18 TXMLBroker.GetHasAppServer Method

Pascal

```
function GetHasAppServer: Boolean;
```

C++

```
__fastcall Boolean GetHasAppServer();
```

Description

This is GetHasAppServer, a member of class TXMLBroker.

81.1.1.4.19 TXMLBroker.GetNotify Method

Pascal

```
function GetNotify(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetNotify(int Index);
```

Description

This is GetNotify, a member of class TXMLBroker.

81.1.1.4.20 TXMLBroker.GetNotifyCount Method

Pascal

```
function GetNotifyCount: Integer;
```

C++

```
__fastcall int GetNotifyCount();
```

Description

This is GetNotifyCount, a member of class TXMLBroker.

81.1.1.4.21 TXMLBroker.GetProducerName Method

Pascal

```
function GetProducerName(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetProducerName(TWebRequest Request);
```

Description

This is GetProducerName, a member of class TXMLBroker.

81.1.1.4.22 TXMLBroker.GetRedirect Method

Pascal

```
function GetRedirect(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetRedirect(TWebRequest Request);
```

Description

This is GetRedirect, a member of class TXMLBroker.

81.1.1.4.23 TXMLBroker.GetResponse Method

Pascal

```
function GetResponse(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetResponse(TWebRequest Request, TWebResponse Response);
```

Description

This is GetResponse, a member of class TXMLBroker.

81.1.1.4.24 TXMLBroker.GetXMLRecords Method

Retrieves an XML data packet from the application server.

Pascal

```
function GetXMLRecords(var RecsOut: Integer; var OwnerData: OleVariant; XMLOptions:
TXMLOptions): string;
```

C++

```
__fastcall UnicodeString GetXMLRecords(int RecsOut, OleVariant OwnerData, TXMLOptions
XMLOptions);
```

Description

Call GetXMLRecords to obtain the XML data packet from the application server. GetXMLRecords calls the interface specified by AppServer (see page 7274), requesting an XML data packet with at most MaxRecords (see page 7276) records. It returns the resulting XML data packet in a form that is ready for insertion into an HTML document.

RecsOut returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags that indicate how the broker should format the XML data packet after it is retrieved from the application server.

GetXMLRecords returns the formatted XML data packet.

Note: Unlike the RequestRecords (see page 7268) method, GetXMLRecords bypasses the OnRequestRecords (see page 7284) event.

See Also

RequestRecords (see page 7268)

AS_GetRecords

MaxRecords (see page 7276)

AppServer (see page 7274)

BeforeGetRecords

AfterGetRecords

Using an XML Broker

81.1.1.4.25 TXMLBroker.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

Description

This is GetXMLStream, a member of class TXMLBroker.

81.1.1.4.26 TXMLBroker.HTMLSubmitFormName Method

Returns the name of the HTML form that submits updates to the XML broker.

Pascal

```
function HTMLSubmitFormName: string;
```

C++

```
__fastcall UnicodeString HTMLSubmitFormName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. HTMLSubmitFormName is the name of that HTML form.

The HTML form named by this return value sends the HTTP request messages containing an XML delta packet of updates to be applied.

See Also

SubmitFormVarName (see page 7273)

Content

81.1.1.4.27 TXMLBroker.MasterRowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function MasterRowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall UnicodeString MasterRowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the variable is first declared, the InternetExpress page producer calls MasterRowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set. These names come from previous calls to the MasterRowSetVarName method of their XML brokers. The last string in Path refers to the XML broker's dataset itself.

MasterRowSetVarName returns the name of the XML broker, qualified by all strings in Path except for the last one, which is assumed to refer to the XML broker's dataset. Thus, to change the name of the generated variable for an XML broker's dataset, change the Name property of the XML broker.

Note: To retrieve the name of the XML broker preceded by all the datasets in Path, use the RowSetVarName (see page 7270) method instead.

See Also

Name

Content

RowSetVarName (see page 7270)

81.1.1.4.28 TXMLBroker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

81.1.1.4.29 TXMLBroker.RemoveNotify Method

Removes an object to the Notify (see page 7276) property list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list maintained by the Notify (see page 7276) property. Each object in this list is notified every time a property that affects the metadata of the XML broker's data packets changes.

See Also

NotifyCount (see page 7277)

Notify (see page 7276)

AddNotify (see page 7259)

81.1.1.4.30 TXMLBroker.RequestRecords Method

Returns an XML data packet from the application server.

Pascal

```
function RequestRecords(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var
OwnerData: OleVariant; XMLOptions: TXMLOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString RequestRecords(TObject * Sender, TWebRequest Request, int
RecCount, OleVariant OwnerData, TXMLOptions XMLOptions);
```

Description

Call RequestRecords to obtain a current XML data packet, while first allowing the OnRequestRecords (see page 7284) event handler to customize it. After generating the OnRequestRecords (see page 7284) event, RequestRecords fetches the XML data packet by calling GetXMLRecords (see page 7265), unless the event handler supplies the XML data packet first.

Sender is the object that is requesting the XML data packet (usually an InternetExpress page producer).

Request describes an HTTP request message that originated the need for an XML data packet. Typically, this is a request message for which the InternetExpress page producer is creating a response.

RecCount returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the OnRequestRecords (see page 7284) event handler, and then forwarded (with any changes provided by the event handler) to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags indicating how the XML broker should format the XML data packet that the application server or OnRequestRecords (see page 7284) event handler supplies.

RequestRecords returns the XML data packet.

See Also

GetXMLRecords (see page 7265)

AS_GetRecords

OnRequestRecords (see page 7284)

AppServer (see page 7274)

BeforeGetRecords

AfterGetRecords

Using an XML Broker

81.1.1.4.31 TXMLBroker.RequestUpdate Method

Responds to an HTTP update request.

Pascal

```
function RequestUpdate(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RequestUpdate(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

Call RequestUpdate when using a Web action item to respond to update requests from a Web browser. When using a Web action item to respond to update requests, instead of letting the XML broker receive the update requests directly, you should set the Enabled property of the WebDispatch (see page 7280) to false.

RequestUpdate generates an OnRequestUpdate (see page 7284) event, to allow the event handler to apply the updates in the HTTP request message and generate a response. If the OnRequestUpdate (see page 7284) event handler does not handle the message, RequestUpdate applies the updates using the ApplyXMLUpdates (see page 7260) method. Finally, if there are no update errors, RequestUpdate generates an OnGetResponse (see page 7283) event. If there were update errors,

RequestResponse generates an OnGetErrorResponse (see page 7282) event and (if the event handler does not generate a response) assigns the content of ReconcileProducer (see page 7279) to the response message. If there are update errors and the response is not completed by an event handler or the ReconcileProducer (see page 7279), the XML broker raises an exception.

Sender is the object that received the request message.

Request describes an HTTP request message that contains the updates that need to be applied.

Response is the emerging response object that should be completed by RequestUpdate so that the XML broker can send it back to the Web browser.

RequestUpdate returns true if the response message is ready to send back to the Web browser, false if it still needs more processing.

See Also

ApplyXMLUpdates (see page 7260)

AS_ApplyUpdates

OnRequestUpdate (see page 7284)

AppServer (see page 7274)

BeforeApplyUpdates

AfterApplyUpdates

OnGetResponse (see page 7283)

OnGetErrorResponse (see page 7282)

ReconcileProducer (see page 7279)

WebDispatch (see page 7280)

Enabled

81.1.1.4.32 TXMLBroker.RowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function RowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall UnicodeString RowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the InternetExpress page producer needs to determine whether a RowSet variable was already declared, it looks up the variable name by calling RowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set.

RowSetVarName returns the name of the XML broker, qualified by all strings in Path.

Note: Variable names are assigned for an XML broker's dataset using the MasterRowSetVarName (see page 7267) method.

See Also

Name

Content

MasterRowSetVarName (see page 7267)

81.1.1.4.33 TXMLBroker.SendConnectEvent Method

Pascal

```
procedure SendConnectEvent(Connecting: Boolean);
```

C++

```
__fastcall SendConnectEvent(Boolean Connecting);
```

Description

This is SendConnectEvent, a member of class TXMLBroker.

81.1.1.4.34 TXMLBroker.SetAppServer Method

Pascal

```
procedure SetAppServer(Value: IAppServer);
```

C++

```
__fastcall SetAppServer(IAppServer Value);
```

Description

This is SetAppServer, a member of class TXMLBroker.

81.1.1.4.35 TXMLBroker.SetConnected Method

Pascal

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

C++

```
__fastcall SetConnected(Boolean Value);
```

Description

This is SetConnected, a member of class TXMLBroker.

81.1.1.4.36 TXMLBroker.SetParams Method

Pascal

```
procedure SetParams(const Value: TXMLParams);
```

C++

```
__fastcall SetParams(const TXMLParams Value);
```

Description

This is SetParams, a member of class TXMLBroker.

81.1.1.4.37 TXMLBroker.SetProvider Method

Associates the XML broker with a provider in the same process space.

Pascal

```
procedure SetProvider(Provider: TComponent);
```

C++

```
__fastcall SetProvider(TComponent * Provider);
```

Description

Use SetProvider to specify a local provider component that provides data to and applies updates from the XML broker. The XML broker communicates with this provider using the IAppServer interface specified by the AppServer (see page 7274) property.

Do not use SetProvider for remote providers on an application server. If the provider is not in the same process space as the XML broker, use the ProviderName (see page 7278) property.

The Provider parameter must be a descendant of TCustomProvider. Otherwise, SetProvider raises an exception.

See Also

AppServer (see page 7274)

ProviderName (see page 7278)

TDataSetProvider

TCustomProvider

81.1.1.4.38 TXMLBroker.SetProviderName Method

Pascal

```
procedure SetProviderName(const Value: string);
```

C++

```
__fastcall SetProviderName(const UnicodeString Value);
```

Description

This is SetProviderName, a member of class TXMLBroker.

81.1.1.4.39 TXMLBroker.SetReconcileProducer Method

Pascal

```
procedure SetReconcileProducer(const Value: TCustomContentProducer);
```

C++

```
__fastcall SetReconcileProducer(const TCustomContentProducer Value);
```

Description

This is SetReconcileProducer, a member of class TXMLBroker.

81.1.1.4.40 TXMLBroker.SetRemoteServer Method

Pascal

```
procedure SetRemoteServer(Value: TCustomRemoteServer);
```

C++

```
__fastcall SetRemoteServer(TCustomRemoteServer Value);
```

Description

This is SetRemoteServer, a member of class TXMLBroker.

81.1.1.4.41 TXMLBroker.SetWebDispatch Method

Pascal

```
procedure SetWebDispatch(const Value: TWebDispatch);
```

C++

```
__fastcall SetWebDispatch(const TWebDispatch Value);
```

Description

This is SetWebDispatch, a member of class TXMLBroker.

81.1.1.4.42 TXMLBroker.SubmitFormVarName Method

Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7266).

Pascal

```
function SubmitFormVarName: string;
```

C++

```
__fastcall UnicodeString SubmitFormVarName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. In addition, it declares a variable that refers to that form. SubmitFormVarName is the name of that variable.

See Also

HTMLSubmitFormName (see page 7266)

Content

81.1.1.5 TXMLBroker Properties

The properties of the TXMLBroker class are listed here.

Public Properties

	Name	Description
	AppServer (see page 7274)	Provides access to the interface the XML broker uses to communicate with the application server.
R	HasAppServer (see page 7275)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.
R	Notify (see page 7276)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
R	NotifyCount (see page 7277)	Indicates the number of objects in the Notify (see page 7276) property list.

Published Properties

	Name	Description
	Connected (see page 7274)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 7275)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 7276)	Specifies the maximum number of records that can appear in XML data packets.
	Params (see page 7277)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 7278)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 7279)	Specifies the producer that generates response message content when an update generates errors.
	RemoteServer (see page 7280)	Specifies the connection component that this XML broker uses to connect to an application server.
	WebDispatch (see page 7280)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

81.1.1.5.1 TXMLBroker.AppServer Property

Provides access to the interface the XML broker uses to communicate with the application server.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to communicate with the providers that supply data to and resolve updates for the XML broker. If the providers are on a remote application server, AppServer has the same value as the AppServer property of the connection component. If the XML broker uses a provider in the same process space, AppServer is the interface to an internally created object that dispatches IAppServer calls to the local provider.

Applications don't need to work directly with the IAppServer interface. The properties and methods of the XML broker are automatically translated into IAppServer calls as needed.

See Also

HasAppServer (see page 7275)

AppServer

SetProvider (see page 7272)

81.1.1.5.2 TXMLBroker.Connected Property

Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Use Connected to determine whether the XML broker can supply XML data packets and apply updates. If the XML broker uses a provider on an application server, setting Connected to true sets the Connected property of the connection component to true. If

the XML broker uses a local provider component, setting Connected to true causes the XML broker to generate an internal object that communicates with the provider using the IAppServer interface. Thus, the value of Connected indicates whether an IAppServer interface is available for communicating with a provider, and so the value of Connected is the same as the value of HasAppServer (see page 7275).

You can't set Connected to true unless the XML broker has a provider to which it can connect. Trying to set Connected to true before the ProviderName (see page 7278) property is set or a local provider is assigned using the SetProvider (see page 7272) method causes the XML broker to raise an exception.

Note: In the Web Page editor, default fields or parameter values may not appear if the associated XML broker is not connected.

See Also

HasAppServer (see page 7275)

AppServer (see page 7274)

Connected

TDataSetProvider

ProviderName (see page 7278)

SetProvider (see page 7272)

81.1.1.5.3 TXMLBroker.HasAppServer Property

Indicates whether the XML broker has an IAppServer interface for communicating with its provider.

Pascal

```
property HasAppServer: Boolean;
```

C++

```
__property Boolean HasAppServer;
```

Description

Read (see page 3727) HasAppServer to determine whether the XML broker can get its data from an IAppServer interface. When HasAppServer is true, the XML broker can interact with its provider through the IAppServer interface.

Use HasAppServer to check whether the XML broker is connected rather than reading the AppServer (see page 7274) property. Reading AppServer (see page 7274) causes the XML broker to obtain an IAppServer interface if it does not already have one.

See Also

IAppServer

AppServer (see page 7274)

Connected (see page 7274)

81.1.1.5.4 TXMLBroker.MaxErrors Property

Specifies the maximum number of update errors allowed before aborting an update operations.

Pascal

```
property MaxErrors: Integer;
```

C++

```
__property int MaxErrors;
```

Description

Set MaxErrors to indicate the maximum number of errors that the associated provider should allow before prematurely stopping an update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

See Also

AS_ApplyUpdates

ReconcileProducer (see page 7279)

ApplyXMLUpdates (see page 7260)

81.1.1.5.5 TXMLBroker.MaxRecords Property

Specifies the maximum number of records that can appear in XML data packets.

Pascal

```
property MaxRecords: Integer;
```

C++

```
__property int MaxRecords;
```

Description

Set MaxRecords to limit the size of XML data packets retrieved from the associated provider.

When MaxRecords is –1, XML data packets contain all the records in the provider's dataset.

When MaxRecords is 0, XML data packets contain only metadata.

When MaxRecords is greater than 0, XML data packets contain up to MaxRecords records, depending on the number of records in the provider's dataset.

Tip: Because an InternetExpress application sends the entire data packet to client applications, it is important to limit the size of data packets. Otherwise, the time to download web pages containing data packets can be prohibitively long.

Note: When limiting the number of records that appear in data packets, you may need to fetch several data packets before all data is returned. Because InternetExpress applications are always stateless, you will need to supply information about what record should start the next data packet as the OwnerData parameter of GetXMLRecords (see page 7265) or RequestRecords (see page 7268). You can use a query form to obtain information about the last record seen by a particular Web client.

See Also

AS_GetRecords

GetXMLRecords (see page 7265)

RequestRecords (see page 7268)

TQueryForm

Using an XML Broker

81.1.1.5.6 TXMLBroker.Notify Property

Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.

Pascal

```
property Notify [Index: Integer]: TObject;
```

C++

```
__property TObject * Notify[int Index];
```

Description

Read (see page 3727) Notify to access the objects that respond when the XML broker's connection to its provider changes. These objects are sent a notification when the ProviderName (see page 7278), AppServer (see page 7274), or RemoteServer (see page 7280) property changes.

Most applications do not need to use the Notify property. It allows the Web page editor to remain synchronized with property changes made at design time.

See Also

NotifyCount (see page 7277)

AddNotify (see page 7259)

RemoveNotify (see page 7268)

RemoteServer (see page 7280)

AppServer (see page 7274)

ProviderName (see page 7278)

81.1.1.5.7 TXMLBroker.NotifyCount Property

Indicates the number of objects in the Notify (see page 7276) property list.

Pascal

```
property NotifyCount: Integer;
```

C++

```
__property int NotifyCount;
```

Description

NotifyCount specifies the number of objects listed in the Notify (see page 7276) property.

See Also

Notify (see page 7276)

AddNotify (see page 7259)

RemoveNotify (see page 7268)

81.1.1.5.8 TXMLBroker.Params Property

Contains parameter values that are sent to or retrieved from the application server.

Pascal

```
property Params: TXMLParams;
```

C++

```
__property TXMLParams Params;
```

Description

Use Params to specify parameter values that should be passed to a source dataset on the application server. These parameter values are used in one of two ways:

If the associated provider represents a query or stored procedure, they supply parameter values that the application server uses when executing the query or stored procedure.

If the associated provider represents a table, they limit the records that are sent in data packets so that they only include records where each field with the same name as a parameter has the same value as the parameter.

After the application specifies parameters (at design time or runtime), they are sent to the application server automatically when the XML broker fetches data.

To send parameter values to a query or stored procedure, add a parameter object (TParam) to Params for every parameter on the query or stored procedure for which you want to supply a value. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the application server when the XML broker fetches data.

To send parameter values to a table, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the application server only contain records that match the assigned value on the corresponding fields.

Tip: You can assign parameter values from the content fields supplied to a request message by TSubmitQueryButton using the AssignStrings method of Params.

Note: To initialize Params so that it reflects the parameters of a query or stored procedure on an application server, call the FetchParams (see page 7262) method.

See Also

Params

Params

FetchParams (see page 7262)

TSubmitQueryButton

AssignStrings (see page 7287)

Using an XML Broker

81.1.1.5.9 TXMLBroker.ProviderName Property

Specifies the name of a registered provider from the application server.

Pascal

```
property ProviderName: string;
```

C++

```
__property UnicodeString ProviderName;
```

Description

Use ProviderName to specify the provider with which to communicate on the server (or in the same Web module). This property is passed as a parameter to IAppServer interface calls when the XML broker fetches data or applies updates. If an application server contains multiple provider objects, an XML broker can set ProviderName to any of them once the connection component

establishes a connection to the application server.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 7280) property is set.

Note: ProviderName can also be used to link the XML broker to a provider in the same process space, as long as the provider component has the same Owner as the XML broker. If the local provider component has a different Owner, use the SetProvider (see page 7272) method instead.

See Also

RemoteServer (see page 7280)

TDataSetProvider

IAppServer

SetProvider (see page 7272)

Using an XML Broker

81.1.1.5.10 TXMLBroker.ReconcileProducer Property

Specifies the producer that generates response message content when an update generates errors.

Pascal

```
property ReconcileProducer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer ReconcileProducer;
```

Description

Use ReconcileProducer to specify the component that generates the content of response messages to update requests when the update operation generated errors.

When the XML broker applies updates, the application server returns a delta packet containing all records that could not be updated because of errors. The XML broker can attempt to respond to those errors or generate its own response message using the OnGetErrorResponse (see page 7282) event. However, unless the OnGetErrorResponse (see page 7282) event handler generates an HTML document to be sent in the response message and indicates that the update is completely handled, the XML broker must still generate a response to the update operation. To do so, it calls the Content method of the ReconcileProducer to generate the content of the response message.

Tip: The Reconcile producer will probably need information about the update errors that the application server returned. You can provide it with this information either by initializing the Reconcile producer in an OnGetErrorResponse (see page 7282) event handler or by having the Reconcile producer call the GetErrors (see page 7264) method.

Note: If there is no ReconcileProducer assigned and an update generates errors, the XML broker raises an exception unless the OnGetErrorResponse (see page 7282) event handler generates a response message.

Tip: The InetXCustom demo contains a TReconcilePageProducer component that you can use as the value of the ReconcileProducer property or as the basis of your own reconcile producer.

See Also

MaxErrors (see page 7275)

OnGetErrorResponse (see page 7282)

ApplyXMLUpdates (see page 7260)

GetErrors (see page 7264)

Using an XML Broker

81.1.1.5.11 TXMLBroker.RemoteServer Property

Specifies the connection component that this XML broker uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

Use RemoteServer to specify the connection component that establishes a connection to the remote application server. The connection component obtains an IAppServer interface for data exchange and maintains the interface of the remote data module on the application server.

If a Web application uses more than connection component, the application can change which connection an XML broker uses by changing the value for this property.

See Also

ProviderName (see page 7278)

TSocketConnection (see page 176)

TDCOMConnection

TWebConnection (see page 209)

AppServer

IAppServer

Using an XML Broker

81.1.1.5.12 TXMLBroker.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the HTTP messages to which the XML broker responds. Typically, these originate from a button added to HTML forms by TApplyUpdatesButton.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which the XML broker responds, as well as the types of messages it accepts (typically post messages). You can also use this property to disable the XML broker so that the Web dispatcher does not send it any HTTP messages. When the XML broker is disabled in this way, it can still provide XML data packets to an InternetExpress page producer, but any update messages must be handled by a Web action item.

See Also

TWebModule

TWebDispatcher

TApplyUpdatesButton

Using an XML Broker

81.1.1.6 TXMLBroker Events

The events of the TXMLBroker class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 7281)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 7282)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 7282)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 7283)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 7284)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 7284)	Occurs immediately before the XML broker applies updates to the application server.

81.1.1.6.1 TXMLBroker.AfterDispatch Event

Occurs after the XML broker generates the content of a response message.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write (see page 3789) an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the XML broker an update message. Usually by the time AfterDispatch occurs, the XML broker has already applied the updates and generated the response message (using the ReconcileProducer (see page 7279) or one of the events that occurs when the XML broker receives an update request).

See Also

WebDispatch (see page 7280)

BeforeDispatch (see page 7282)

OnRequestUpdate (see page 7284)

OnGetResponse (see page 7283)

OnGetErrorResponse (see page 7282)

ReconcileProducer (see page 7279)

Using an XML Broker

81.1.1.6.2 TXMLBroker.BeforeDispatch Event

Occurs when the XML broker receives an HTTP request message.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write (see page 3789) a BeforeDispatch event handler to initialize the response to an update request or to handle any messages sent to the XML broker that are not update requests. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing the XML broker from applying the updates.

Unless the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an OnRequestUpdate (see page 7284) event after the BeforeDispatch event handler exits. After the OnRequestUpdate (see page 7284) event, the XML broker applies the updates to the application server and generates the contents of the response based on whether there were any update errors. Finally, the XML broker receives an AfterDispatch (see page 7281) event where it can perform any final changes to the response message before it is sent back to the Web browser.

If the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an AfterDispatch (see page 7281) event, but does not attempt to apply updates from the request message or receive any of the events that accompany that process.

See Also

WebDispatch (see page 7280)

AfterDispatch (see page 7281)

OnRequestUpdate (see page 7284)

Using an XML Broker

81.1.1.6.3 TXMLBroker.OnGetErrorResponse Event

Occurs when there are errors applying updates to the application server.

Pascal

```
property OnGetErrorResponse: TGetErrorResponseEvent;
```

C++

```
__property TGetErrorResponseEvent OnGetErrorResponse;
```

Description

Write (see page 3789) an OnGetErrorResponse event handler to generate the response message after an update request encountered errors or to initialize a ReconcileProducer (see page 7279) before it generates the response.

The Sender parameter is the XML broker that applied updates to the application server and received back a delta packet of update errors.

ErrorCount is the number of update errors found by the application server before it halted the update operation.

XMLErrors is a string of XML returned by the application server. It is a delta packet containing all the updates that could not be applied.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as an error message or a reconcile form) that indicates the errors in XMLErrors.

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if ReconcileProducer (see page 7279) should generate the content of Response.

Note: If the OnGetErrorResponse event handler does not set Handled to true and there is no ReconcileProducer (see page 7279), the XML broker raises an exception.

See Also

MaxErrors (see page 7275)

ReconcileProducer (see page 7279)

OnGetResponse (see page 7283)

ApplyXMLUpdates (see page 7260)

Using an XML Broker

81.1.1.6.4 TXMLBroker.OnGetResponse Event

Occurs when there are no errors applying updates to the application server.

Pascal

```
property OnGetResponse: TGetResponseEvent;
```

C++

```
__property TGetResponseEvent OnGetResponse;
```

Description

Write (see page 3789) an OnGetResponse event handler to edit the response message after an update request is successfully applied.

The Sender parameter is the XML broker that applied updates to the application server with no errors.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as refreshing the data).

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if the XML broker should send a redirect response that sends the browser back to the page that generated the update request.

See Also

OnGetErrorResponse (see page 7282)

ApplyXMLUpdates (see page 7260)

Using an XML Broker

81.1.1.6.5 TXMLBroker.OnRequestRecords Event

Occurs when the XML broker needs to fetch a data packet from its associated provider.

Pascal

```
property OnRequestRecords: TRequestRecordsEvent;
```

C++

```
__property TRequestRecordsEvent OnRequestRecords;
```

Description

Write (see page 3789) an OnRequestRecords event handler to provide an alternate mechanism for providing XML data packets or to perform other actions whenever the XML broker needs to fetch an XML data packet. For example, the OnRequestRecords event handler might call GetXMLRecords (see page 7265) to fetch an XML data packet from the application server and then edit its contents.

OnRequestRecords occurs when the InternetExpress page producer requests an XML data packet (usually in response to a request message) or when the application calls the XML broker's RequestRecords (see page 7268) method.

The Sender parameter is the object that originates the request (usually an InternetExpress page producer).

Request is the request message that prompted the request for an XML data packet. This value may be nil (Delphi) or NULL (C++) if the request did not arise from a page producer or if the page producer is not processing a request message.

OwnerData contains an application-defined variant that is sent to the provider's BeforeGetRecords event handler. The InternetExpress page producer that is requesting records or the caller of RequestRecords (see page 7268) may initialize this value. OwnerData can be used to pass persistent state information to a stateless application server (for example, the first record that should appear in the data packet when MaxRecords (see page 7276) is greater than 0).

Records optionally returns an XML data packet as a string. If Records is an empty string on exit from the event handler, the XML broker automatically fetches the XML data packet from its associated provider. Otherwise, the XML broker uses the value of Records as the new XML data packet.

See Also

RequestRecords (see page 7268)

GetXMLRecords (see page 7265)

GetXMLData

BeforeGetRecords

MaxRecords (see page 7276)

Using an XML Broker

81.1.1.6.6 TXMLBroker.OnRequestUpdate Event

Occurs immediately before the XML broker applies updates to the application server.

Pascal

```
property OnRequestUpdate: TRequestUpdateEvent;
```

C++

```
__property TRequestUpdateEvent OnRequestUpdate;
```

Description

Write (see page 3789) an OnRequestUpdate event handler to provide an alternate mechanism for applying updates to the application server. For example, the OnRequestUpdate event handler might call GetDelta (see page 7263) to extract the delta packet from the request message and then preprocess the XML string before applying the updates with a call to ApplyXMLUpdates (see page 7260).

OnRequestUpdate occurs immediately after the BeforeDispatch (see page 7282) event or when the application calls the XML broker's RequestUpdate (see page 7269) method.

The Sender parameter is the object that received the update request (the XML broker or a Web action item).

Request is the request message that contains the updates to be applied. The XML delta packet can be extracted from this request object using the GetDelta (see page 7263) method.

Response is the emerging response message that will be sent back to the Web browser that generated the update request. The event handler can set properties on this response object to partially or completely specify the message will be returned to the Web browser.

Handled indicates whether the event handler applied the updates. If Handled is true on exit from the event handler, the XML broker does not apply the updates, does not receive an OnGetResponse (see page 7283) or OnGetErrorResponse (see page 7282) event, and does not call on the ReconcileProducer (see page 7279) to generate a response message. (It does still receive an AfterDispatch (see page 7281) event). If Handled is false on exit from the event handler, the XML broker proceeds to apply the updates contained in the request message and receives all the associated events.

Note: If the event handler applies the updates, it must also generate the response message.

See Also

BeforeDispatch (see page 7282)

ApplyXMLUpdates (see page 7260)

RequestUpdate (see page 7269)

GetDelta (see page 7263)

OnGetResponse (see page 7283)

OnGetErrorResponse (see page 7282)

ReconcileProducer (see page 7279)

AfterDispatch (see page 7281)

Using an XML Broker

81.1.2 TXMLParams Class

TXMLParams manages a list of parameters for an XML broker.

Class Hierarchy

Pascal

```
TXMLParams = class(TParams);
```

C++

```
class TXMLParams : public TParams;
```

File

XMLBrokr

Description

XML brokers use TXMLParams to manage a list of parameters that it passes to a provider on the application server. These parameters supply values to a query or stored procedure on the application server, or limit the records provided from a table on the application server.

Use the properties and methods of TXMLParams to:

Access a specific parameter.

Add or delete parameters from the list.

Get or set the values of individual parameters.

Assign parameter values from a list such as the one supplied in a request message from a Web browser.

Compare parameters.

Iterate through all parameters.

Copy (see page 3648) a set of parameters to another parameter list.

See Also

TParam

TXMLBroker (see page 7255)

81.1.2.1 TXMLParams Members

The following tables list the members exposed by TXMLParams.

Protected Methods

	Name	Description
	AssignTo (see page 7287)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	AssignStrings (see page 7287)	Assigns values to the parameters from a list of name/value pairs.

81.1.2.2 TXMLParams Methods

The methods of the TXMLParams class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 7287)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	AssignStrings (see page 7287)	Assigns values to the parameters from a list of name/value pairs.

81.1.2.2.1 TXMLParams.AssignStrings Method

Assigns values to the parameters from a list of name/value pairs.

Pascal

```
procedure AssignStrings(Value: TStrings);
```

C++

```
__fastcall AssignStrings(TStrings Value);
```

Description

Use AssignStrings to update the values of the parameters in the Items list to match the values in the list specified by Value. For each string in Value of the form Name=Value, AssignStrings locates the parameter (if any) that has the same name as the name portion of the string. It then assigns the Value portion of the string as the parameter's value.

AssignStrings makes it very convenient to assign parameter values that are received as the content of a request message. In particular, TSubmitQueryButton generates an HTML control that sends the values in an HTML query form to the Web application. If the query form displays parameter values that should be sent to a provider on the application server, you can call AssignStrings with the ContentFields property of that Web request message.

See Also

Items

ParamValues

ContentFields

TQueryForm

TSubmitQueryButton

81.1.2.2.2 TXMLParams.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3985)

81.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 7289)	This is class XMLBrokr.INotifyConnectionChange.

81.2.1 INotifyConnectionChange Interface

Class Hierarchy

XMLBrokr.INotifyConnectionChange

Pascal

```
INotifyConnectionChange = interface;
```

C++

```
__interface INotifyConnectionChange;
```

File

XMLBrokr

Description

This is class XMLBrokr.INotifyConnectionChange.

81.2.1.1 INotifyConnectionChange Members

The following tables list the members exposed by INotifyConnectionChange.

Public Methods

	Name	Description
	ConnectionChange (see page 7290)	This is ConnectionChange, a member of class INotifyConnectionChange.

81.2.1.2 INotifyConnectionChange Methods

The methods of the INotifyConnectionChange class are listed here.

Public Methods

	Name	Description
	ConnectionChange (see page 7290)	This is ConnectionChange, a member of class INotifyConnectionChange.

81.2.1.2.1 INotifyConnectionChange.ConnectionChange Method

Pascal

```
procedure ConnectionChange(Sender: TComponent; Connected: Boolean);
```

C++

```
__fastcall ConnectionChange(TComponent * Sender, Boolean Connected);
```

Description

This is ConnectionChange, a member of class INotifyConnectionChange.

81.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FormatXML (see page 7291)	This is function XMLBrokr.FormatXML.

81.3.1 XMLBrokr.FormatXML Function

Pascal

```
function FormatXML(const Value: string; XMLOptions: TXMLOptions): string;
```

C++

```
UnicodeString FormatXML(const UnicodeString Value, TXMLOptions XMLOptions);
```

File

XMLBrokr

Description

This is function XMLBrokr.FormatXML.

81.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TXMLOption (see page 7292)	This is record XMLBrokr.TXMLOption.

81.4.1 XMLBrokr.TXMLOption Enumeration

Pascal

```
TXMLOption = (
  xoQuote
);
```

C++

```
enum TXMLOption {
  xoQuote
};
```

File

XMLBrokr

Description

This is record XMLBrokr.TXMLOption.

81.5 Types

The following table lists types in this documentation.

Types

Name	Description
TGetErrorResponseEvent (see page 7293)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 7293)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 7294)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 7294)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 7294)	This is type XMLBrokr.TXMLOptions.

81.5.1 XMLBrokr.TGetErrorResponseEvent Type

Pascal

```
TGetErrorResponseEvent = procedure (Sender: TObject; ErrorCount: Integer; XMLErrors: string;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ErrorCount: Integer; XMLErrors: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) ( TGetErrorResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetErrorResponseEvent.

81.5.2 XMLBrokr.TGetResponseEvent Type

Pascal

```
TGetResponseEvent = procedure (Sender: TObject;Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject;Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TGetResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetResponseEvent.

81.5.3 XMLBrokr.TRequestRecordsEvent Type

Pascal

```
TRequestRecordsEvent = procedure (Sender: TObject; Request: TWebRequest; out RecCount:
Integer; var OwnerData: OleVariant; var Records: string) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var OwnerData: OleVariant; var
Records: string) ( TRequestRecordsEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestRecordsEvent.

81.5.4 XMLBrokr.TRequestUpdateEvent Type

Pascal

```
TRequestUpdateEvent = procedure (Sender: TObject; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TRequestUpdateEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestUpdateEvent.

81.5.5 XMLBrokr.TXMLOptions Type

Pascal

```
TXMLOptions = set of TXMLOption;
```

C++

```
TXMLOption TXMLOptions;
```

File

XMLBrokr

Description

This is type XMLBrokr.TXMLOptions.

81.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sPostDelta (see page 7295)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 7295)	This is constant XMLBrokr.sProducer.
sRedirect (see page 7295)	This is constant XMLBrokr.sRedirect.

81.6.1 XMLBrokr.sPostDelta Constant

Pascal

```
sPostDelta = 'postdelta';
```

C++

```
sPostDelta = 'postdelta';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sPostDelta.

81.6.2 XMLBrokr.sProducer Constant

Pascal

```
sProducer = 'PRODUCER';
```

C++

```
sProducer = 'PRODUCER';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sProducer.

81.6.3 XMLBrokr.sRedirect Constant

Pascal

```
sRedirect = 'REDIRECT';
```

C++

```
sRedirect = 'REDIRECT';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sRedirect.

82 XMLDoc Namespace

This is namespace XMLDoc.

Topics

Name	Description
Classes (see page 7299)	The following table lists classes in this documentation.
Interfaces (see page 7429)	The following table lists interfaces in this documentation.
Functions (see page 7439)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7445)	The following table lists structs, records, enums in this documentation.
Types (see page 7449)	The following table lists types in this documentation.
Variables (see page 7453)	The following table lists variables in this documentation.

Classes

	Name	Description
	TXMLDocument (see page 7299)	TXMLDocument represents an XML (see page 7344) document.
	TXMLNode (see page 7348)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 7396)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 7411)	TXMLNodeList represents a set of nodes in an XML document.

Functions

	Name	Description
	CheckEncoding (see page 7439)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 7439)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 7440)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 7441)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 7441)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 7441)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 7442)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 7442)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 7443)	Raises an EXMLDocError (see page 7585) exception.
	XMLDocError (see page 7443)	Raises an EXMLDocError (see page 7585) exception.
	XMLStringToUnicodeString (see page 7443)	This is function XMLDoc.XMLStringToUnicodeString.
	XMLStringToWideString (see page 7443)	This is function XMLDoc.XMLStringToWideString.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 7429)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 7431)	This is class XMLDoc.IXMLNodeAccess.

Structs, Records, Enums

	Name	Description
	TNodeChange (see page 7445)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeClassInfo (see page 7446)	TNodeClassInfo and TNodeClassArray (see page 7449) represent information about the classes that implement XML node interfaces.
	TNodeListOperation (see page 7446)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7649) interface.
	TXMLDocumentSource (see page 7447)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 7448)	This is record XMLDoc.TXMLPrologItem.

Types

Name	Description
TNodeChangeEvent (see page 7449)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 7449)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 7450)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 7411) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 7451)	TXMLNodeArray is a dynamic array of TXMLNode (see page 7348) objects.
TXMLNodeClass (see page 7451)	TXMLNodeClass is a class reference to a TXMLNode (see page 7348) descendant.
TXMLNodeCollectionClass (see page 7452)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7396) descendant.

Variables

Name	Description
UseXSDBooleanStrings (see page 7453)	This is variable XMLDoc.UseXSDBooleanStrings.

82.1 Classes

The following table lists classes in this documentation.

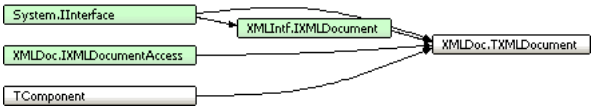
Classes

	Name	Description
	TXMLDocument (see page 7299)	TXMLDocument represents an XML (see page 7344) document.
	TXMLNode (see page 7348)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 7396)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 7411)	TXMLNodeList represents a set of nodes in an XML document.

82.1.1 TXMLDocument Class

TXMLDocument represents an XML (see page 7344) document.

Class Hierarchy



Pascal

```
TXMLDocument = class(TComponent, IInterface, IXMLDocument, IXMLDocumentAccess);
```

C++

```
class TXMLDocument : public TComponent, public IInterface, public IXMLDocument, public
IXMLDocumentAccess;
```

File

XMLDoc

Description

Use TXMLDocument to represent an XML (see page 7344) document. TXMLDocument can read an existing XML (see page 7344) document from a file, it can be associated with an in-memory string that is the contents of an XML (see page 7344) document, or it can create a new, empty XML (see page 7344) document.

You can use TXMLDocument directly to load an XML (see page 7344) document, read and edit it, and save any changes. You can also use TXMLDocument to access the objects generated by the XML (see page 7344) Data Binding wizard.

TXMLDocument uses an external DOM (Document Object Model) parser to analyze the XML (see page 7344) document. This parser is usually assigned by specifying the DOMVendor (see page 7337) property.

When you set the Active (see page 7333) property to true, TXMLDocument uses the DOM parser to parse the XML (see page 7344) document so that you can examine or modify it. In particular, the DocumentElement (see page 7335) property provides an interface to the root node of the document. You can use that interface to access its child nodes and to add or delete child nodes.

TXMLDocument implements the IXMLDocument (see page 7586) interface. Applications obtain this interface from TXMLDocument or from the nodes in the document.

When TXMLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TXMLDocument instance is automatically freed. When TXMLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner. When you add a TXMLDocument component from the component palette to a form or data module, it is automatically created with an Owner. When the TXMLDocument component is created using the global LoadXMLDocument (see page 7442) function (or by a function that the XML (see page 7344) Data Binding wizard generates to return the root node of the document), the function creates a TXMLDocument instance without an Owner.

See Also

IXMLNode (see page 7615)

EXMLDocError (see page 7585)

IXMLDocument (see page 7586)

LoadXMLDocument (see page 7442)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

82.1.1.1 TXMLDocument Members

The following tables list the members exposed by TXMLDocument.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.
	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7313)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.

	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7320)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 7327)	Sets the value of the Active property.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7307)	Responds after the last constructor has executed.
	Create (see page 7303)	This is the overview for the Create constructor overload.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
V	Destroy (see page 7304)	Disposes of a TXMLDocument (see page 7299) object.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.

	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.

Public Properties

	Name	Description
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.
R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.

	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

82.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TXMLDocument.Create (DOMString) (see page 7303)	Instantiates a TXMLDocument (see page 7299) object.
	TXMLDocument.Create (TComponent) (see page 7303)	Instantiates a TXMLDocument (see page 7299) object.

82.1.1.2.1 TXMLDocument.Create Constructor (DOMString)

Instantiates a TXMLDocument (see page 7299) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const AFileName: DOMString); overload; reintroduce;
```

C++

```
virtual __fastcall TXMLDocument(TComponent * AOwner);
__fastcall TXMLDocument(const DOMString AFileName);
```

Description

Call Create to instantiate a TXMLDocument (see page 7299) component at runtime. You do not need to explicitly create TXMLDocument (see page 7299) components that are added to forms or data modules at design time: they are created automatically.

AFileName specifies the file that the new TXMLDocument (see page 7299) instance represents.

AOwner is another component that is responsible for freeing the TXMLDocument (see page 7299) instance. It becomes the value of the Owner property.

TXMLDocument (see page 7299) components that are created at runtime without an owner are freed automatically when all references to their IXMLDocument (see page 7586) interface are released. This differs from TXMLDocument (see page 7299) components added to a form or data module at design time or those created with an owner. When added to a form or data module at design time, the object is explicitly freed by its Owner (the form or data module).

See Also

FileName (see page 7339)

IXMLDocument (see page 7586)

Owner

82.1.1.3 TXMLDocument.Destroy Destructor

Disposes of a TXMLDocument (see page 7299) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDocument();
```

Description

It is not necessary to explicitly free a TXMLDocument (see page 7299) component. If the component is created with an Owner, that Owner handles the task of freeing the document. If the component is created without an Owner, it is freed automatically when the reference count on its IXMLDocument (see page 7586) interface drops to zero.

See Also

Create (see page 7303)

82.1.1.4 TXMLDocument Methods

The methods of the TXMLDocument class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.
	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7313)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.

	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7320)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 7327)	Sets the value of the Active property.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7307)	Responds after the last constructor has executed.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.
	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.

82.1.1.4.1 TXMLDocument._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke

QueryInterface

_Release

ComObject

_AddRef (see page 3504)

82.1.1.4.2 TXMLDocument._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke

QueryInterface

_AddRef

ComObject

82.1.1.4.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLDocument.AddChild (DOMString) (see page 7307)	Creates and adds a node to the child nodes of this document.
	TXMLDocument.AddChild (DOMString, DOMString) (see page 7307)	Creates and adds a node to the child nodes of this document.

82.1.1.4.3.1 TXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 7335) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 7344) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 7618)

CreateNode (see page 7309)

CreateElement (see page 7309)

ChildNodes (see page 7335)

DocumentElement (see page 7335)

82.1.1.4.4 TXMLDocument.AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject::TObject does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 3368)

OnCreate

System::TObject::Create (see page 3366)

82.1.1.4.5 TXMLDocument.AssignParseOptions Method

Pascal

```
procedure AssignParseOptions;
```

C++

```
__fastcall AssignParseOptions();
```

Description

This is AssignParseOptions, a member of class TXMLDocument.

82.1.1.4.6 TXMLDocument.CheckActive Method

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

This is CheckActive, a member of class TXMLDocument.

82.1.1.4.7 TXMLDocument.CheckAutoSave Method

Pascal

```
procedure CheckAutoSave; dynamic;
```

C++

```
__fastcall CheckAutoSave();
```

Description

This is CheckAutoSave, a member of class TXMLDocument.

82.1.1.4.8 TXMLDocument.CheckDOM Method

Pascal

```
procedure CheckDOM; dynamic;
```

C++

```
__fastcall CheckDOM();
```

Description

This is CheckDOM, a member of class TXMLDocument.

82.1.1.4.9 TXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7335) list of any node in the document (including this document's ChildNodes (see page 7335) property). It can also be assigned to the DocumentElement (see page 7335) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7307) method.

See Also

AddChild (see page 7618)

ChildNodes (see page 7335)

DocumentElement (see page 7335)

AddChild (see page 7307)

CreateNode (see page 7309)

82.1.1.4.10 TXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 7344) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7335) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 7335) property). It can also be assigned to the DocumentElement (see page 7335) property to replace the entire data hierarchy of this document.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7307) method.

See Also

AddChild (see page 7618)

ChildNodes (see page 7335)

AttributeNodes (see page 7390)

DocumentElement (see page 7335)

AddChild (see page 7307)

CreateElement (see page 7309)

CreateDOMNode (see page 7439)

82.1.1.4.11 TXMLDocument.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor

DefineBinaryProperty

DefineProperties

DefineProperty

82.1.1.4.12 TXMLDocument.DoAfterClose Method

Pascal

```
procedure DoAfterClose; dynamic;
```

C++

```
__fastcall DoAfterClose();
```

Description

This is DoAfterClose, a member of class TXMLDocument.

82.1.1.4.13 TXMLDocument.DoAfterOpen Method

Pascal

```
procedure DoAfterOpen; dynamic;
```

C++

```
__fastcall DoAfterOpen();
```

Description

This is DoAfterOpen, a member of class TXMLDocument.

82.1.1.4.14 TXMLDocument.DoBeforeClose Method

Pascal

```
procedure DoBeforeClose; dynamic;
```

C++

```
__fastcall DoBeforeClose();
```

Description

This is DoBeforeClose, a member of class TXMLDocument.

82.1.1.4.15 TXMLDocument.DoBeforeOpen Method

Pascal

```
procedure DoBeforeOpen; dynamic;
```

C++

```
__fastcall DoBeforeOpen();
```

Description

This is DoBeforeOpen, a member of class TXMLDocument.

82.1.1.4.16 TXMLDocument.DoNodeChange Method

Pascal

```
procedure DoNodeChange(const Node: IXMLNode; ChangeType: TNodeChange; BeforeOperation:
Boolean);
```

C++

```
__fastcall DoNodeChange(const IXMLNode Node, TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

This is DoNodeChange, a member of class TXMLDocument.

82.1.1.4.17 TXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

Call GeneratePrefix to generate a new namespace prefix that is not already used in the document to represent a namespace URI. GeneratePrefix uses the NSPrefixBase (see page 7341) property to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase (see page 7341) followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase (see page 7341) is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

Node (see page 7340) is the interface for the node which will use the newly-generated namespace prefix.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase (see page 7341), they will generate the same namespace prefixes.

See Also

NSPrefixBase (see page 7341)

82.1.1.4.18 TXMLDocument.GetActive Method

Returns the value of the Active property.

Pascal

```
function GetActive: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active property.

See Also

Active (see page 7607)

82.1.1.4.19 TXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState property.

See Also

AsyncLoadState (see page 7608)

82.1.1.4.20 TXMLDocument.GetChildNodeClass Method

Pascal

```
function GetChildNodeClass(const Node: IDOMNode): TXMLNodeClass; virtual;
```

C++

```
virtual __fastcall TXMLNodeClass GetChildNodeClass(const IDOMNode Node);
```

Description

This is GetChildNodeClass, a member of class TXMLDocument.

82.1.1.4.21 TXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes property.

See Also

ChildNodes (see page 7609)

82.1.1.4.22 TXMLDocument.GetDocBinding Method

Replaces the document element with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class and tag name for the document element. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7335)

RegisterDocBinding (see page 7324)

ChildNodes (see page 7335)

82.1.1.4.23 TXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement property.

See Also

DocumentElement (see page 7609)

82.1.1.4.24 TXMLDocument.GetDocumentNode Method

Returns the value of the Node property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node property.

See Also

Node (see page 7611)

82.1.1.4.25 TXMLDocument.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class TXMLDocument.

82.1.1.4.26 TXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument property.

See Also

DOMDocument (see page 7609)

82.1.1.4.27 TXMLDocument.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class TXMLDocument.

82.1.1.4.28 TXMLDocument.GetEncoding Method

Returns the value of the Encoding property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding property.

See Also

Encoding (see page 7610)

82.1.1.4.29 TXMLDocument.GetFileName Method

Returns the value of the FileName property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName property.

See Also

FileName (see page 7610)

82.1.1.4.30 TXMLDocument.GetModified Method

Returns the value of the Modified property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified property.

See Also

Modified (see page 7611)

82.1.1.4.31 TXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr property.

See Also

NodeIndentStr (see page 7612)

82.1.1.4.32 TXMLDocument.GetOptions Method

Returns the value of the Options property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options property.

See Also

Options (see page 7612)

82.1.1.4.33 TXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions property.

See Also

ParseOptions (see page 7613)

82.1.1.4.34 TXMLDocument.GetPrologNode Method

Pascal

```
function GetPrologNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetPrologNode();
```

Description

This is GetPrologNode, a member of class TXMLDocument.

82.1.1.4.35 TXMLDocument.GetPrologValue Method

Pascal

```
function GetPrologValue(PrologItem: TXMLPrologItem; const Default: DOMString = ''): DOMString;
```

C++

```
__fastcall DOMString GetPrologValue(TXMLPrologItem PrologItem, const DOMString Default = '');
```

Description

This is GetPrologValue, a member of class TXMLDocument.

82.1.1.4.36 TXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef property.

See Also

SchemaRef (see page 7613)

82.1.1.4.37 TXMLDocument.GetStandAlone Method

Returns the value of the StandAlone property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone property.

See Also

StandAlone (see page 7614)

82.1.1.4.38 TXMLDocument.GetVersion Method

Returns the value of the Version property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version property.

See Also

Version (see page 7614)

82.1.1.4.39 TXMLDocument.GetXML Method

Returns the value of the XML property.

Pascal

```
function GetXML: TStrings;
```

C++

```
__fastcall TStrings GetXML();
```

Description

GetXML is the read implementation of the XML property.

See Also

XML (see page 7614)

82.1.1.4.40 TXMLDocument.InternalSetPrologValue Method

Pascal

```
function InternalSetPrologValue(const PrologNode: IXMLNode; const Value: Variant; PrologItem:
TXMLPrologItem): string;
```

C++

```
__fastcall UnicodeString InternalSetPrologValue(const IXMLNode PrologNode, const Variant
Value, TXMLPrologItem PrologItem);
```

Description

This is InternalSetPrologValue, a member of class TXMLDocument.

82.1.1.4.41 TXMLDocument.IsEmptyDoc Method

Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the TXMLDocument (see page 7299) instance represents an empty document.

IsEmptyDoc returns true if the DOMDocument (see page 7336) property is not set (for example, if there is no DOM implementation) or if this object represents a document with no child nodes.

See Also

XML (see page 7344)

ChildNodes (see page 7335)

DOMDocument (see page 7336)

82.1.1.4.42 TXMLDocument.LoadData Method

Pascal

```
procedure LoadData; dynamic;
```

C++

```
__fastcall LoadData();
```

Description

This is LoadData, a member of class TXMLDocument.

82.1.1.4.43 TXMLDocument.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

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

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

82.1.1.4.44 TXMLDocument.LoadFromFile Method

Loads an XML (see page 7344) document and activates it.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString = '');
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName = '');
```

Description

Call LoadFromFile to load the XML (see page 7344) document specified by AFileName and set the Active (see page 7333) property to true so that you can examine or modify the document.

AFileName is the name of the XML (see page 7344) document to load from disk. If AFileName is an empty string, TXMLDocument (see page 7299) uses the value of the FileName (see page 7339) property. If AFileName is not an empty string, TXMLDocument (see page 7299) changes the FileName (see page 7339) property to AFileName.

Once you have loaded an XML (see page 7344) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7325) method.

See Also

SaveToFile (see page 7325)

FileName (see page 7339)

Active (see page 7333)

Refresh (see page 7323)

LoadFromXML (see page 7322)

LoadFromStream (see page 7321)

82.1.1.4.45 TXMLDocument.LoadFromStream Method

Loads an XML (see page 7344) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 7344) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 7344) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 7344) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 7333) property to true.

See Also

SaveToStream (see page 7325)

LoadFromFile (see page 7321)

Active (see page 7333)

LoadFromXML (see page 7322)

82.1.1.4.46 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	TXMLDocument.LoadFromXML (AnsiString) (see page 7322)	Loads a string representation of an XML (see page 7344) document and activates it.
	TXMLDocument.LoadFromXML (DOMString) (see page 7322)	Loads a string representation of an XML (see page 7344) document and activates it.

82.1.1.4.46.1 TXMLDocument.LoadFromXML Method (AnsiString)

Loads a string representation of an XML (see page 7344) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: AnsiString); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 7344) document. Unlike the XML (see page 7344) property, which lets you assign XML (see page 7344) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 7344) document as a whole.

The XML (see page 7344) parameter is a string containing the text of an XML (see page 7344) document. It should represent the XML (see page 7344) text encoded using UTF-8, Unicode, or UTF-16.

After assigning the XML (see page 7344) property as the contents of the document, LoadFromXML sets the Active (see page 7333) property to true.

See Also

SaveToXML (see page 7326)

LoadFromFile (see page 7321)

Active (see page 7333)

LoadFromStream (see page 7321)

XML (see page 7344)

82.1.1.4.47 TXMLDocument.NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance.

See Also

System::TObject::FreeInstance (see page 3386)

System::TObject::InstanceSize (see page 3389)

System::TObject::InitInstance (see page 3389)

82.1.1.4.48 TXMLDocument.ReadDOMVendor Method

Pascal

```
procedure ReadDOMVendor(Reader: TReader);
```

C++

```
__fastcall ReadDOMVendor(TReader Reader);
```

Description

This is ReadDOMVendor, a member of class TXMLDocument.

82.1.1.4.49 TXMLDocument.Refresh Method

Updates the parsed XML (see page 7344) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 7344) document specified by FileName (see page 7339) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 7344) document (which can be traversed starting with DocumentElement (see page 7335)) to reflect the current contents of the file specified by FileName (see page 7339).

The Active (see page 7333) property must be true when you call Refresh. If the XML (see page 7344) document is not active when you call Refresh, TXMLDocument (see page 7299) raises an EXMLDocError (see page 7585) exception.

Note: Do not call Refresh after making changes directly to the XML (see page 7344) property, because assigning a value to XML (see page 7344) automatically closes the XML (see page 7344) document. To refresh the XML (see page 7344) document after assigning a new value to XML (see page 7344), set the Active (see page 7333) property to true.

See Also

FileName (see page 7339)

Active (see page 7333)

DOMDocument (see page 7336)

DocumentElement (see page 7335)

Resync (see page 7325)

82.1.1.4.50 TXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7335)

GetDocBinding (see page 7314)

ChildNodes (see page 7335)

82.1.1.4.51 TXMLDocument.ReleaseDoc Method

Pascal

```
procedure ReleaseDoc(const CheckSave: Boolean = True); dynamic;
```

C++

```
__fastcall ReleaseDoc(const Boolean CheckSave = True);
```

Description

This is ReleaseDoc, a member of class TXMLDocument.

82.1.1.4.52 TXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7335)

Refresh (see page 7323)

82.1.1.4.53 TXMLDocument.SaveToFile Method

Saves the XML (see page 7344) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString = ''); dynamic;
```

C++

```
__fastcall SaveToFile(const DOMString AFileName = '');
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 7344) document.

AFileName is the name of the file to save. If AFileName is an empty string, TXMLDocument (see page 7299) uses the value of the FileName (see page 7339) property.

See Also

LoadFromFile (see page 7321)

FileName (see page 7339)

SaveToXML (see page 7326)

SaveToStream (see page 7325)

Using Code That the XML Data Binding Wizard Generates

82.1.1.4.54 TXMLDocument.SaveToStream Method

Saves the XML (see page 7344) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 7344) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 7344) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 7321)

SaveToXML (see page 7326)

SaveToFile (see page 7325)

82.1.1.4.55 TXMLDocument.SaveToUTF8String Method

Pascal

```
procedure SaveToUTF8String(var XML: UTF8String);
```

C++

```
__fastcall SaveToUTF8String(UTF8String XML);
```

Description

This is SaveToUTF8String, a member of class TXMLDocument.

82.1.1.4.56 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	TXMLDocument.SaveToXML (DOMString) (see page 7326)	Saves the XML (see page 7344) document to a string-type variable.
	TXMLDocument.SaveToXML (UTF8String) (see page 7326)	Saves the XML (see page 7344) document to a string-type variable.
	TXMLDocument.SaveToXML (WideString) (see page 7326)	Saves the XML (see page 7344) document to a string-type variable.

82.1.1.4.56.1 TXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 7344) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: WideString); overload;
procedure SaveToXML(var XML: UTF8String); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
__fastcall SaveToXML(BSTR XML);
__fastcall SaveToXML(UTF8String XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 7344) document to the string-type variable specified by XML (

see page 7344). SaveToXML writes the contents of XML (see page 7344) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 7344) parameter.

Unlike the XML (see page 7344) property, which lets you write individual lines from the XML (see page 7344) document, SaveToXML writes the entire text of the XML (see page 7344) document.

See Also

LoadFromXML (see page 7322)

SaveToStream (see page 7325)

SaveToFile (see page 7325)

XML (see page 7344)

82.1.1.4.57 TXMLDocument.SaveToXMLStrings Method

Pascal

```
procedure SaveToXMLStrings;
```

C++

```
__fastcall SaveToXMLStrings();
```

Description

This is SaveToXMLStrings, a member of class TXMLDocument.

82.1.1.4.58 TXMLDocument.SetActive Method

Sets the value of the Active property.

Pascal

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

C++

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

Description

SetActive is the write implementation of the Active property.

Value is the new value to set.

See Also

Active (see page 7607)

82.1.1.4.59 TXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 7609)

82.1.1.4.60 TXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 7609)

82.1.1.4.61 TXMLDocument.SetEncoding Method

Sets the value of the Encoding property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 7610)

82.1.1.4.62 TXMLDocument.SetFileName Method

Sets the value of the FileName property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName property.

Value is the name of an XML document or, if the vendor supports it, an URL for the XML document.

See Also

FileName (see page 7610)

82.1.1.4.63 TXMLDocument.SetModified Method

Pascal

```
procedure SetModified(const Value: Boolean);
```

C++

```
__fastcall SetModified(const Boolean Value);
```

Description

This is SetModified, a member of class TXMLDocument.

82.1.1.4.64 TXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 7612)

82.1.1.4.65 TXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML document progresses to a certain point.

Value is a method that is called when the AsyncLoadState property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 7608)

82.1.1.4.66 TXMLDocument.SetOptions Method

Sets the value of the Options property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions is the write implementation of the Options property.

Value is the set of options to assign.

See Also

Options (see page 7612)

82.1.1.4.67 TXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions is the write implementation of the ParseOptions property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 7613)

82.1.1.4.68 TXMLDocument.SetPrologValue Method

Pascal

```
procedure SetPrologValue(const Value: Variant; PrologItem: TXMLPrologItem);
```

C++

```
__fastcall SetPrologValue(const Variant Value, TXMLPrologItem PrologItem);
```

Description

This is SetPrologValue, a member of class TXMLDocument.

82.1.1.4.69 TXMLDocument.SetStandAlone Method

Sets the value of the StandAlone property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 7614)

82.1.1.4.70 TXMLDocument.SetVersion Method

Sets the value of the Version property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 7614)

82.1.1.4.71 TXMLDocument.SetXML Method

Sets the value of the XML property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML property.

Value contains the raw (unparsed) XML to assign.

See Also

XML (see page 7614)

82.1.1.4.72 TXMLDocument.SetXMLStrings Method

Pascal

```
procedure SetXMLStrings(const Value: string);
```

C++

```
__fastcall SetXMLStrings(const UnicodeString Value);
```

Description

This is SetXMLStrings, a member of class TXMLDocument.

82.1.1.4.73 TXMLDocument.WriteDOMVendor Method

Pascal

```
procedure WriteDOMVendor(Writer: TWriter);
```

C++

```
__fastcall WriteDOMVendor(TWriter Writer);
```

Description

This is WriteDOMVendor, a member of class TXMLDocument.

82.1.1.4.74 TXMLDocument.XMLStringsChanging Method

Pascal

```
procedure XMLStringsChanging(Sender: TObject);
```

C++

```
__fastcall XMLStringsChanging(TObject * Sender);
```

Description

This is XMLStringsChanging, a member of class TXMLDocument.

82.1.1.5 TXMLDocument Properties

The properties of the TXMLDocument class are listed here.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Properties

	Name	Description
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.

R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.
	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

82.1.1.5.1 TXMLDocument.Active Property

Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use TXMLDocument (see page 7299) to examine or change the contents of an XML (see page 7344) document, it must be active. Setting Active to true causes TXMLDocument (see page 7299) to do the following:

1.Generate a BeforeOpen (see page 7347) event.

2.If necessary, load the DOM parser and assign the DOMImplementation (see page 7337) property.

3.Parse the XML (see page 7344) document specified by FileName (see page 7339) or, if FileName (see page 7339) is an empty string, by the XML (see page 7344) property using the options specified by ParseOptions (see page 7342). If both FileName (see page 7339) and XML (see page 7344) are empty strings, TXMLDocument (see page 7299) creates an empty document. The interface for the parsed document is assigned to the DOMDocument (see page 7336) property.

4. Generate an AfterOpen (see page 7345) event.

Note: Once the DOMDocument (see page 7336) property is assigned, you can also use the DocumentElement (see page 7335) property.

Setting Active to false causes TXMLDocument (see page 7299) to do the following:

1.Generate a BeforeClose (see page 7346) event.

2.Release (see page 3732) all of the DOM interfaces, such as DOMDocument (see page 7336) and DocumentElement (

see page 7335), that represent the parsed XML (see page 7344) document.

3. Generate an AfterClose (see page 7344) event.

See Also

DOMDocument (see page 7336)

DocumentElement (see page 7335)

FileName (see page 7339)

XML (see page 7344)

ParseOptions (see page 7342)

BeforeOpen (see page 7347)

AfterOpen (see page 7345)

domimplementation (see page 7337)

BeforeClose (see page 7346)

AfterClose (see page 7344)

Using TXMLDocument

82.1.1.5.2 TXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 7344) document asynchronously. AsyncLoadState is only meaningful when the value of ParseOptions (see page 7342) is poAsyncLoad.

The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 7344) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 7344) document.
2	Loaded (see page 7320): The parser has finished reading and parsing the structure of the XML (see page 7344) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 7344) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 7344) document.

See Also

OnAsyncLoad (see page 7347)

ParseOptions (see page 7342)

82.1.1.5.3 TXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 7335), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

Note: ChildNodes is not available unless the XML (see page 7344) document is Active (see page 7333).

See Also

DocumentElement (see page 7335)

Active (see page 7333)

82.1.1.5.4 TXMLDocument.DocBindingInfo Property

Pascal

```
property DocBindingInfo: TNodeClassArray;
```

C++

```
__property TNodeClassArray DocBindingInfo;
```

Description

This is DocBindingInfo, a member of class TXMLDocument.

82.1.1.5.5 TXMLDocument.DocSource Property

Pascal

```
property DocSource: TXMLDocumentSource;
```

C++

```
__property TXMLDocumentSource DocSource;
```

Description

This is DocSource, a member of class TXMLDocument.

82.1.1.5.6 TXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 7344) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 7344) document. Because XML (see page 7344) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 7344) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

Note: For some DOM implementations, if you set DocumentElement, all of the new nodes must already be associated with this document. That is, the new value should be a node that came from the existing node hierarchy or that was created using the CreateNode (see page 7309) method.

The IXMLNode (see page 7615) interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

Note: DocumentElement is not available unless the XML (see page 7344) document is Active (see page 7333).

See Also

DOMDocument (see page 7336)

Active (see page 7333)

DocumentElement

CreateNode (see page 7309)

GetDocBinding (see page 7314)

Using TXMLDocument

Working with XML Nodes

82.1.1.5.7 TXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 7344) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument (see page 7482) interface for the parsed XML (see page 7344) document. This is the interface supported by the DOM parser for the document that is represented in raw (unparsed) form by the FileName (see page 7339) or XML (see page 7344) property.

DOMDocument provides access to the IDOMImplementation (see page 7504) and IDOMElement (see page 7494) interfaces that are surfaced as the DOMImplementation (see page 7337) and DocumentElement (see page 7335) properties. It also provides access to an IDOMDocumentType (see page 7490) interface that describes the structure of the XML (see page 7344) document, and supports a number of methods for creating elements within the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 7335). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Level 1 Specification.

Note: DOMDocument is not available unless the XML (see page 7344) document is Active (see page 7333).

See Also

Active (see page 7333)

DocumentElement (see page 7335)

domimplementation (see page 7337)

XML (see page 7344)

FileName (see page 7339)

82.1.1.5.8 TXMLDocument.DOMImplementation Property

Pascal

```
property DOMImplementation: IDOMImplementation;
```

C++

```
__property IDOMImplementation DOMImplementation;
```

Description

This is DOMImplementation, a member of class TXMLDocument.

82.1.1.5.9 TXMLDocument.DOMParseOptions Property

Pascal

```
property DOMParseOptions: IDOMParseOptions;
```

C++

```
__property IDOMParseOptions DOMParseOptions;
```

Description

This is DOMParseOptions, a member of class TXMLDocument.

82.1.1.5.10 TXMLDocument.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class TXMLDocument.

82.1.1.5.11 TXMLDocument.DOMVendor Property

Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.

Pascal

```
property DOMVendor: TDOMVendor;
```

C++

```
__property TDOMVendor DOMVendor;
```

Description

DOMVendor is the object that represents a registered DOM implementation. TXMLDocument (see page 7299) uses this object to obtain an IDOMImplementation (see page 7504) interface, which it can use to access all the other DOM interfaces needed for parsing and manipulating the XML (see page 7344) document. Once DOMVendor is assigned, TXMLDocument (see page 7299) obtains the IDOMImplementation (see page 7504) interface (when needed) and assigns it to the DOMImplementation (see page 7337) property.

DOMVendor must be registered before it can be used to obtain an IDOMImplementation (see page 7504) interface. For a list of all the currently registered DOM implementations, use the global DOMVendors (see page 7568) variable. To obtain a specific DOM implementation, given its description, use the global GetDOMVendor (see page 7560) function. Use the RegisterDOMVendor (see page 7562) procedure to register a new DOM implementation.

You must assign a value to either DOMVendor or DOMImplementation (see page 7337) before you can set the Active (see page 7333) property to true.

See Also

domimplementation (see page 7337)

DOMVendors

Active (see page 7333)

RegisterDOMVendor

Active (see page 7333)

GetDOMVendor

Using TXMLDocument

82.1.1.5.12 TXMLDocument.Encoding Property

Indicates the character set that the XML (see page 7344) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 7344) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 7339)

DOMDocument (see page 7336)

XML (see page 7344)

82.1.1.5.13 TXMLDocument.FileName Property

Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Set FileName to specify the XML (see page 7344) document you want to examine or manipulate using the TXMLDocument (see page 7299) object. Once you have set FileName to the name of an XML (see page 7344) document, you can activate the XML (see page 7344) document and examine or manipulate it using the DocumentElement (see page 7335) property.

When you set the value of FileName, the XML (see page 7344) property automatically changes (at runtime) to represent the contents of that file.

Note: Setting the value of FileName automatically sets the Active (see page 7333) property to false.

After activating the XML (see page 7344) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7325) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

XML (see page 7344)

LoadFromFile (see page 7321)

SaveToFile (see page 7325)

DocumentElement (see page 7335)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

82.1.1.5.14 TXMLDocument.Modified Property

Indicates whether the XML (see page 7344) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read (see page 3727) Modified to determine whether the application has made any changes to the nodes in the XML (see page 7344) document. When Modified is true, the value of the XML (see page 7344) property may differ from the contents of the file specified by FileName (see page 7339). You can call the SaveToFile (see page 7325) method to update the XML (see page 7344) document file to reflect the changes.

Note: Modfied changes to false when you call the Refresh (see page 7323) or SaveToFile (see page 7325) method.

See Also

BeforeNodeChange (see page 7346)

AfterNodeChange (see page 7345)

Refresh (see page 7323)

SaveToFile (see page 7325)

XML (see page 7344)

FileName (see page 7339)

82.1.1.5.15 TXMLDocument.Node Property

Provides access to the document node for the XML (see page 7344) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read (see page 3727) Node to access the document node for the XML (see page 7344) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 7335) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 7335)

ChildNodes (see page 7335)

82.1.1.5.16 TXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how TXMLDocument (see page 7299) formats nodes that are added. When the Options (see page 7341) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 7344) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, TXMLDocument (see page 7299) inserts two spaces to represent each nesting level.

See Also

Options (see page 7341)

NestingLevel (see page 7380)

FormatXMLData (see page 7441)

XML (see page 7344)

82.1.1.5.17 TXMLDocument.NSPrefixBase Property

Specifies the base string for automatically-generated namespace prefixes.

Pascal

```
property NSPrefixBase: DOMString;
```

C++

```
__property DOMString NSPrefixBase;
```

Description

The AddChild (see page 7307) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node uses NSPrefixBase to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase, they will generate the same namespace prefixes.

See Also

AddChild (see page 7618)

GeneratePrefix (see page 7312)

82.1.1.5.18 TXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 7344) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7649) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 7344) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null (see page 5075) Variant (see page 3430) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.

doNamespaceDecl	When the AddChild (see page 7307) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 7307) creates the new namespace but does not add an attribute to declare it.
doAutoSave	When you close the document (set Active (see page 7333) to false), any changes are automatically saved back to the file specified by FileName (see page 7339) or to the XML (see page 7344) property.

See Also

Nodes (see page 7660)

Attributes (see page 7634)

AddChild (see page 7618)

Active (see page 7333)

FileName (see page 7339)

XML (see page 7344)

82.1.1.5.19 TXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 7344) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, TXMLDocument (see page 7299) raises an EXMLDocError (see page 7585) exception.

See Also

domimplementation (see page 7337)

EXMLDocError (see page 7585)

Using TXMLDocument

82.1.1.5.20 TXMLDocument.PrefixID Property

Pascal

```
property PrefixID: Integer;
```

C++

```
__property int PrefixID;
```

Description

This is PrefixID, a member of class TXMLDocument.

82.1.1.5.21 TXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 7344) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read (see page 3727) SchemaRef to determine the schema associated with this XML (see page 7344) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 7336)

82.1.1.5.22 TXMLDocument.StandAlone Property

Indicates whether the XML (see page 7344) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 7344) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 7336)

82.1.1.5.23 TXMLDocument.Version Property

Indicates the XML (see page 7344) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 7344) version number that the document uses. An XML (see page 7344) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 7344)

82.1.1.5.24 TXMLDocument.XML Property

Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read (see page 3727) XML to obtain the contents of the associated XML document. Set XML to specify the XML document you want to examine or manipulate. Once XML is set, you can activate the XML document and examine or manipulate it using the DocumentElement (see page 7335) property.

XML is a TStrings object. This means you can supply a value using the Strings editor at design time, or manipulate the XML line by line at runtime. To read or write the XML in the document as a single string, use the LoadFromXML (see page 7322) or SaveToXML (see page 7326) method.

See Also

FileName (see page 7339)

LoadFromXML (see page 7322)

SaveToXML (see page 7326)

DocumentElement (see page 7335)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

82.1.1.6 TXMLDocument Events

The events of the TXMLDocument class are listed here.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

82.1.1.6.1 TXMLDocument.AfterClose Event

Occurs after the XML (see page 7344) document is closed.

Pascal

```
property AfterClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterClose;
```

Description

Write (see page 3789) an AfterClose event handler to take specific action after the XML (see page 7344) document interfaces are released. For example, you can use an AfterClose event to update your user interface to indicate that the XML (see page 7344) document is no longer available.

AfterClose is the last event to occur when the Active (see page 7333) property is set to false.

See Also

BeforeClose (see page 7346)

Active (see page 7333)

82.1.1.6.2 TXMLDocument.AfterNodeChange Event

Occurs after a change is made to one of the nodes in the document.

Pascal

```
property AfterNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent AfterNodeChange;
```

Description

Write (see page 3789) an AfterNodeChange event handler to take specific action after a change is made to a node in the XML (see page 7344) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 7339) property is set to true.
2. A BeforeNodeChange (see page 7346) event occurs.
3. The change is made to the node.
4. An AfterNodeChange event occurs.

See Also

BeforeNodeChange (see page 7346)

Modified (see page 7339)

ChildNodes (see page 7335)

82.1.1.6.3 TXMLDocument.AfterOpen Event

Occurs immediately after the XML (see page 7344) document is opened.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write (see page 3789) an AfterOpen event handler to take specific action once the XML (see page 7344) document interfaces become available. For example, you can use an AfterOpen event to update your user interface to reflect the structure of the XML (see page 7344) document once it is opened.

AfterOpen is the last event to occur when the Active (see page 7333) property is set to true.

See Also

BeforeOpen (see page 7347)

Active (see page 7333)

82.1.1.6.4 TXMLDocument.BeforeClose Event

Occurs immediately before the XML (see page 7344) document is closed.

Pascal

```
property BeforeClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeClose;
```

Description

Write (see page 3789) a BeforeClose event handler to take specific action before the XML (see page 7344) document interfaces are released. For example, you can use an BeforeClose event to cache information about the XML (see page 7344) document before it becomes unavailable.

BeforeClose is the first event to occur when the Active (see page 7333) property is set to false.

See Also

AfterClose (see page 7344)

Active (see page 7333)

82.1.1.6.5 TXMLDocument.BeforeNodeChange Event

Occurs before a change is made to one of the nodes in the document.

Pascal

```
property BeforeNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent BeforeNodeChange;
```

Description

Write (see page 3789) a BeforeNodeChange event handler to take specific action before a change is made to a node in the XML (see page 7344) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 7339) property is set to true.
2. A BeforeNodeChange event occurs.
3. The change is made to the node.
4. An BeforeNodeChange event occurs.

See Also

AfterNodeChange (see page 7345)

Modified (see page 7339)

ChildNodes (see page 7335)

82.1.1.6.6 TXMLDocument.BeforeOpen Event

Occurs immediately before the XML (see page 7344) document is opened.

Pascal

```
property BeforeOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeOpen;
```

Description

Write (see page 3789) a BeforeOpen event handler to take specific action before the XML (see page 7344) document interfaces become available. For example, you can use a BeforeOpen event to provide feedback to the user that the XML (see page 7344) document is being parsed (which can be cleared in an AfterOpen (see page 7345) event handler).

BeforeOpen is the first event to occur when the Active (see page 7333) property is set to true.

See Also

AfterOpen (see page 7345)

Active (see page 7333)

82.1.1.6.7 TXMLDocument.OnAsyncLoad Event

Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Pascal

```
property OnAsyncLoad: TAsyncEventHandler;
```

C++

```
__property TAsyncEventHandler OnAsyncLoad;
```

Description

Write (see page 3789) a OnAsyncLoad event handler to respond appropriately during the asynchronous parsing of an XML (see page 7344) document. For example, OnAsyncLoad can update a progress indicator, or initiate operations that can't be performed until the document is fully parsed. OnAsyncLoad occurs every time the AsyncLoadState (see page 7334) property changes to reflect a new stage in the parsing process.

See Also

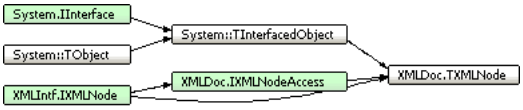
AsyncLoadState (see page 7334)

XML (see page 7344)

82.1.2 TXMLNode Class

TXMLNode represents a node in an XML document.

Class Hierarchy



Pascal

```
TXMLNode = class(TInterfacedObject, IXMLNode, IXMLNodeAccess);
```

C++

```
class TXMLNode : public TInterfacedObject, public IXMLNode, public IXMLNodeAccess;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNode as a base class for the classes it generates to represent nodes that have child nodes. For example, if the XML document includes the following:

```
<Address country="US">
  <Name>
    <First> John </First>
    <MI> Q. </MI>
    <Last> Public </Last>
  </Name>
  <Street> 123 Easy Street </Street>
  <City> Anytown </City>
  <State> CA </State>
</Address>
```

then the XML Data Binding Wizard generates TXMLNode descendants for the elements Address and Name. Child nodes and node attributes appear as properties of the TXMLNode descendant.

Note: The XML Data Binding Wizard also generates code that can create TXMLNode objects for attributes as well as element nodes. However, there are no special TXMLNode descendants for attribute nodes. Rather, each attribute is represented directly by TXMLNode.

TXMLNode implements the IXMLNode (see page 7615) interface. Each TXMLNode descendant implements a class-specific interface that descends from IXMLNode (see page 7615). Typically, applications do not work directly with TXMLNode. Instead, they use the IXMLNode (see page 7615) interface or the IXMLNode (see page 7615) descendant that is implemented by a TXMLNode descendant.

See Also

TXMLDocument (see page 7299)

TXMLNodeCollection (see page 7396)

IXMLNode (see page 7615)

TXMLNodeList (see page 7411)

82.1.2.1 TXMLNode Members

The following tables list the members exposed by TXMLNode.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.
	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.

R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7352)	Instantiates a Create object.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
V	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
V	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Attributes (see page 7634)	Represents the values of this node's attributes.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

82.1.2.2 TXMLNode.Create Constructor

Instantiates a Create object.

Pascal

```
constructor Create(const ADOMNode: IDOMNode; const AParentNode: TXMLNode; const OwnerDoc:
TXMLDocument);
```

C++

```
__fastcall TXMLNode(const IDOMNode ADOMNode, const TXMLNode AParentNode, const TXMLDocument
OwnerDoc);
```

Description

Create instantiates a new Create object to wrap an existing DOM node implementation. Using the first syntax,

ADOMNode is the interface for the DOM implementation of the node.

AParentNode is the implementation of the parent node of this node.

OwnerDoc represents the XML document in which the new node occurs.

Using the second syntax,

HostNode (see page 7393) is another Create object that represents the host node of this node. In this case, this node is a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call the Create constructor. Instead, new nodes are created automatically for an XML document as necessary when the application reads their interfaces, starting with the DocumentElement property of the TXMLDocument (see page 7299) object. To add new nodes to a document, applications call the parent node's AddChild (see page 7356) method.

See Also

OwnerDocument (see page 7394)

ParentNode (see page 7395)

DocumentElement (see page 7335)

AddChild (see page 7356)

82.1.2.3 TXMLNode.CreateHosted Constructor

Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.

Pascal

```
constructor CreateHosted(HostNode: TXMLNode);
```

C++

```
__fastcall TXMLNode(TXMLNode HostNode);
```

Description

CreateHosted instantiates a new TXMLNode (see page 7348) object to represent a specific view on an existing TXMLNode (see page 7348) object.

HostNode (see page 7393) is another TXMLNode (see page 7348) object that represents the host node of this node. The new node becomes a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call CreateHosted to create a hosted node. Instead, this method is called in code generated by the XML Data Binding Wizard, or by the CreateCollection (see page 7363) method of the host node.

HostNode (see page 7393) (or, if HostNode (see page 7393) is hosted, the host node of HostNode (see page 7393)) becomes the value of the new node's HostNode (see page 7393) property, and the new node is added to the HostedNodes (see page 7393) property list of the new host node.

See Also

HostedNodes (see page 7393)

HostNode (see page 7393)

OnHostAttrNotify (see page 7395)

OnHostChildNotify (see page 7396)

CreateCollection (see page 7363)

82.1.2.4 TXMLNode.Destroy Destructor

Frees the memory associated with the TXMLNode (see page 7348) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNode();
```

Description

Do not explicitly destroy TXMLNode (see page 7348) objects. As interfaced objects, XML nodes are destroyed automatically when the reference count of their IXMLNode (see page 7615) interface drops to zero.

See Also

RefCount (see page 3360)

82.1.2.5 TXMLNode Methods

The methods of the TXMLNode class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 7355)	Increments the reference count for this node.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7360)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.

	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7385)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.

82.1.2.5.1 TXMLNode._AddRef Method

Increments the reference count for this node.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IInterface (see page 3503) or IUnknown (see page 3828) method that is called when an application obtains a reference to this node's interface pointer. __AddRef increments the RefCount property. In addition, to ensure that the document object is alive, it also increments the reference count for the owning document.

When the caller is through with the interface, it calls _Release (see page 7356). This allows the node to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 7356)

OwnerDocument (see page 7394)

RefCount (see page 3360)

82.1.2.5.2 TXMLNode._Release Method

Decrements the reference count for this node.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IInterface (see page 3503) or IUnknown (see page 3828) method that is called when an application releases a reference to this node's interface pointer. __Release decrements the RefCount property. In addition, it also decrements the reference count for the owning document, which it incremented when the _AddRef (see page 7355) method was called.

When __Release causes the node's reference count to drop to zero, __Release causes the node to free itself.

See Also

_AddRef (see page 7355)

OwnerDocument (see page 7394)

RefCount (see page 3360)

82.1.2.5.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 7356)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, DOMString, TXMLNodeClass, Integer) (see page 7356)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, Integer) (see page 7356)	Adds a new child node to this node.

82.1.2.5.3.1 TXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected AddChild method. Only the first two versions of AddChild are available on the IXMLNode (see page 7615) interface. The third syntax is used by TXMLNode (see page 7348) descendants that the XML data binding wizard creates.

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

NamespaceURI identifies the namespace that includes the new node's definition. If NamespaceURI is omitted (the first syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI parameter, the namespace is the same as this node's NamespaceURI property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

NodeClass identifies the implementation class to use for the new child node. It must be TXMLNode (see page 7348) or one of its descendants.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 7391) property array.

See Also

ChildNodes (see page 7391)

AddChild (see page 7618)

CreateNode (see page 7309)

Options (see page 7341)

CreateElement (see page 7309)

NSPrefixBase (see page 7341)

DeclareNamespace (see page 7364)

82.1.2.5.4 TXMLNode.AddHostedNode Method

Adds a node to the HostedNodes (see page 7393) property list.

Pascal

```
procedure AddHostedNode(Node: TXMLNode);
```

C++

```
__fastcall AddHostedNode(TXMLNode Node);
```

Description

AddHostedNode is used internally to add nodes to the HostedNodes (see page 7393) array.

Node is the node to add to the array.

See Also

HostedNodes (see page 7393)

RemoveHostedNode (see page 7383)

82.1.2.5.5 TXMLNode.AttributeListNotify Method

Responds to notifications of changes in the list of attributes.

Pascal

```
procedure AttributeListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall AttributeListNotify(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the AttributeNodes (see page 7390) property calls AttributeListNotify when attribute nodes are inserted or removed, or when a new attribute node needs to be created. AttributeListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change occurred.

Node is the interface for the attribute node that was added or removed. If a new attribute node needs to be created, Node returns the new node.

IndexOrName is the name of the attribute when Operation is nlCreateNode.

BeforeOperation indicates whether the call occurs before the attribute node is added or removed.

See Also

AttributeNodes (see page 7390)

TNodeListNotification (see page 7450)

BeforeNodeChange (see page 7346)

AfterNodeChange (see page 7345)

82.1.2.5.6 TXMLNode.CheckNotHosted Method

Raises an EXMLDocError (see page 7585) exception if this is a hosted node.

Pascal

```
procedure CheckNotHosted;
```

C++

```
__fastcall CheckNotHosted();
```

Description

Applications can't call this protected method. It is used internally in methods that can only be used by the main node that corresponds to an element. If the node represents a hosted node (that is, if the node is used to represent a subset of the main, or host, node), then CheckNotHosted raises an EXMLDocError (see page 7585) exception.

For example, the CreateCollection (see page 7363) method calls CheckNotHosted, because only the main node for an element can host collection nodes.

See Also

HostNode (see page 7393)

HostedNodes (see page 7393)

EXMLDocError (see page 7585)

CreateCollection (see page 7363)

82.1.2.5.7 TXMLNode.CheckReadOnly Method

Raises an EXMLDocError (see page 7585) exception if this is a read-only node.

Pascal

```
procedure CheckReadOnly;
```

C++

```
__fastcall CheckReadOnly();
```

Description

Applications can't call this protected method. It is used internally in methods that change the value, children, or attributes of the node, to check whether the change is permitted, and if not, raise an exception.

See Also

GetReadOnly (see page 7377)

SetReadOnly (see page 7388)

ReadOnly (see page 7642)

EXMLDocError (see page 7585)

82.1.2.5.8 TXMLNode.CheckTextNode Method

Generates a DOM text node implementation if it does not yet exist.

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

Applications can't call this protected method. It is used internally to generate a DOM node implementation for text values.

If the XML node is an element that has child nodes (other than a single DOM text node), CheckTextNode raises an exception.

82.1.2.5.9 TXMLNode.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 7391) property calls ChildListNotify when child nodes are inserted or removed, or when a node needs to be created. ChildListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change is occurring.

Node is the interface for the child node about which this method is called. When nodes are added or removed, this is the node that is added or removed. When a node needs to be created, this parameter returns the newly created node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name of the child node to create.

BeforeOperation indicates whether the call occurs before the child node is added or removed.

See Also

ChildNodes (see page 7391)

AddChild (see page 7356)

TNodeListNotification (see page 7450)

BeforeNodeChange (see page 7346)

AfterNodeChange (see page 7345)

82.1.2.5.10 TXMLNode.ClearDocumentRef Method

Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

Applications can't call this protected method. The document that contains this node calls ClearDocumentRef when it is freed so that the node does not maintain a reference to a non-existent document.

In addition to setting the OwnerDocument (see page 7394) property to nil (Delphi) or NULL (C++), ClearDocumentRef propagates this call to all of this nodes child nodes and attribute nodes.

See Also

OwnerDocument (see page 7394)

ChildNodes (see page 7391)

AttributeNodes (see page 7390)

82.1.2.5.11 TXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

Applications can't call the protected CloneNode method. To create a copy of this node, use the CloneNode method of this node's IXMLNode (see page 7615) interface.

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

See Also

ChildNodes (see page 7391)

AddChild (see page 7356)

ParentNode (see page 7395)

CloneNode (see page 7619)

CloneNode (see page 7512)

82.1.2.5.12 TXMLNode.CreateAttributeList Method

Creates the object that implements the AttributeNodes (see page 7390) property.

Pascal

```
function CreateAttributeList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateAttributeList();
```

Description

Applications can't call the protected CreateAttributeList method. This method is used internally the first time an application reads the AttributeNodes (see page 7390) property.

CreateAttributeList creates an implementation class for the AttributeNodes (see page 7390) property, telling it to call the AttributeListNotify (see page 7358) method when it changes the list of attributes for this node, and filling it with nodes based on the current attributes of this node.

See Also

AttributeNodes (see page 7390)

AttributeListNotify (see page 7358)

82.1.2.5.13 TXMLNode.CreateAttributeNode Method

Creates a new TXMLNode (see page 7348) object for an attribute of this node.

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateAttributeNode method. TXMLNode (see page 7348) uses this method internally to generate the nodes that populate the AttributeNodes (see page 7390) property.

CreateAttributeNode creates a new TXMLNode (see page 7348) object for the specified IDOMNode (see page 7509) interface and returns its interface.

ADOMNode is the interface for the DOM implementation's version of the attribute node.

See Also

AttributeNodes (see page 7390)

82.1.2.5.14 TXMLNode.CreateChildList Method

Creates the object that implements the ChildNodes (see page 7391) property.

Pascal

```
function CreateChildList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateChildList();
```

Description

Applications can't call the protected CreateChildList method. This method is used internally the first time an application reads the ChildNodes (see page 7391) property.

CreateChildList creates an implementation class for the ChildNodes (see page 7391) property, telling it to call the ChildListNotify (see page 7360) method when it changes the list of attributes for this node, and filling it with nodes that are based on the current children of this node.

See Also

ChildNodes (see page 7391)

ChildListNotify (see page 7360)

82.1.2.5.15 TXMLNode.CreateChildNode Method

Creates a new TXMLNode (see page 7348) object for a child node of this node.

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateChildNode method. This method is used internally to create the implementation object for a new child node.

CreateChildNode creates a new TXMLNode (see page 7348) descendant for the specified IDOMNode (see page 7509) interface, with this node as the parent. If the node name of the DOM node matches one of the names listed by the ChildNodeClasses (see page 7390) property, the corresponding descendant of TXMLNode (see page 7348) becomes the implementation class of the new node. Otherwise, the new node is an instance of TXMLNode (see page 7348).

ADOMNode is the interface for the DOM implementation's version of the node.

CreateChildNode returns the interface for the newly created child node.

See Also

Create (see page 7352)

ChildNodeClasses (see page 7390)

CreateDOMNode (see page 7439)

82.1.2.5.16 TXMLNode.CreateCollection Method

Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemInterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemInterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

Applications can't call the protected CreateCollection method. It is used internally to generate the collections that are listed in the HostedNodes (see page 7393) property.

CollectionClass specifies the class (a TXMLNodeCollection (see page 7396) descendant) that implements the collection of child nodes.

ItemInterface specifies the interface for each child node in the collection.

ItemTag specifies the tag name for each child node in the collection.

ItemNS specifies the namespace in which ItemTag is defined.

CreateCollection returns the new hosted collection node. This new node has the same parent node, DOM interface, attribute nodes, and child nodes as this node, but it also maintains a list of only those child nodes with the specified tag and interface. The new collection node lists this node as the value of its HostNode (see page 7393) property.

Note: CreateCollection raises an exception if this is a hosted node.

See Also

HostedNodes (see page 7393)

ChildNodes (see page 7391)

ParentNode (see page 7395)

DOMNode (see page 7392)

HostNode (see page 7393)

ItemInterface (see page 7410)

itemtag (see page 7411)

itemns (see page 7410)

82.1.2.5.17 TXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected DeclareNamespace method. DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 7341)

AttributeNodes (see page 7390)

FindNamespaceDecl (see page 7366)

DeclareNamespace (see page 7620)

82.1.2.5.18 TXMLNode.DOMElement Method

Returns the IDOMElement (see page 7494) interface for this node if it exists.

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

Application's can't call the protected DOMElement method. TXMLNode (see page 7348) uses this method internally to obtain the DOM element interface, which it uses for working with attributes.

If the TXMLNode (see page 7348) instance does not represent an element node, DOMElement raises an EXMLDocError (see page 7585) exception.

See Also

AttributeNodes (see page 7390)

GetAttribute (see page 7367)

SetAttribute (see page 7384)

82.1.2.5.19 TXMLNode.DoNodeChange Method

Generates a BeforeNodeChange or AfterNodeChange event on this node's document.

Pascal

```
procedure DoNodeChange(ChangeType: TNodeChange; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall DoNodeChange(TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

Applications can't call the protected DoNodeChange method. It is called internally before and after every change that is made to the node. DoNodeChange checks whether the node's OwnerDocument (see page 7394) property is set, and if so, generates a BeforeNodeChange or AfterNodeChange event on that document, as appropriate.

ChangeType indicates the type of change that is occurring to the node.

BeforeOperation is true if the change is about to occur, false if the change has just occurred.

See Also

AfterNodeChange (see page 7345)

BeforeNodeChange (see page 7346)

OwnerDocument (see page 7394)

82.1.2.5.20 TXMLNode.FindHostedNode Method

Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

Applications can't call the protected FindHostedNode method. Descendant classes call this method to locate a hosted node of a particular class type.

FindHostedNode searches the nodes in the HostedNodes (see page 7393) list and returns the first one it finds that is an instance of descendant of NodeClass. If this node does not have a hosted node of the specified class, FindHostedNode returns

nil (Delphi) or NULL (C++).

See Also

HostedNodes (see page 7393)

CreateCollection (see page 7363)

82.1.2.5.21 TXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected FindNamespaceDecl method. FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI specified by NamespaceURI. If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI is the namespace URI to locate.

See Also

FindNamespaceURI (see page 7366)

DeclareNamespace (see page 7364)

FindNamespaceDecl (see page 7620)

82.1.2.5.22 TXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected FindNamespaceURI method. FindNamespaceDecl (see page 7366) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 7373)

GetPrefix (see page 7376)

FindNamespaceDecl (see page 7366)

FindNamespaceURI (see page 7620)

82.1.2.5.23 TXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetAttribute method. GetAttribute returns the value of an attribute of this node.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, the value that GetAttribute returns is determined by the document's Options property. When Options include doAttrNull, GetAttribute returns a Null (see page 5075) Variant (see page 3430) for missing attributes. When Options does not include doAttrNull, GetAttribute returns an empty string as the value of missing attributes.

If this node is not an element node, GetAttribute raises an EXMLDocError (see page 7585) exception.

See Also

SetAttribute (see page 7384)

AttributeNodes (see page 7390)

GetAttributeNS (see page 7368)

82.1.2.5.24 TXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 7390) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 7390) property on the IXMLNode (see page 7615) interface for this node.

GetAttributeNodes generates the list that implements the AttributeNodes (see page 7390) property the first time an application reads the AttributeNodes (see page 7390) property.

See Also

GetAttribute (see page 7367)

AttributeNodes (see page 7390)

82.1.2.5.25 TXMLNode.GetAttributeNS Method

Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetAttributeNS method. GetAttributeNS returns the value of an attribute of this node.

AttrName is the name of the attribute.

NamespaceURI identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the specified attribute. If AttrName and NamespaceURI do not identify an attribute of this node, GetAttributeNS returns a Null (see page 5075) Variant (see page 3430).

If this node is not an element node, GetAttributeNS raises an EXMLDocError (see page 7585) exception.

See Also

GetAttribute (see page 7367)

Attributes (see page 7634)

GetNamespaceURI (see page 7373)

SetAttributeNS (see page 7385)

AttributeNodes (see page 7390)

82.1.2.5.26 TXMLNode.GetChildNodeClasses Method

Returns a list of the classes that can implement child nodes of this node.

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

GetChildNodeClasses is used internally to obtain a list of the classes that can be used to represent child nodes of this node.

If this node is not a hosted node, GetChildNodeClasses returns an array containing every class that has been registered using the RegisterChildNode (see page 7382) method. If this node is a hosted node, GetChildNodeClasses returns the ChildNodeClasses (see page 7390) property of the host node.

See Also

ChildNodeClasses (see page 7390)

RegisterChildNode (see page 7382)

HostNode (see page 7393)

82.1.2.5.27 TXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 7391) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the protected read implementation of the ChildNodes (see page 7391) property.

GetChildNodes generates the list that implements the ChildNodes (see page 7391) property the first time an application reads the ChildNodes (see page 7391) property.

See Also

GetChildValue (see page 7369)

ChildNodes (see page 7391)

82.1.2.5.28 TXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetChildValue method. GetChildValue returns the value of the node.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

If IndexOrName does not identify a child of this node, GetChildValue nil (Delphi) or NULL (C++). Otherwise, the return value depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, GetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the GetAttribute (

see page 7367) method instead.

See Also

ChildNodes (see page 7391)

ParentNode (see page 7395)

GetAttribute (see page 7367)

GetNodeValue (see page 7375)

82.1.2.5.29 TXMLNode.GetCollection Method

Returns the value of the Collection (see page 7391) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetCollection method. GetCollection returns the value of the Collection (see page 7391) property. If this node is not a member of a collection (that is, if it does not represent one of a set of child nodes that are all the same type), then GetCollection returns nil (Delphi) or NULL (C++).

See Also

Collection (see page 7391)

82.1.2.5.30 TXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 7392) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the protected read implementation of the DOMNode (see page 7392) property.

See Also

DOMNode (see page 7392)

82.1.2.5.31 TXMLNode.GetHasChildNodes Method

Indicates whether this node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetHasChildNodes method. GetHasChildNodes returns true if the underlying DOM node has any child nodes, false if it doesn't.

Note that even if the underlying DOM node has a child node, there may not be a TXMLNode (see page 7348) object to represent that child. For example, text nodes do not, by default, have corresponding TXMLNode (see page 7348) wrappers. Rather, the text becomes the value of the parent element node.

See Also

ChildNodes (see page 7391)

DOMNode (see page 7392)

GetIsTextElement (see page 7371)

82.1.2.5.32 TXMLNode.GetHostedNodes Method

Returns a list an array of the hosted nodes that share this node's attribute and child node lists.

Pascal

```
function GetHostedNodes: TXMLNodeArray;
```

C++

```
__fastcall TXMLNodeArray GetHostedNodes();
```

Description

GetHostedNodes is the read implementation of the HostedNodes (see page 7393) property. It returns a list an array of all nodes that represent aspects of this node, sharing its attributes and list of child nodes.

See Also

HostedNodes (see page 7393)

82.1.2.5.33 TXMLNode.GetHostNode Method

Returns the value of the HostNode (see page 7393) property.

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

GetHostNode returns the node that uses this node to represent a particular aspect of it, sharing the same attributes and list of child nodes. If this is not a hosted node, GetHostNode returns nil (Delphi) or NULL (C++).

See Also

HostNode (see page 7393)

82.1.2.5.34 TXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetIsTextElement method. GetIsTextElement returns true if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

where the node has a tag name (in this case Title) and a single text value (in this case 'Understanding XML').

GetIsTextElement returns false if the node is any other type of node.

Note: In the underlying DOM implementation, nodes for which GetIsTextElement is true are element nodes that have a single child node that is a text node. TXMLNode (see page 7348) flattens this out so that the child text node does not appear as a child node. Instead, the text can be accessed through the GetText (see page 7378) method.

See Also

GetText (see page 7378)

GetHasChildNodes (see page 7370)

82.1.2.5.35 TXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetLocalName method. GetLocalName returns the local name of the underlying DOM node.

For element nodes, the local name is derived from the name that appears in the XML tag. For example, the local name for <xsi:Name First="John" Last="Doe"> is 'Name'.

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <Name First=John Last=Doe> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, the local name is nil (Delphi) or NULL (C++).

Note: If both GetPrefix (see page 7376) and GetLocalName return a value for a node, the node name (returned by GetNodeName (see page 7373)) is the combination of these two values.

See Also

GetNodeName (see page 7373)

GetNodeType (see page 7374)

GetPrefix (see page 7376)

GetNamespaceURI (see page 7373)

LocalName (see page 7521)

82.1.2.5.36 TXMLNode.GetNamespaceURI Method

Returns the URI for the namespace used in the qualified node name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetNamespaceURI method. GetNamespaceURI returns the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, GetNamespaceURI returns nil (Delphi) or NULL (C++).

The namespace URI is only available if it is explicitly assigned when the node is created. By default, TXMLNode (see page 7348) does not assign namespaces to the new nodes it creates.

See Also

GetNodeName (see page 7373)

GetNodeType (see page 7374)

GetPrefix (see page 7376)

GetLocalName (see page 7372)

CreateDOMNode (see page 7439)

NamespaceURI (see page 7521)

82.1.2.5.37 TXMLNode.GetNodeName Method

Returns the node name.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetNodeName method. GetNodeName returns the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'

ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If GetPrefix (see page 7376) and GetLocalName (see page 7372) both return values for a node, the node name is the combination of these two values.

See Also

GetNodeType (see page 7374)

GetPrefix (see page 7376)

GetLocalName (see page 7372)

82.1.2.5.38 TXMLNode.GetNodeObject Method

Returns a reference to this TXMLNode (see page 7348) object.

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

GetNodeObject returns Self (Delphi) or this (C++).

82.1.2.5.39 TXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetNodeType method. GetNodeType returns the type of the underlying DOM node that this TXMLNode (see page 7348) object represents.

See Also

GetIsTextElement (see page 7371)

82.1.2.5.40 TXMLNode.GetNodeValue Method

Returns the value of this node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetNodeValue method. GetNodeValue returns the value of the node. This value depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, GetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

GetChildValue (see page 7369)

GetText (see page 7378)

82.1.2.5.41 TXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetOwnerDocument method. GetOwnerDocument returns the value of the OwnerDocument (see page 7394) property.

See Also

OwnerDocument (see page 7394)

82.1.2.5.42 TXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetParentNode method. GetParentNode returns the interface of the node's parent node.

See Also

ParentNode (see page 7395)

82.1.2.5.43 TXMLNode.GetPrefix Method

Returns the namespace prefix of the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetPrefix method. GetPrefix returns the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

Note: If both GetPrefix and GetLocalName (see page 7372) return a value for a node, the node name (returned by GetNodeName (see page 7373)) is the combination of these two values.

See Also

GetLocalName (see page 7372)

GetNodeType (see page 7374)

GetNamespaceURI (see page 7373)

GetPrefixedName (see page 7377)

82.1.2.5.44 TXMLNode.GetPrefixedName Method

Returns a node name includes a namespace prefix for a specified namespace URI.

Pascal

```
function GetPrefixedName(const Name: DOMString; const NamespaceURI: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetPrefixedName(const DOMString Name, const DOMString NamespaceURI);
```

Description

GetPrefixedName adds a prefix to a local node name based on the specified namespace URI. If there is no corresponding namespace already declared or if the name is already prefixed, GetPrefixedName returns the original node name.

Name is the node name to which the namespace prefix should be added.

NamespaceURI is the URI for the namespace that should be identified by the namespace prefix.

Note: If the document containing this node does not include doAutoPrefix in its Options property, GetPrefixedName always returns the value of Name, without trying to generate or add a namespace prefix.

See Also

GetLocalName (see page 7372)

GetPrefix (see page 7376)

GetNamespaceURI (see page 7373)

Options (see page 7341)

82.1.2.5.45 TXMLNode.GetReadOnly Method

Indicates whether the node can be modified.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly returns true if the node can't be modified. Read (see page 3727)-only nodes raise an EXMLDocError (see page 7585) exception when an attempt is made to change their value, child nodes, or attributes.

GetReadOnly returns false if the node can be modified.

Note: Use the ReadOnly property on the IXMLNode (see page 7615) interface to specify whether a node is read-only.

See Also

SetReadOnly (see page 7388)

ReadOnly (see page 7642)

EXMLDocError (see page 7585)

82.1.2.5.46 TXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetText method. GetText returns the text of the node. For example, if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

GetText returns the string 'Understanding XML'.

GetText is intended for use when the GetIsTextElement (see page 7371) method returns true. If GetIsTextElement (see page 7371) returns false, GetText returns an empty string if the node has no children, otherwise it raises an exception.

See Also

GetIsTextElement (see page 7371)

GetNodeValue (see page 7375)

SetText (see page 7388)

82.1.2.5.47 TXMLNode.GetXML Method

Returns the XML that corresponds to the subtree rooted at this node.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected GetXML method. GetXML returns the XML that corresponds to this node and any child nodes it contains.

Note: GetXML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetText (see page 7378)

ChildNodes (see page 7391)

DOMVendor (see page 7337)

82.1.2.5.48 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	TXMLNode.HasAttribute (DOMString) (see page 7379)	Indicates whether the node has an attribute with a specified name.
	TXMLNode.HasAttribute (DOMString, DOMString) (see page 7379)	Indicates whether the node has an attribute with a specified name.

82.1.2.5.48.1 TXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected HasAttribute method.

Name is the name of the attribute about which you want to know.

NamespaceURI is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 7390)

GetAttribute (see page 7367)

GetAttributeNS (see page 7368)

82.1.2.5.49 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	TXMLNode.HasChildNode (DOMString) (see page 7379)	Indicates whether the node has a child node with the specified name.
	TXMLNode.HasChildNode (DOMString, DOMString) (see page 7379)	Indicates whether the node has a child node with the specified name.

82.1.2.5.49.1 TXMLNode.HasChildNode Method (DOMString)

Indicates whether the node has a child node with the specified name.

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
```

```
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

Applications can't call the protected HasChildNode method. It is used internally to determine when the underlying DOM node has a child with the specified name.

ChildTag is the local tag name of the child to find.

NamespaceURI identifies the namespace in which ChildTag is defined.

HasChildNode returns true if the underlying DOM node includes the specified child node. Otherwise, it returns false.

See Also

ChildNodes (see page 7391)

DOMNode (see page 7392)

82.1.2.5.50 TXMLNode.InternalAddChild Method

Provides the underlying implementation for the AddChild (see page 7356) method.

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

Applications can't call the protected InternalAddChild method. It is used internally to perform all the tasks common to the various overloads of the AddChild (see page 7356) method. InternalAddChild creates a new element node as the child of this node and returns its interface.

NodeClass specifies the implementation class to use for the new child node. It must be TXMLNode (see page 7348) or one of its descendants.

NodeName specifies the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

See Also

AddChild (see page 7356)

82.1.2.5.51 TXMLNode.NestingLevel Method

Returns the number of parents for this node in the node hierarchy.

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

Application's can't call the protected NestingLevel property. It is used internally when formatting the XML text that corresponds to this XML document.

NestingLevel returns the number of ancestors for this node in the node hierarchy.

See Also

ParentNode (see page 7395)

ChildNodes (see page 7391)

82.1.2.5.52 TXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected NextSibling method. NextSibling returns the node that follows this one in the parent node's ChildNodes (see page 7391) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 7395)

PreviousSibling (see page 7382)

ChildNodes (see page 7391)

82.1.2.5.53 TXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode (see page 7615) interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 7391)

AttributeNodes (see page 7390)

82.1.2.5.54 TXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected PreviousSibling method. PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 7391) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 7395)

NextSibling (see page 7381)

ChildNodes (see page 7391)

82.1.2.5.55 TXMLNode.RegisterChildNode Method

Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

Applications can't call the protected RegisterChildNode method. It is called by the RegisterChildNodes (see page 7383) method to register a TXMLNode (see page 7348) descendant so that it can be used as a child of this node. Each TXMLNode (see page 7348) descendant is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of a node.

RegisterChildNode adds an entry to the ChildNodeClasses (see page 7390) property for the specified node class and tag name.

TagName is the tag name of the child node that the class represents.

NodeClass is the TXMLNode (see page 7348) descendant that implements child nodes that have the specified tag name.

NamespaceURI identifies the namespace in which TagName is defined.

Note: RegisterChildNode raises an exception if this is a hosted node.

See Also

RegisterChildNodes (see page 7383)

ChildNodeClasses (see page 7390)

82.1.2.5.56 TXMLNode.RegisterChildNodes Method

Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

Applications can't call the protected RegisterChildNodes method. It is used in code that the XML Data Binding wizard generates to register the special TXMLNode (see page 7348) descendants the are created for nodes in the XML document. Each of these descendant classes is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of the node.

RegisterChildNodes adds an entry to the ChildNodeClasses (see page 7390) property for each specified node class and tag name.

TagNames is an array containing the names of the child nodes that the classes represent.

TagNames_Size is the index of the last name in TagNames (one less than the number of elements).

NodeClasses is an array containing the TXMLNode (see page 7348) descendants that implement child nodes that have the tag names in the corresponding entries in TagNames.

NodeClasses_Size is the index of the last class in NodeClasses (one less than the number of elements).

See Also

RegisterChildNode (see page 7382)

ChildNodeClasses (see page 7390)

82.1.2.5.57 TXMLNode.RemoveHostedNode Method

Removes a node from the HostedNodes (see page 7393) property list.

Pascal

```
procedure RemoveHostedNode(Node: TXMLNode);
```

C++

```
__fastcall RemoveHostedNode(TXMLNode Node);
```

Description

RemoveHostedNode is used internally to remove nodes from the HostedNodes (see page 7393) array.

Node is the node to remove from the array.

See Also

HostedNodes (see page 7393)

AddHostedNode (see page 7357)

82.1.2.5.58 TXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected Resync method. Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7391)

AttributeNodes (see page 7390)

Refresh (see page 7323)

82.1.2.5.59 TXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected SetAttribute method. SetAttribute sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

GetAttribute (see page 7367)

AttributeNodes (see page 7390)

SetAttributeNS (see page 7385)

82.1.2.5.60 TXMLNode.SetAttributeNodes Method

Sets the value of the AttributeNodes (see page 7390) property.

Pascal

```
procedure SetAttributeNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetAttributeNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the AttributeNodes (see page 7390) property to supply the interface that implements that property.

See Also

AttributeNodes (see page 7390)

82.1.2.5.61 TXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected SetAttributeNS method. SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 7384)

GetAttributeNS (see page 7368)

AttributeNodes (see page 7390)

82.1.2.5.62 TXMLNode.SetChildNodes Method

Sets the value of the ChildNodes (see page 7391) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 7391) property to supply the interface that implements that property.

See Also

ChildNodes (see page 7391)

82.1.2.5.63 TXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected SetChildValue method. SetChildValue sets the value of a specified child node to a specified value.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, SetChildValue raises an exception.

Value is the value to assign. It's interpretation depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, SetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, SetChildValue raises an exception.	

Note: Attributes are not considered child nodes of a node. To set the values of this node's attributes, use the SetAttribute (see page 7384) method instead.

See Also

ChildNodes (see page 7391)

ParentNode (see page 7395)

SetAttribute (see page 7384)

SetNodeValue (see page 7387)

82.1.2.5.64 TXMLNode.SetCollection Method

Sets the value of the Collection (see page 7391) property.

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

SetCollection assigns Value as the value of the Collection (see page 7391) property. This protected method is called when this node is added to the collection specified by the Value property.

Value is a TXMLNodeCollection (see page 7396) descendant that represents a repeating set of child nodes, of which this node is a member.

See Also

Collection (see page 7391)

82.1.2.5.65 TXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected SetNodeValue method. SetNodeValue sets the value of this node.

Value is the value to assign. It's meaning depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, SetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the node is any other node type, SetNodeValue raises an exception.	

See Also

SetChildValue (see page 7386)

SetText (see page 7388)

82.1.2.5.66 TXMLNode.SetParentNode Method

Sets the value of the ParentNode (see page 7395) property.

Pascal

```
procedure SetParentNode(const Value: TXMLNode); virtual;
```

C++

```
virtual __fastcall SetParentNode(const TXMLNode Value);
```

Description

Applications can't use this protected property. It is used internally when nodes are added to the ChildNodes (see page 7391) or AttributeNodes (see page 7390) property.

See Also

ParentNode (see page 7395)

ChildNodes (see page 7391)

AttributeNodes (see page 7390)

82.1.2.5.67 TXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly assigns the read-only status of the node.

Value is the new read-only status of the node. When Value is true, the node raises an EXMLDocError (see page 7585) exception when an attempt is made to change its value, child nodes, or attributes. When Value is false, the node can be modified.

See Also

GetReadOnly (see page 7377)

ReadOnly (see page 7642)

EXMLDocError (see page 7585)

82.1.2.5.68 TXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected SetText method. SetText replaces the text of the node.

SetText is intended for use when the GetIsTextElement (see page 7371) method returns true. If GetIsTextElement (see page 7371) returns false, SetText adds a new text value to a node that has no children, otherwise, it raises an exception.

See Also

GetIsTextElement (see page 7371)

SetNodeValue (see page 7387)

GetText (see page 7378)

82.1.2.5.69 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	TXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 7389)	Transforms the subtree rooted at this node, using a specified XSL interface.
	TXMLNode.TransformNode (IXMLNode, WideString) (see page 7389)	Transforms the subtree rooted at this node, using a specified XSL interface.

82.1.2.5.69.1 TXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Use the TXMLNode (see page 7348) object's IXMLNode (see page 7615) interface to call the protected TransformNode method.

stylesheet is a node in an XML document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString (see page 3476), TransformNode returns a string of XML. If output is the interface to an XML document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetXML (see page 7378)

82.1.2.6 TXMLNode Properties

The properties of the TXMLNode class are listed here.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

82.1.2.6.1 TXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

TXMLNode (see page 7348) uses AttributeNodes to keep track of its attributes. AttributeNodes is an IXMLNodeList (see page 7649) interface, which can be used to access individual attribute nodes or to add attributes to or delete them from this node.

Applications can't use this protected property to access the attributes of a node. Instead, they use the AttributeNodes property of the IXMLNode (see page 7615) interface for this node.

Attributes modify a node in an XML document. For example, given the node

```
<Movie genre=scifi title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 7391) property.

See Also

IXMLNode (see page 7615)

ChildNodes (see page 7391)

AttributeNodes (see page 7634)

82.1.2.6.2 TXMLNode.ChildNodeClasses Property

Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

ChildNodeClasses is used internally when creating child nodes for this node. The XML Data Binding wizard defines TXMLNode (see page 7348) descendants for each node in the XML document that represents a node that has (or can have) children. ChildNodeClasses lists only those TXMLNode (see page 7348) descendants that can represent child nodes of this node. These classes become the types of the corresponding properties on the TXMLNode (see page 7348) descendant that represents this node.

Classes are added to ChildNodeClasses by the RegisterChildNode (see page 7382) method.

See Also

RegisterChildNode (see page 7382)

RegisterChildNodes (see page 7383)

ChildNodes (see page 7391)

CreateChildNode (see page 7362)

82.1.2.6.3 TXMLNode.ChildNodes Property

Lists the child nodes of this node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

TXMLNode (see page 7348) uses ChildNodes to keep track of the subnodes of a node. ChildNodes is an IXMLNodeList (see page 7649) interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Applications can't use this protected property to access the child nodes of a node. Instead, they use the ChildNodes property of the IXMLNode (see page 7615) interface for this node.

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes property instead.

See Also

IXMLNode (see page 7615)

ChildNodes (see page 7635)

ParentNode (see page 7395)

82.1.2.6.4 TXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: TXMLNodeCollection;
```

C++

```
__property TXMLNodeCollection Collection;
```

Description

Collection indicates the collection that contains this node if it represents one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML document as this node's ParentNode (see page 7395) property, but it is specialized to work with this node and any other child nodes of the same type.

The node identified by Collection is one of the collections listed by the parent node's HostedNodes (see page 7393) property, and it lists ParentNode (see page 7395) as the value of its HostNode (see page 7393) property.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 7395) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

Applications can't use this protected property to access the Collection that contains this node. Instead, they use the Collection property of the IXMLNode (see page 7615) interface for this node.

See Also

ParentNode (see page 7395)

HostedNodes (see page 7393)

HostNode (see page 7393)

Collection (see page 7636)

82.1.2.6.5 TXMLNode.DOMNode Property

Provides access to the IDOMNode (see page 7509) interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode (see page 7509) is the interface for this node provided by the DOM implementation that parsed the XML document. It can be accessed through the DOMNode property of the node's IXMLNode (see page 7615) interface.

Typically, applications that contain TXMLNode (see page 7348) descendants use the IXMLNode (see page 7615) interface to work with a node rather than the IDOMNode (see page 7509) interface. IDOMNode (see page 7509) is used more typically in applications that work directly with the DOM interfaces.

The value of DOMNode is set to the IDOMNode (see page 7509) interface that is provided to the TXMLNode (see page 7348) constructor.

See Also

DOMNode (see page 7637)

Create (see page 7352)

82.1.2.6.6 TXMLNode.HostedNodes Property

Lists the nodes that share this node's lists of child and attribute nodes.

Pascal

```
property HostedNodes: TXMLNodeArray;
```

C++

```
__property TXMLNodeArray HostedNodes;
```

Description

TXMLNode (see page 7348) uses HostedNodes to keep track of the nodes it uses to represent this same node, restricted to a specialized purpose. For example, any node that represents a subset of this node's child nodes that are all of the same type is a hosted node.

Each object in the HostedNodes list is a representation of a particular aspect of this node. Each hosted node acts as a sibling to this TXMLNode (see page 7348) object but does not represent a separate element in the XML document. Rather, it represents a different view of this element.

For example, consider the following XML fragment:

```
<students>
  <fulltime student>Joe Smith</fulltime student>
  <fulltime student>Jane Doe</fulltime student>
  <auditor>Bill Jones</auditor>
</students>
```

The TXMLNode (see page 7348) that represents students would define two hosted nodes for collections of repeating child elements, one for the fulltime students, and one for the auditors. The two collections share the same underlying DOM node as students, but each represents only those child elements of a particular type.

See Also

CreateCollection (see page 7363)

ChildNodes (see page 7391)

TXMLNodeCollection (see page 7396)

Collection (see page 7391)

HostNode (see page 7393)

82.1.2.6.7 TXMLNode.HostNode Property

Indicates the main object for the node to which this node corresponds when it represents a specialized view.

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

If this node represents a hosted node, HostNode is the main node for which this node represents a subset or specialized view. For example, if this node represents a set of repeating child elements, HostNode is the node that uses this node to represent its repeating child elements.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
  <auditor name="Susan Brown"></auditor>
</students>
```

The <students> node spawns two hosted nodes to represent the collections of repeating child nodes: one for all of the child nodes of type <student>, and one for all of the child nodes of type <auditor>. If this node represents one of those hosted nodes, then HostNode is the <students> node to which this node belongs. The host node lists this node in its HostedNodes (see page 7393) property.

Applications can't use this protected property to access the host node that created this node. Instead, they use the HostNode property of the IXMLNode (see page 7615) interface for this node.

See Also

Collection (see page 7391)

HostedNodes (see page 7393)

TXMLNodeCollection (see page 7396)

82.1.2.6.8 TXMLNode.OwnerDocument Property

Provides access to the document that contains this node.

Pascal

```
property OwnerDocument: TXMLDocument;
```

C++

```
__property TXMLDocument OwnerDocument;
```

Description

Applications can't use this protected property to access the document that contains this. Instead, they use the OwnerDocument property of the IXMLNode (see page 7615) interface for this node.

The value of OwnerDocument is set to the TXMLDocument (see page 7299) that is provided to the TXMLNode (see page 7348) constructor.

See Also

Create (see page 7352)

OwnerDocument (see page 7641)

82.1.2.6.9 TXMLNode.ParentNode Property

Identifies the object that represents this node's parent node.

Pascal

```
property ParentNode: TXMLNode;
```

C++

```
__property TXMLNode ParentNode;
```

Description

TXMLNode (see page 7348) uses ParentNode to access the implementation of the node's parent, if it exists. Some nodes (for example attribute nodes) do not have parents.

Applications can't use this protected property to access the parent node of a node. Instead, they use the ParentNode property of the IXMLNode (see page 7615) interface for this node.

See Also

ChildNodes (see page 7391)

ParentNode (see page 7641)

82.1.2.7 TXMLNode Events

The events of the TXMLNode class are listed here.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

82.1.2.7.1 TXMLNode.OnHostAttrNotify Event

Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).

Pascal

```
property OnHostAttrNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostAttrNotify;
```

Description

Use the OnHostAttrNotify event to update a TXMLNode (see page 7348) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its attribute list.

The AttributeListNotify (see page 7358) method of a host node generates an OnHostAttrNotify event on all of its hosted nodes. This allows the hosted nodes (for example, the nodes that represent a subset of the child nodes that are all the same type) to update their own representation of node attributes so that they reflect the attributes of the host node.

See Also

OnHostChildNotify (see page 7396)

AttributeListNotify (see page 7358)

HostNode (see page 7393)

HostedNodes (see page 7393)

AttributeNodes (see page 7390)

82.1.2.7.2 TXMLNode.OnHostChildNotify Event

Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Pascal

```
property OnHostChildNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostChildNotify;
```

Description

Use the OnHostChildNotify event to update a TXMLNode (see page 7348) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its list of child nodes. For example, TXMLNodeCollection (see page 7396) assigns its UpdateCollectionList method as an OnHostChildNotify event handler so that its list of repeating child elements reflects any changes to the host node's list of child nodes.

The ChildListNotify (see page 7360) method of a host node generates an OnHostChildNotify event on all of its hosted nodes.

See Also

OnHostAttrNotify (see page 7395)

ChildListNotify (see page 7360)

HostNode (see page 7393)

HostedNodes (see page 7393)

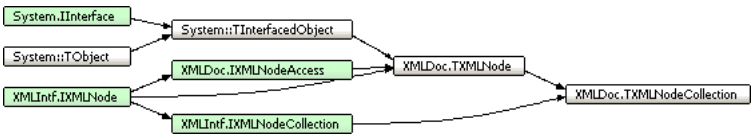
ChildNodes (see page 7391)

UpdateCollectionList (see page 7409)

82.1.3 TXMLNodeCollection Class

TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.

Class Hierarchy



Pascal

```
TXMLNodeCollection = class(TXMLNode, IXMLNodeCollection);
```

C++

```
class TXMLNodeCollection : public TXMLNode, public IXMLNodeCollection;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNodeCollection as a base class for the classes it generates to represent nodes that have repeating elements for child nodes. For example, if the XML document includes the following:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

then the XML Data Binding Wizard generates 3 nodes to represent the <students> node: a TXMLNode (see page 7348) descendant to represent the entire <students> node, and two TXMLNodeCollection descendants to represent the repeating elements: one for the child nodes with the tag <student>, and one for the child nodes with the tag <auditor>.

As with any of the classes the Data Binding Wizard generates to represent nodes in an XML document, child nodes and node attributes appear as properties of the TXMLNodeCollection descendant.

TXMLNodeCollection implements the IXMLNodeCollection (see page 7644) interface. Each TXMLNodeCollection descendant implements a class-specific interface that descends from IXMLNodeCollection (see page 7644). Typically, applications do not work directly with TXMLNodeCollection. Instead, they use the IXMLNodeCollection (see page 7644) interface or the IXMLNodeCollection (see page 7644) descendant that is implemented by a TXMLNodeCollection descendant.

See Also

TXMLDocument (see page 7299)

TXMLNode (see page 7348)

IXMLNodeCollection (see page 7644)

82.1.3.1 TXMLNodeCollection Members

The following tables list the members exposed by TXMLNodeCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7395)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7393).
	OnHostChildNotify (see page 7396)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7393).

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7355)	Increments the reference count for this node.

	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	_Release (see page 7356)	Decrements the reference count for this node.
	AddChild (see page 7356)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7357)	Adds a node to the HostedNodes (see page 7393) property list.
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7358)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7358)	Raises an EXMLDocError (see page 7585) exception if this is a hosted node.
	CheckReadOnly (see page 7359)	Raises an EXMLDocError (see page 7585) exception if this is a read-only node.
	CheckTextNode (see page 7359)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	ClearDocumentRef (see page 7360)	Sets the value of OwnerDocument (see page 7394) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7361)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7361)	Creates the object that implements the AttributeNodes (see page 7390) property.
V	CreateAttributeNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for an attribute of this node.
	CreateChildList (see page 7362)	Creates the object that implements the ChildNodes (see page 7391) property.
V	CreateChildNode (see page 7362)	Creates a new TXMLNode (see page 7348) object for a child node of this node.
	CreateCollection (see page 7363)	Creates an instance of a TXMLNodeCollection (see page 7396) descendant and returns it.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	DeclareNamespace (see page 7364)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7364)	Returns the IDOMElement (see page 7494) interface for this node if it exists.
V	DoNodeChange (see page 7365)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7365)	Returns the IXMLNode (see page 7615) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7366)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7366)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7367)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7367)	Returns the value of the AttributeNodes (see page 7390) property.
	GetAttributeNS (see page 7368)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7368)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7369)	Returns the value of the ChildNodes (see page 7391) property.
	GetChildValue (see page 7369)	Returns the value of a specified child node.
	GetCollection (see page 7370)	Returns the value of the Collection (see page 7391) property.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
	GetDOMNode (see page 7370)	Returns the value of the DOMNode (see page 7392) property.
	GetHasChildNodes (see page 7370)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7371)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7371)	Returns the value of the HostNode (see page 7393) property.
	GetIsTextElement (see page 7371)	Indicates whether the node has a single text value.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetLocalName (see page 7372)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7373)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7373)	Returns the node name.
	GetNodeObject (see page 7374)	Returns a reference to this TXMLNode (see page 7348) object.
	GetNodeType (see page 7374)	Returns the type of the node.
	GetNodeValue (see page 7375)	Returns the value of this node.

	GetOwnerDocument (see page 7375)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7376)	Returns the interface for the node's parent node.
	GetPrefix (see page 7376)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7377)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7377)	Indicates whether the node can be modified.
	GetText (see page 7378)	Returns the text value of the node.
	GetXML (see page 7378)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7378)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7379)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	InternalAddChild (see page 7380)	Provides the underlying implementation for the AddChild (see page 7356) method.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7380)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7381)	Returns the next child of this node's parent.
	Normalize (see page 7381)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7382)	Returns the previous child of this node's parent.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7382)	Registers a descendant of TXMLNode (see page 7348) that can represent the children of this node.
	RegisterChildNodes (see page 7383)	Registers a set of TXMLNode (see page 7348) descendants that can represent the children of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7383)	Removes a node from the HostedNodes (see page 7393) property list.
	Resync (see page 7384)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7384)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7384)	Sets the value of the AttributeNodes (see page 7390) property.
	SetAttributeNS (see page 7385)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	SetChildValue (see page 7386)	Sets the value of a specified child node.
	SetCollection (see page 7386)	Sets the value of the Collection (see page 7391) property.
	SetNodeValue (see page 7387)	Sets the value of this node.
V	SetParentNode (see page 7387)	Sets the value of the ParentNode (see page 7395) property.
	SetReadOnly (see page 7388)	Specifies whether the node can be modified.
	SetText (see page 7388)	Sets the text value of the node.
	TransformNode (see page 7389)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7390)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7390)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7391)	Lists the child nodes of this node.
	Collection (see page 7391)	Indicates the collection in which this node appears.
R	Count (see page 7409)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7392)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HostedNodes (see page 7393)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7393)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.

	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7394)	Provides access to the document that contains this node.
R	ParentNode (see page 7395)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7352)	Instantiates a Create object.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7353)	Creates a TXMLNode (see page 7348) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 7353)	Frees the memory associated with the TXMLNode (see page 7348) object.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.

	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	ToString (see page 3391)	Returns a string containing the class name.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
R	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

82.1.3.2 TXMLNodeCollection Methods

The methods of the TXMLNodeCollection class are listed here.

Protected Methods

	Name	Description
V	AddItem (see page 7403)	Adds a new repetition of the repeating child element to this node.
V	ChildListNotify (see page 7404)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7404)	Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.
	CreateItemList (see page 7405)	Creates the object that implements the List (see page 7411) property.
	Delete (see page 7405)	Removes one of the repeating child nodes from this node.
	GetCount (see page 7406)	Returns the value of the Count (see page 7409) property.
V	GetList (see page 7406)	Returns the value of the List (see page 7411) property.
	GetNode (see page 7406)	Returns the interface for one of the repeating child nodes.
	InsertInCollection (see page 7407)	Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.
	IsCollectionItem (see page 7407)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	Remove (see page 7408)	Removes one of the repeating child nodes from this node.
V	SetChildNodes (see page 7408)	Sets the value of the ChildNodes (see page 7391) property.
	UpdateCollectionList (see page 7409)	Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Public Methods

	Name	Description
V	AfterConstruction (see page 7403)	Initializes the node collection object once it is fully created.

82.1.3.2.1 TXMLNodeCollection.AddItem Method

Adds a new repetition of the repeating child element to this node.

Pascal

```
function AddItem(Index: Integer): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode AddItem(int Index);
```

Description

Applications can't call the protected AddItem method. It is used internally to create a new child node for this node that is an instance of the repeating element. The new child node is given the first tag name listed by the ItemTag (see page 7411) property.

Index indicates the position of the new child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

Note: Index refers to the list of repeating elements (the List (see page 7411) property), not the list of all children (the ChildNodes (see page 7391) property).

AddItem returns the interface for the newly created child node.

See Also

AddChild (see page 7356)

InsertInCollection (see page 7407)

itemtag (see page 7411)

List (see page 7411)

ChildNodes (see page 7391)

82.1.3.2.2 TXMLNodeCollection.AfterConstruction Method

Initializes the node collection object once it is fully created.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

AfterConstruction sets the UpdateCollectionList (see page 7409) method as the event handler for the collection's OnHostChildNotify (see page 7396) event. This allows the collection to update itself when the list of child nodes (for which this collection manages a subset) changes.

See Also

OnHostChildNotify (see page 7396)

UpdateCollectionList (see page 7409)

82.1.3.2.3 TXMLNodeCollection.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); override;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 7391) property calls ChildListNotify when child nodes are inserted, removed, or need to be created. ChildListNotify calls the inherited method to generate a BeforeNodeChange or AfterNodeChange event and to ensure that the DOM implementation for this node reflects the change. After calling the inherited method, ChildListNotify updates the List (see page 7411) property so that it stays in sync with the ChildNodes (see page 7391) property.

Operation indicates what type of change is about to occur.

Node is the interface for the child node that is added or removed. If Operation is nlCreateNode, Node returns a newly created child node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name or value of the new node to create.

BeforeOperation indicates whether the call occurs before or after the change.

See Also

ChildNodes (see page 7391)

AddChild (see page 7356)

List (see page 7411)

BeforeNodeChange (see page 7346)

AfterNodeChange (see page 7345)

82.1.3.2.4 TXMLNodeCollection.Clear Method

Removes all child nodes in the List (see page 7411) property from the ChildNodes (see page 7391) and List (see page 7411) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the IXMLNodeCollection (see page 7644) interface for this node to call the protected Clear method. Clear deletes every child node from the collection (all the nodes in the List (see page 7411) property), along with the corresponding nodes in the ChildNodes (see page 7391) property.

See Also

List (see page 7411)

Delete (see page 7405)

Remove (see page 7408)

ChildNodes (see page 7391)

Clear (see page 7646)

82.1.3.2.5 TXMLNodeCollection.CreateItemList Method

Creates the object that implements the List (see page 7411) property.

Pascal

```
procedure CreateItemList(CheckFirst: Boolean = False);
```

C++

```
__fastcall CreateItemList(Boolean CheckFirst = False);
```

Description

Applications can't call the protected CreateItemList method. It is used internally when TXMLNodeCollection (see page 7396) needs to create a separate object to implement the List (see page 7411) property. A separate implementation object is only needed if this node contains child nodes that are not in the collection. Otherwise, TXMLNodeCollection (see page 7396) can simply use the ChildNodes (see page 7391) property.

CheckFirst specifies whether CreateItemList should check whether the ChildNodes (see page 7391) property currently includes any nodes that are not in the collection. When CheckFirst is true, CreateItemList first checks the ChildNodes (see page 7391) property, and if all nodes are in the collection, it does not create a separate list to store the nodes of the collection. When CheckFirst is false (or when the ChildNodes (see page 7391) property includes nodes that are not in the collection), CreateItemList instantiates a list to hold the nodes in the collection, and adds to it any nodes listed in ChildNodes (see page 7391) that belong in the collection.

See Also

List (see page 7411)

ChildNodes (see page 7391)

82.1.3.2.6 TXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Use the TXMLNodeCollection (see page 7396) object's IXMLNodeCollection (see page 7644) interface to call the protected Delete method. Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the list of repeating child nodes (The List (see page 7411) property, not the ChildNodes (see page 7391) property). The index 0 represents the first repetition, 1 represents the second

repetition, and so on.

See Also

Remove (see page 7408)

Delete (see page 7646)

List (see page 7411)

ChildNodes (see page 7391)

82.1.3.2.7 TXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 7409) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 7409) property. It returns the number of nodes in the List (see page 7411) property.

See Also

Count (see page 7409)

List (see page 7411)

82.1.3.2.8 TXMLNodeCollection.GetList Method

Returns the value of the List (see page 7411) property.

Pascal

```
function GetList: IXMLNodeList; virtual;
```

C++

```
virtual __fastcall IXMLNodeList GetList();
```

Description

GetList is the protected read implementation of the List (see page 7411) property.

See Also

List (see page 7411)

82.1.3.2.9 TXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

Use the TXMLNodeCollection (see page 7396) object's IXMLNodeCollection (see page 7644) interface to call the protected GetNode method. GetNode returns an interface for working with one of the repeating child nodes.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on. Note that Index is the index of the node in the List (see page 7411) property, not in the index in the ChildNodes (see page 7391) property.

See Also

Nodes (see page 7649)

GetCount (see page 7406)

GetNode (see page 7647)

ChildNodes (see page 7391)

82.1.3.2.10 TXMLNodeCollection.InsertInCollection Method

Copies a child node from the ChildNodes (see page 7391) property to the List (see page 7411) property.

Pascal

```
procedure InsertInCollection(Node: IXMLNode; Index: Integer);
```

C++

```
__fastcall InsertInCollection(IXMLNode Node, int Index);
```

Description

Applications can't call this protected method. It is called automatically when nodes are added to the ChildNodes (see page 7391) property list.

Node is the node that was just added to the ChildNodes (see page 7391) property.

Index is the index of Node in the ChildNodes (see page 7391) property.

InsertInCollection first checks whether Node belongs in the collection. If so, it adds it to the List (see page 7411) property in the position that corresponds to Index (that is, in the index of Node when you ignore all the nodes in ChildNodes (see page 7391) that do not belong in the collection.)

See Also

ChildNodes (see page 7391)

List (see page 7411)

82.1.3.2.11 TXMLNodeCollection.IsCollectionItem Method

Indicates whether a specified node represents the type of node that appears as a repeating child of this node.

Pascal

```
function IsCollectionItem(const Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean IsCollectionItem(const IXMLNode Node);
```

Description

Applications can't call the protected IsCollectionItem method. TXMLNodeCollection (see page 7396) uses this method

internally to distinguish nodes that represent instances of the repeating child node type.

IsCollectionItem checks whether Node implements the interface specified by the ItemInterface (see page 7410) property . If so, it returns true. Otherwise, it returns false.

See Also

List (see page 7411)

ItemInterface (see page 7410)

82.1.3.2.12 TXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const AItem: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode AItem);
```

Description

Use the TXMLNodeCollection (see page 7396) object's IXMLNodeCollection (see page 7644) interface to call the protected Remove method. Remove removes a specified child node from this node.

AItem is the interface of the child to remove. It must represent on of the repeating child nodes.

Remove deletes the child node from both the List (see page 7411) property and from the ChildNodes (see page 7391) property. It returns the index that the child node held in the List (see page 7411) property.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 7391) property instead.

See Also

Delete (see page 7405)

List (see page 7411)

ChildNodes (see page 7391)

82.1.3.2.13 TXMLNodeCollection.SetChildNodes Method

Sets the value of the ChildNodes (see page 7391) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); override;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 7391) property to supply the interface that implements that property.

TXMLNodeCollection (see page 7396) overrides the inherited method to update the List (see page 7411) property as well.

See Also

ChildNodes (see page 7391)

List (see page 7411)

82.1.3.2.14 TXMLNodeCollection.UpdateCollectionList Method

Updates the List (see page 7411) property to reflect changes to the set of child nodes.

Pascal

```
procedure UpdateCollectionList(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall UpdateCollectionList(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. This method is called automatically when the ChildNodes (see page 7391) property changes so that the collection can update the List (see page 7411) property to reflect the repeating elements in the current set of child nodes.

Operation indicates whether the change to ChildNodes (see page 7391) is the addition of a new node (nlInsert) or the deletion of a child node (nlRemove).

Node is the interface for a child node that is added or removed.

IndexOrName is the index or tag name of the node that is added or removed.

BeforeOperation indicates whether this method is called before ChildNodes (see page 7391) has changed or after the change has occurred.

See Also

List (see page 7411)

ChildListNotify (see page 7404)

ChildNodes (see page 7391)

OnHostChildNotify (see page 7396)

82.1.3.3 TXMLNodeCollection Properties

The properties of the TXMLNodeCollection class are listed here.

Protected Properties

	Name	Description
R	Count (see page 7409)	Indicates the number of repeating child nodes.
	ItemInterface (see page 7410)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7410)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 7411)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 7411)	Lists the interfaces for the children that are repeated elements.

82.1.3.3.1 TXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use the IXMLNodeCollection (see page 7644) interface to read this protected property. Count is the number of child nodes for this node that are repetitions of the repeating child node type. That is, Count is the number of nodes in the List (see page 7411) property.

Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in using the GetNode (see page 7406) method).

Count is changed when you add child nodes using the AddChild (see page 7356) method or delete them using the Delete (see page 7405) or Remove (see page 7408) method. You can access the individual repeating child nodes using the GetNode (see page 7406) method.

See Also

GetNode (see page 7406)

AddChild (see page 7618)

Remove (see page 7408)

Delete (see page 7405)

List (see page 7411)

82.1.3.3.2 TXMLNodeCollection.ItemInterface Property

Indicates the interface type for the repeating child nodes.

Pascal

```
property ItemInterface: TGuid;
```

C++

```
__property TGuid ItemInterface;
```

Description

Applications can't use this protected property. It is used internally to identify the interface for all child nodes that represent the repeating child element.

ItemInterface is the GUID for the default interface of the class that implements the nodes in the collection.

See Also

IXMLNode (see page 7615)

IsCollectionItem (see page 7407)

List (see page 7411)

itemtag (see page 7411)

itemns (see page 7410)

82.1.3.3.3 TXMLNodeCollection.ItemNS Property

Pascal

```
property ItemNS: DOMString;
```

C++

```
__property DOMString ItemNS;
```

Description

This is ItemNS, a member of class TXMLNodeCollection.

82.1.3.3.4 TXMLNodeCollection.ItemTag Property

Pascal

```
property ItemTag: DOMString;
```

C++

```
__property DOMString ItemTag;
```

Description

This is ItemTag, a member of class TXMLNodeCollection.

82.1.3.3.5 TXMLNodeCollection.List Property

Lists the interfaces for the children that are repeated elements.

Pascal

```
property List: IXMLNodeList;
```

C++

```
__property IXMLNodeList List;
```

Description

Applications can't use the protected List property. TXMLNodeCollection (see page 7396) uses this property internally to identify those child nodes that are repeated elements. Because TXMLNodeCollection (see page 7396) can correspond to a node in the XML document that has additional children that are not instances of the repeating node type, List is kept separate from the ChildNodes (see page 7391) property, which lists all child nodes.

Applications that need to access the child nodes in this list should use the Nodes property of the IXMLNodeCollection (see page 7644) interface for this node or the Items property of an IXMLNodeCollection (see page 7644) descendant that is generated by the XML Data Binding wizard.

See Also

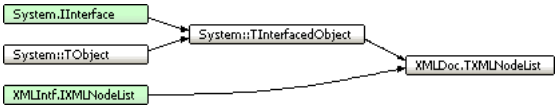
Nodes (see page 7649)

ChildNodes (see page 7391)

82.1.4 TXMLNodeList Class

TXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy



Pascal

```
TXMLNodeList = class(TInterfacedObject, IXMLNodeList);
```

C++

```
class TXMLNodeList : public TInterfacedObject, public IXMLNodeList;
```

File

XMLDoc

Description

TXMLNodeList is used to represent a set of related nodes (IXMLNode (see page 7615) interfaces) in an XML document. For example, TXMLNodeList is used to represent all of the children of a node, or all of the attributes of a node, or all of the nodes in a collection of repeating nodes.

TXMLNodeList can be used to add or delete nodes from the collection, or to access specific nodes.

Typically, applications do not use TXMLNodeList directly. Instead, they access it through the IXMLNodeList (see page 7649) interface.

See Also

ChildNodes

AttributeNodes

TXMLNodeCollection (see page 7396)

IXMLNodeList (see page 7649)

IXMLNode (see page 7615)

TObjectList

TInterfaceList

82.1.4.1 TXMLNodeList Members

The following tables list the members exposed by TXMLNodeList.

Protected Data Members

	Name	Description
	FRefCount (see page 3356)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
R	NotificationProc (see page 7428)	Specifies a callback that TXMLNodeList (see page 7411) calls when nodes are added to or removed from the list, or when new nodes need to be created

Protected Methods

	Name	Description
	_AddRef (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_AddRef method.
	_Release (see page 3357)	Implements the IInterface (see page 3503) System::TInterfacedObject::_Release method.
	Add (see page 7416)	Adds a new node to the end of the list.
	BeginUpdate (see page 7416)	Increments the value of UpdateCount (see page 7427).
	Clear (see page 7417)	Removes all nodes from the list.
	Delete (see page 7417)	This is the overview for the Delete method overload.

	DoNotify (see page 7418)	Executes the callback specified by NotificationProc (see page 7428).
	EndUpdate (see page 7419)	Decrements the value of UpdateCount (see page 7427).
	FindNode (see page 7419)	This is the overview for the FindNode method overload.
	FindSibling (see page 7420)	Returns a node that appears a specified amount before or after another node.
	First (see page 7420)	Returns the first node in the list.
	Get (see page 7421)	Returns a specified node in the list.
	GetCount (see page 7421)	Returns the number of nodes in the list.
	GetNode (see page 7422)	Returns a specified node from the list.
	GetUpdateCount (see page 7422)	Returns the value of the UpdateCount (see page 7427) property.
	IndexOf (see page 7422)	This is the overview for the IndexOf method overload.
	Insert (see page 7423)	Inserts a new node into a specified position in the list.
	InternalInsert (see page 7424)	Provides the underlying implementation of the Insert (see page 7423) method
	Last (see page 7424)	Returns the last node in the list.
	QueryInterface (see page 3359)	Implements the IInterface (see page 3503) System::TInterfacedObject::QueryInterface method.
	Remove (see page 7424)	Removes a specified node from the list.
	ReplaceNode (see page 7425)	Replaces a node in the list with another node.

Protected Properties

	Name	Description
R	Count (see page 7426)	Indicates the number of nodes in the collection.
R	DefaultNamespaceURI (see page 7426)	Identifies the default Namespace for the nodes in this collection.
R	List (see page 7426)	Stores the interfaces for every node in the list.
R	Owner (see page 7427)	Specifies the node that uses this TXMLNodeList (see page 7411) object to manage a set of its child or attribute nodes.
R	UpdateCount (see page 7427)	Indicates the number of times an application has called BeginUpdate (see page 7416) without a matching call to EndUpdate (see page 7419).

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3358)	Responds after the last constructor executes.
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7414)	Instantiates a TXMLNodeList (see page 7411) object.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Delete (see page 7652)	This is the overview for the Delete method overload.
V	Destroy (see page 7415)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	FindNode (see page 7654)	This is the overview for the FindNode method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.

	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 7657)	This is the overview for the IndexOf method overload.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3358)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3391)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Nodes (see page 7660)	Provides access to the nodes in the list.
	RefCount (see page 3360)	Indicates the number of interface pointers currently dependent upon the object.

82.1.4.2 TXMLNodeList.Create Constructor

Instantiates a TXMLNodeList (see page 7411) object.

Pascal

```
constructor Create(Owner: TXMLNode; const DefaultNamespaceURI: DOMString; NotificationProc:
TNodeListNotification);
```

C++

```
__fastcall TXMLNodeList(TXMLNode Owner, const DOMString DefaultNamespaceURI,
TNodeListNotification NotificationProc);
```

Description

Createinstantiates a new TXMLNodeList (see page 7411) object. Typically, TXMLNodeList (see page 7411) is created by another object that represents a node. The TXMLNodeList (see page 7411) represents a set of nodes that are of interest to the other object that creates it.

AOwner is the node that creates TXMLNodeList (see page 7411) to represent a set of its child nodes or attributes.

DefaultNamespaceURI (see page 7426) identifies the namespace that is used by default for nodes added to this list when no namespace is explicitly specified.

NotificationProc (see page 7428) is a callback that TXMLNodeList (see page 7411) calls when nodes are added to or removed from the list, or when new nodes need to be created.

See Also

DefaultNamespaceURI (see page 7426)

Owner (see page 7427)

NotificationProc (see page 7428)

82.1.4.3 TXMLNodeList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNodeList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

82.1.4.4 TXMLNodeList Methods

The methods of the TXMLNodeList class are listed here.

Protected Methods

	Name	Description
	Add (see page 7416)	Adds a new node to the end of the list.
	BeginUpdate (see page 7416)	Increments the value of UpdateCount (see page 7427).
	Clear (see page 7417)	Removes all nodes from the list.
	Delete (see page 7417)	This is the overview for the Delete method overload.
	DoNotify (see page 7418)	Executes the callback specified by NotificationProc (see page 7428).
	EndUpdate (see page 7419)	Decrements the value of UpdateCount (see page 7427).
	FindNode (see page 7419)	This is the overview for the FindNode method overload.
	FindSibling (see page 7420)	Returns a node that appears a specified amount before or after another node.
	First (see page 7420)	Returns the first node in the list.
	Get (see page 7421)	Returns a specified node in the list.
	GetCount (see page 7421)	Returns the number of nodes in the list.
	GetNode (see page 7422)	Returns a specified node from the list.
	GetUpdateCount (see page 7422)	Returns the value of the UpdateCount (see page 7427) property.
	IndexOf (see page 7422)	This is the overview for the IndexOf method overload.
	Insert (see page 7423)	Inserts a new node into a specified position in the list.

	InternalInsert (see page 7424)	Provides the underlying implementation of the Insert (see page 7423) method
	Last (see page 7424)	Returns the last node in the list.
	Remove (see page 7424)	Removes a specified node from the list.
	ReplaceNode (see page 7425)	Replaces a node in the list with another node.

82.1.4.4.1 TXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 7417)

Remove (see page 7424)

IndexOf (see page 7423)

Insert (see page 7423)

List (see page 7426)

82.1.4.4.2 TXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 7427).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 7427) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 7419) to decrement the UpdateCount (see page 7427) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 7419), so that the UpdateCount (see page 7427) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7427)

EndUpdate (see page 7419)

82.1.4.4.3 TXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 7416) and EndUpdate (see page 7419) methods. NotificationProc (see page 7428) is called for every node in the list as it is removed.

See Also

Delete (see page 7417)

Remove (see page 7424)

List (see page 7426)

NotificationProc

82.1.4.4.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	TXMLNodeList.Delete (DOMString) (see page 7417)	Removes a specified node from the list.
	TXMLNodeList.Delete (DOMString, DOMString) (see page 7417)	Removes a specified node from the list.
	TXMLNodeList.Delete (Integer) (see page 7417)	Removes a specified node from the list.

82.1.4.4.4.1 TXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name. If the node is identified by Name, but no NamespaceURI parameter is provided, Delete uses DefaultNamespaceURI (see page 7426).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 7426) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 7424)

Clear (see page 7417)

IndexOf (see page 7423)

LocalName (see page 7638)

DefaultNamespaceURI (see page 7426)

NamespaceURI (see page 7639)

82.1.4.4.5 TXMLNodeList.DoNotify Method

Executes the callback specified by NotificationProc (see page 7428).

Pascal

```
function DoNotify(Operation: TNodeListOperation; const Node: IXMLNode; const IndexOrName:
OleVariant; BeforeOperation: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode DoNotify(TNodeListOperation Operation, const IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

DoNotify executes the callback that is assigned to NotificationProc (see page 7428) when the list adds, deletes, or needs to create nodes in the list. The parameters are supplied as parameters to the callback, and DoNotify returns the node that was added, deleted, or created.

Operation indicates what change is about to occur or has just occurred.

Node is the node that is being added or removed from the list, or that acts as a parent if the callback must create a node (when Operation is nlCreateNode).

IndexOrName is the name of the node to be created when Operation is nlCreateNode. Otherwise, it is the name or index of the node being deleted or added.

BeforeOperation indicates whether the call occurs before the indicated operation or after. When nodes are inserted and removed, a call is made first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

NotificationProc (see page 7428)

82.1.4.4.6 TXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 7427).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 7427) property, which was incremented by a previous call to BeginUpdate (see page 7416).

Every call to BeginUpdate (see page 7416) should be matched by a call to EndUpdate, so that the UpdateCount (see page 7427) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7427)

BeginUpdate (see page 7416)

82.1.4.4.7 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	TXMLNodeList.FindNode (DOMString) (see page 7419)	Returns a specified node from the list.
	TXMLNodeList.FindNode (DOMString, DOMString) (see page 7419)	Returns a specified node from the list.
	TXMLNodeList.FindNode (TGuid) (see page 7419)	Returns a specified node from the list.

82.1.4.4.7.1 TXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node. If the first syntax is used (omitting NamespaceURI), FindNode uses the value of the DefaultNamespaceURI (see page 7426) property.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name. When using ChildNodeType, FindNode returns the first child node it finds that implements the specified interface.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 7420)

First (see page 7420)

Last (see page 7424)

IndexOf (see page 7423)

LocalName (see page 7638)

NamespaceURI (see page 7639)

DefaultNamespaceURI (see page 7426)

82.1.4.4.8 TXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 7419)

First (see page 7420)

Last (see page 7424)

IndexOf (see page 7423)

82.1.4.4.9 TXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 7419)

FindSibling (see page 7420)

Last (see page 7424)

82.1.4.4.10 TXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 7426) property.

See Also

List (see page 7426)

GetNode (see page 7422)

Count (see page 7426)

82.1.4.4.11 TXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7426) property.

See Also

Count (see page 7426)

82.1.4.4.12 TXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode returns the interface to a node that is identified by name or by position.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 7419)

List (see page 7426)

LocalName (see page 7638)

Get (see page 7421)

82.1.4.4.13 TXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 7427) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 7427) property.

See Also

UpdateCount (see page 7427)

82.1.4.4.14 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	TXMLNodeList.IndexOf (DOMString) (see page 7423)	Returns the index of a specified node.

	TXMLNodeList.IndexOf (DOMString, DOMString) (see page 7423)	Returns the index of a specified node.
	TXMLNodeList.IndexOf (IXMLNode) (see page 7423)	Returns the index of a specified node.

82.1.4.4.14.1 TXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName. If Name is used without a NamespaceURI parameter (the first syntax), IndexOf uses the value of the DefaultNamespaceURI (see page 7426) property.

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 7419)

LocalName (see page 7638)

NamespaceURI (see page 7639)

DefaultNamespaceURI (see page 7426)

82.1.4.4.15 TXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 7417)

Remove (see page 7424)

IndexOf (see page 7423)

Add (see page 7416)

82.1.4.4.16 TXMLNodeList.InternalInsert Method

Provides the underlying implementation of the Insert (see page 7423) method

Pascal

```
function InternalInsert(Index: Integer; const Node: IXMLNode): Integer;
```

C++

```
__fastcall int InternalInsert(int Index, const IXMLNode Node);
```

Description

The Insert (see page 7423) method calls InternalInsert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index is –1, the node is added to the end of the list.

Node is the node to add to the list.

InternalInsert returns the index of the node after it is inserted. This is the same as the Index parameter when Index is not –1.

See Also

Insert (see page 7423)

82.1.4.4.17 TXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 7419)

FindSibling (see page 7420)

First (see page 7420)

82.1.4.4.18 TXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 7417)

Clear (see page 7417)

82.1.4.4.19 TXMLNodeList.ReplaceNode Method

Replaces a node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

List (see page 7426)

82.1.4.5 TXMLNodeList Properties

The properties of the TXMLNodeList class are listed here.

Protected Properties

	Name	Description
R	Count (see page 7426)	Indicates the number of nodes in the collection.
R	DefaultNamespaceURI (see page 7426)	Identifies the default Namespace for the nodes in this collection.
R	List (see page 7426)	Stores the interfaces for every node in the list.
R	Owner (see page 7427)	Specifies the node that uses this TXMLNodeList (see page 7411) object to manage a set of its child or attribute nodes.
R	UpdateCount (see page 7427)	Indicates the number of times an application has called BeginUpdate (see page 7416) without a matching call to EndUpdate (see page 7419).

82.1.4.5.1 TXMLNodeList.Count Property

Indicates the number of nodes in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to determine the number of nodes in the list. To change the value of Count, use the IXMLNodeList (see page 7649) interface. Add (see page 7416) nodes using the Add (see page 7416) or Insert (see page 7423) method, or remove them using the Delete (see page 7417) or Remove (see page 7424) method.

See Also

Nodes (see page 7660)

Add (see page 7651)

Delete (see page 7653)

Remove (see page 7659)

Insert (see page 7658)

82.1.4.5.2 TXMLNodeList.DefaultNamespaceURI Property

Identifies the default Namespace for the nodes in this collection.

Pascal

```
property DefaultNamespaceURI: DOMString;
```

C++

```
__property DOMString DefaultNamespaceURI;
```

Description

Read (see page 3727) DefaultNamespaceURI to determine the namespace that is used by default when locating nodes in the list. The IndexOf (see page 7423), FindNode (see page 7419), and Delete (see page 7417) methods all use this namespace when trying to match the name of a node if a namespace is not explicitly supplied.

The value of DefaultNamespaceURI is set by the TXMLNodeList (see page 7411) constructor.

See Also

NamespaceURI (see page 7639)

Create (see page 7414)

IndexOf (see page 7423)

FindNode (see page 7419)

Delete (see page 7417)

82.1.4.5.3 TXMLNodeList.List Property

Stores the interfaces for every node in the list.

Pascal

```
property List: IInterfaceList;
```

C++

```
__property IInterfaceList List;
```

Description

TXMLNodeList (see page 7411) uses List internally to store and manage the interfaces to the nodes it represents.

See Also

Count (see page 7426)

Nodes (see page 7660)

82.1.4.5.4 TXMLNodeList.Owner Property

Specifies the node that uses this TXMLNodeList (see page 7411) object to manage a set of its child or attribute nodes.

Pascal

```
property Owner: TXMLNode;
```

C++

```
__property TXMLNode Owner;
```

Description

Owner designates the node that instantiates TXMLNodeList (see page 7411) to manage a set of its child or attribute nodes. This value is supplied to the TXMLNodeList (see page 7411) constructor.

See Also

ChildNodes (see page 7391)

Create (see page 7414)

AttributeNodes (see page 7390)

82.1.4.5.5 TXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 7416) without a matching call to EndUpdate (see page 7419).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 7416), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 7419), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 7416) and EndUpdate (see page 7419) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0. For example, the callback that is supplied to the TXMLNodeList (see page 7411) constructor typically checks UpdateCount before responding to change notifications.

See Also

BeginUpdate (see page 7416)

EndUpdate (see page 7419)

Create (see page 7414)

82.1.4.6 TXMLNodeList Events

The events of the TXMLNodeList class are listed here.

Protected Events

	Name	Description
	NotificationProc (see page 7428)	Specifies a callback that TXMLNodeList (see page 7411) calls when nodes are added to or removed from the list, or when new nodes need to be created

82.1.4.6.1 TXMLNodeList.NotificationProc Event

Specifies a callback that TXMLNodeList (see page 7411) calls when nodes are added to or removed from the list, or when new nodes need to be created

Pascal

```
property NotificationProc: TNodeListNotification;
```

C++

```
__property TNodeListNotification NotificationProc;
```

Description

The node that instantiates TXMLNodeList (see page 7411) to manage a set of its child or attribute nodes supplies the value of this callback to the TXMLNodeList (see page 7411) constructor. TXMLNodeList (see page 7411) then calls this callback when its methods are used to add, create, or remove nodes.

See Also

ChildListNotify (see page 7360)

Create (see page 7414)

AttributeListNotify (see page 7358)

82.2 Interfaces

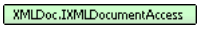
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 7429)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 7431)	This is class XMLDoc.IXMLNodeAccess.

82.2.1 IXMLDocumentAccess Interface

Class Hierarchy



Pascal

```
IXMLDocumentAccess = interface;
```

C++

```
__interface IXMLDocumentAccess;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLDocumentAccess.

82.2.1.1 IXMLDocumentAccess Members

The following tables list the members exposed by IXMLDocumentAccess.

Public Methods

	Name	Description
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

Public Properties

	Name	Description
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.

82.2.1.2 IXMLDocumentAccess Methods

The methods of the IXMLDocumentAccess class are listed here.

Public Methods

	Name	Description
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

82.2.1.2.1 IXMLDocumentAccess.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class IXMLDocumentAccess.

82.2.1.2.2 IXMLDocumentAccess.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class IXMLDocumentAccess.

82.2.1.3 IXMLDocumentAccess Properties

The properties of the IXMLDocumentAccess class are listed here.

Public Properties

	Name	Description
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.

82.2.1.3.1 IXMLDocumentAccess.DocumentObject Property

Pascal

```
property DocumentObject: TXMLDocument;
```

C++

```
__property TXMLDocument DocumentObject;
```

Description

This is DocumentObject, a member of class IXMLDocumentAccess.

82.2.1.3.2 IXMLDocumentAccess.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

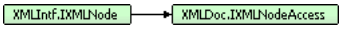
```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class IXMLDocumentAccess.

82.2.2 IXMLNodeAccess Interface

Class Hierarchy



Pascal

```
IXMLNodeAccess = interface(IXMLNode);
```

C++

```
__interface IXMLNodeAccess : public IXMLNode;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLNodeAccess.

82.2.2.1 IXMLNodeAccess Members

The following tables list the members exposed by IXMLNodeAccess.

Public Methods

	Name	Description
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.

	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.

R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

82.2.2.2 IXMLNodeAccess Methods

The methods of the IXMLNodeAccess class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7433)	This is AddChild, a member of class IXMLNodeAccess.
	CheckTextNode (see page 7434)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 7434)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CreateAttributeNode (see page 7434)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7434)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7434)	This is CreateCollection, a member of class IXMLNodeAccess.
	DOMElement (see page 7435)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 7435)	This is FindHostedNode, a member of class IXMLNodeAccess.
	GetChildNodeClasses (see page 7435)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetHostNode (see page 7435)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetNodeObject (see page 7436)	This is GetNodeObject, a member of class IXMLNodeAccess.
	HasChildNode (see page 7436)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7436)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 7437)	This is NestingLevel, a member of class IXMLNodeAccess.
	RegisterChildNode (see page 7437)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7437)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	SetCollection (see page 7437)	This is SetCollection, a member of class IXMLNodeAccess.
	SetParentNode (see page 7438)	This is SetParentNode, a member of class IXMLNodeAccess.

82.2.2.2.1 IXMLNodeAccess.AddChild Method

Pascal

```
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

This is AddChild, a member of class IXMLNodeAccess.

82.2.2.2.2 IXMLNodeAccess.CheckTextNode Method

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

This is CheckTextNode, a member of class IXMLNodeAccess.

82.2.2.2.3 IXMLNodeAccess.ClearDocumentRef Method

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

This is ClearDocumentRef, a member of class IXMLNodeAccess.

82.2.2.2.4 IXMLNodeAccess.CreateAttributeNode Method

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

This is CreateAttributeNode, a member of class IXMLNodeAccess.

82.2.2.2.5 IXMLNodeAccess.CreateChildNode Method

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

This is CreateChildNode, a member of class IXMLNodeAccess.

82.2.2.2.6 IXMLNodeAccess.CreateCollection Method

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemIterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemIterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

This is CreateCollection, a member of class IXMLNodeAccess.

82.2.2.2.7 IXMLNodeAccess.DOMElement Method

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

This is DOMElement, a member of class IXMLNodeAccess.

82.2.2.2.8 IXMLNodeAccess.FindHostedNode Method

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

This is FindHostedNode, a member of class IXMLNodeAccess.

82.2.2.2.9 IXMLNodeAccess.GetChildNodeClasses Method

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

This is GetChildNodeClasses, a member of class IXMLNodeAccess.

82.2.2.2.10 IXMLNodeAccess.GetHostNode Method

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

This is GetHostNode, a member of class IXMLNodeAccess.

82.2.2.2.11 IXMLNodeAccess.GetNodeObject Method

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

This is GetNodeObject, a member of class IXMLNodeAccess.

82.2.2.2.12 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	IXMLNodeAccess.HasChildNode (DOMString) (see page 7436)	This is HasChildNode, a member of class IXMLNodeAccess.
	IXMLNodeAccess.HasChildNode (DOMString, DOMString) (see page 7436)	This is HasChildNode, a member of class IXMLNodeAccess.

82.2.2.2.12.1 IXMLNodeAccess.HasChildNode Method (DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

82.2.2.2.12.2 IXMLNodeAccess.HasChildNode Method (DOMString, DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

82.2.2.2.13 IXMLNodeAccess.InternalAddChild Method

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

This is InternalAddChild, a member of class IXMLNodeAccess.

82.2.2.2.14 IXMLNodeAccess.NestingLevel Method

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

This is NestingLevel, a member of class IXMLNodeAccess.

82.2.2.2.15 IXMLNodeAccess.RegisterChildNode Method

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

This is RegisterChildNode, a member of class IXMLNodeAccess.

82.2.2.2.16 IXMLNodeAccess.RegisterChildNodes Method

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

This is RegisterChildNodes, a member of class IXMLNodeAccess.

82.2.2.2.17 IXMLNodeAccess.SetCollection Method

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

This is SetCollection, a member of class IXMLNodeAccess.

82.2.2.2.18 IXMLNodeAccess.SetParentNode Method

Pascal

```
procedure SetParentNode(const Value: TXMLNode);
```

C++

```
__fastcall SetParentNode(const TXMLNode Value);
```

Description

This is SetParentNode, a member of class IXMLNodeAccess.

82.2.2.3 IXMLNodeAccess Properties

The properties of the IXMLNodeAccess class are listed here.

Public Properties

	Name	Description
	ChildNodeClasses (see page 7438)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	HostNode (see page 7438)	This is HostNode, a member of class IXMLNodeAccess.

82.2.2.3.1 IXMLNodeAccess.ChildNodeClasses Property

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

This is ChildNodeClasses, a member of class IXMLNodeAccess.

82.2.2.3.2 IXMLNodeAccess.HostNode Property

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

This is HostNode, a member of class IXMLNodeAccess.

82.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckEncoding (see page 7439)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 7439)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 7440)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 7441)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 7441)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 7441)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 7442)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 7442)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 7443)	Raises an EXMLDocError (see page 7585) exception.
	XMLDocError (see page 7443)	Raises an EXMLDocError (see page 7585) exception.
	XMLStringToUnicodeString (see page 7443)	This is function XMLDoc.XMLStringToUnicodeString.
	XMLStringToWideString (see page 7443)	This is function XMLDoc.XMLStringToWideString.

82.3.1 XMLDoc.CheckEncoding Function

Pascal

```
procedure CheckEncoding(var XMLData: DOMString; const ValidEncodings: array of string);
```

C++

```
CheckEncoding(DOMString XMLData, const array of string ValidEncodings);
```

File

XMLDoc

Description

This is function XMLDoc.CheckEncoding.

82.3.2 XMLDoc.CreateDOMNode Function

Creates a new DOM node and returns its interface.

Pascal

```
function CreateDOMNode(Doc: IDOMDocument; const NameOrData: DOMString; NodeType: TNodeType =
ntElement; const AddlData: DOMString = ''): IDOMNode;
```

C++

```
IDOMNode CreateDOMNode(IDOMDocument Doc, const DOMString NameOrData, TNodeType NodeType =
ntElement, const DOMString AddlData = '');
```

File

XMLDoc

Description

CreateDOMNode generates a new DOM node.

Doc is the DOM interface for the document to which the new node will belong.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateDOMNode returns the interface for the newly created DOM node.

82.3.3 XMLDoc.DetectCharEncoding Function

Pascal

```
function DetectCharEncoding(S: TStream): TXmlEncodingType;
```

C++

```
TXmlEncodingType DetectCharEncoding(TStream S);
```

File

XMLDoc

Description

This is function XMLDoc.DetectCharEncoding.

82.3.4 XMLDoc.FormatXMLData Function

Formats a string of XML so that it is more readable.

Pascal

```
function FormatXMLData(const XMLData: DOMString): DOMString;
```

C++

```
DOMString FormatXMLData(const DOMString XMLData);
```

File

XMLDoc

Description

Use FormatXMLData to convert a string of XML into a format that represents its structure. FormatXMLData changes the input string (XMLData) so that each element node appears on its own line, indented appropriately to reflect its nesting in the node hierarchy.

82.3.5 XMLDoc.LoadXMLData Function

Creates and returns the interface of an XML document object, given a string of XML.

Pascal

```
function LoadXMLData(const XMLData: DOMString): IXMLDocument; overload;
function LoadXMLData(const XMLData: AnsiString): IXMLDocument; overload;
```

C++

```
IXMLDocument LoadXMLData(const DOMString XMLData);
IXMLDocument LoadXMLData(const AnsiString XMLData);
```

File

XMLDoc

Description

Call LoadXMLData to create an XML document that represents a specified XML string and obtain its interface.

XMLData is the XML for which you want an IXMLDocument (see page 7586) interface.

LoadXMLData returns an interface for the parsed value of XMLData.

LoadXMLData creates a new TXMLDocument (see page 7299) object to parse the XML and returns its IXMLDocument (see page 7586). The TXMLDocument (see page 7299) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLData returns, the TXMLDocument (see page 7299) instance is automatically freed.

See Also

TXMLDocument (see page 7299)

NewXMLDocument (see page 7442)

LoadXMLDocument (see page 7442)

TXMLDocument (see page 7299)

82.3.6 XMLDoc.LoadXMLDocument Function

Returns the interface for an XML document given the name of an XML file.

Pascal

```
function LoadXMLDocument(const FileName: DOMString): IXMLDocument;
```

C++

```
IXMLDocument LoadXMLDocument(const DOMString FileName);
```

File

XMLDoc

Description

Call LoadXMLDocument to load an XML document from a file on disk and obtain its interface. LoadXMLDocument creates a new TXMLDocument (see page 7299) object and uses it to parse the file specified by the FileName parameter. The TXMLDocument (see page 7299) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLDocument returns, the TXMLDocument (see page 7299) instance is automatically freed.

See Also

TXMLDocument (see page 7299)

NewXMLDocument (see page 7442)

LoadXMLData (see page 7441)

82.3.7 XMLDoc.NewXMLDocument Function

Returns the interface for a new, empty XML document.

Pascal

```
function NewXMLDocument(Version: DOMString = '1.0'): IXMLDocument;
```

C++

```
IXMLDocument NewXMLDocument(DOMString Version = '1.0');
```

File

XMLDoc

Description

Call NewXMLDocument to create a TXMLDocument (see page 7299) object for a new, empty XML document.

Version specifies the XML version that the document should use. It becomes the value of the new document's Version property.

See Also

TXMLDocument (see page 7299)

LoadXMLDocument (see page 7442)

IXMLDocument

82.3.8 XMLDoc.XMLDocError Function

Raises an EXMLDocError (see page 7585) exception.

Pascal

```
procedure XMLDocError(const Msg: string); overload;
procedure XMLDocError(const Msg: string; const Args: array of const); overload;
```

C++

```
XMLDocError(const UnicodeString Msg);
XMLDocError(const UnicodeString Msg, const array of const Args);
```

File

XMLDoc

Description

Call XMLDocError to raise an EXMLDocError (see page 7585) exception. Using XMLDocError rather than explicitly creating the EXMLDocError (see page 7585) instance can result in more space-efficient code.

Msg is the error message associated with the EXMLDocError (see page 7585) instance.

Args supplies arguments for any format specifiers embedded in Msg.

Note: Args_Size is the index of the last entry in the Args array. That is one less than the total number of arguments.

See Also

EXMLDocError

82.3.9 XMLDoc.XMLStringToUnicodeString Function

Pascal

```
function XMLStringToUnicodeString(const XMLString: AnsiString): UnicodeString;
```

C++

```
UnicodeString XMLStringToUnicodeString(const AnsiString XMLString);
```

File

XMLDoc

Description

This is function XMLDoc.XMLStringToUnicodeString.

82.3.10 XMLDoc.XMLStringToWideString Function

Pascal

```
function XMLStringToWideString(const XMLString: AnsiString): WideString; deprecated;
```

C++

```
BSTR XMLStringToWideString(const AnsiString XMLString);
```

File

XMLDoc

Description

This is function XMLDoc.XMLStringToWideString.

82.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeChange (see page 7445)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeListOperation (see page 7446)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7649) interface.
	TXMLDocumentSource (see page 7447)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 7448)	This is record XMLDoc.TXMLPrologItem.

Records

	Name	Description
	TNodeClassInfo (see page 7446)	TNodeClassInfo and TNodeClassArray (see page 7449) represent information about the classes that implement XML node interfaces.

82.4.1 XMLDoc.TNodeChange Enumeration

TNodeChange indicates the type of change made to a node in an XML document.

Pascal

```
TNodeChange = (
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
);
```

C++

```
enum TNodeChange {
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
};
```

File

XMLDoc

Description

The following table lists the possible values:

Value	Meaning
ncUpdateValue	The value of the node is changed.
ncInsertChild	A new child node is added to the node.
ncRemoveChild	A child node is removed from the node.

ncAddAttribute	A new attribute is added to the node.
ncRemoveAttribute	An attribute is removed from the node.

See Also

TNodeChangeEvent

82.4.2 XMLDoc.TNodeClassInfo Record

TNodeClassInfo and TNodeClassArray (see page 7449) represent information about the classes that implement XML node interfaces.

Pascal

```
TNodeClassInfo = record
  NodeName: DOMString;
  NamespaceURI: DOMString;
  NodeClass: TXMLNodeClass;
end;
```

C++

```
struct TNodeClassInfo {
  DOMString NodeName;
  DOMString NamespaceURI;
  TXMLNodeClass NodeClass;
};
```

File

XMLDoc

Description

TNodeClassArray (see page 7449) is a dynamic array of TNodeClassInfo values. Each element of the array describes the implementation of a node.

TNodeClassInfo describes the name and implementation class for a node in an XML document. It contains the following fields:

Field	Meaning
NodeName	The name of the node. This string describes the identifier or use of the node. For example, on element nodes, this is the tag name. On attribute nodes, this is the name of the attribute. On comment nodes, this is the string "#comment". This is the value returned by a node's NodeName property, and is used as a property name in classes generated by the XML Data Binding Wizard.
NamespaceURI	The namespace of the node's name. This string is a Uniform Resource Identifier that identifies the namespace that contains NodeName. It is used to interpret the value of NodeName.
NodeClass	The class of a TXMLNode (see page 7348) descendant that represents the node.

82.4.3 XMLDoc.TNodeListOperation Enumeration

TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7649) interface.

Pascal

```
TNodeListOperation = (
  nlInsert,
  nlRemove,
  nlCreateNode
);
```

C++

```
enum TNodeListOperation {
  nlInsert,
  nlRemove,
  nlCreateNode
};
```

File

XMLDoc

Description

When an application makes a change using the IXMLNodeList (see page 7649) interface, the implementing object sends its owner a notification of that change. TNodeListOperation identifies the type of change that is about to occur. The following table lists the possible values:

Value	Meaning
nlInsert	An existing node is about to be added to the list.
nlRemove	A node is about to be removed from the list.
nlCreateNode	A new node needs to be created so that it can be added to the list.

82.4.4 XMLDoc.TXMLDocumentSource Enumeration

Pascal

```
TXMLDocumentSource = (
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
);
```

C++

```
enum TXMLDocumentSource {
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLDocumentSource.

82.4.5 XMLDoc.TXMLPrologItem Enumeration

Pascal

```
TXMLPrologItem = (
  xpVersion,
  xpEncoding,
  xpStandalone
);
```

C++

```
enum TXMLPrologItem {
  xpVersion,
  xpEncoding,
  xpStandalone
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLPrologItem.

82.5 Types

The following table lists types in this documentation.

Types

Name	Description
TNodeChangeEvent (see page 7449)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 7449)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 7450)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 7411) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 7451)	TXMLNodeArray is a dynamic array of TXMLNode (see page 7348) objects.
TXMLNodeClass (see page 7451)	TXMLNodeClass is a class reference to a TXMLNode (see page 7348) descendant.
TXMLNodeCollectionClass (see page 7452)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7396) descendant.

82.5.1 XMLDoc.TNodeChangeEvent Type

TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.

Pascal

```
TNodeChangeEvent = procedure (const Node: IXMLNode; ChangeType: TNodeChange) of object;
```

C++

```
(const Node: IXMLNode; ChangeType: TNodeChange) ( TNodeChangeEvent)();
```

File

XMLDoc

Description

TNodeChangeEvent is the type for the BeforeNodeChange and AfterNodeChange event handlers. These events occur before and after changes are made to the nodes in an XML document.

Node identifies the node to which the change occurs.

ChangeType indicates the type of change made to Node.

See Also

BeforeNodeChange

AfterNodeChange

82.5.2 XMLDoc.TNodeClassArray Type

Pascal

```
TNodeClassArray = array of TNodeClassInfo;
```

C++

```
array of TNodeClassInfo TNodeClassArray;
```

File

XMLDoc

Description

This is type XMLDoc.TNodeClassArray.

82.5.3 XMLDoc.TNodeListNotification Type

TNodeListNotification is the type of the callback the TXMLNodeList (see page 7411) calls when it adds, deletes, or creates XML nodes.

Pascal

```
TNodeListNotification = procedure (Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean) of object;
```

C++

```
(Operation: TNodeListOperation; var Node: IXMLNode; const IndexOrName: OleVariant;
BeforeOperation: Boolean) ( TNodeListNotification)();
```

File

XMLDoc

Description

TNodeListNotification is the type for the callback supplied to TXMLNodeList (see page 7411) in its constructor. TXMLNodeList (see page 7411) calls this procedure before and after it adds or deletes nodes in the list and when it needs to create a new node.

Operation indicates what change is about to occur or has just occurred.

Node specifies the node that is the focus of that change. It's use depends on the operation, as indicated in the following table:

Operation	Node
nlInsert	The node that is about to be added to the list or that has just been added.
nlRemove	The node that is about to be removed from the list or that has just been removed.
nlCreateNode	The newly-created node. This should be created by the callback and returned as the Node parameter.
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

IndexOrName is used only when Operation is nlCreateNode. It is the value assigned to the Nodes property using the index of a node that does not yet exist. The interpretation of IndexOrName depends on the type of node that the callback creates, as indicated in the following table:

NodeType	IndexOrName
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

BeforeOperation indicates whether the callback occurs before the indicated operation or after. When nodes are inserted and removed, the callback is called first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

TXMLNodeList

TXMLNode

TXMLNode

IXMLNodeList

82.5.4 XMLDoc.TXMLNodeArray Type

TXMLNodeArray is a dynamic array of TXMLNode (see page 7348) objects.

Pascal

```
TXMLNodeArray = array of TXMLNode;
```

C++

```
array of TXMLNode TXMLNodeArray;
```

File

XMLDoc

Description

TXMLNodeArray is a dynamic array of TXMLNode (see page 7348) objects, each of which represents a node in an XML document.

82.5.5 XMLDoc.TXMLNodeClass Type

TXMLNodeClass is a class reference to a TXMLNode (see page 7348) descendant.

Pascal

```
TXMLNodeClass = class of TXMLNode;
```

C++

```
TXMLNode TXMLNodeClass;
```

File

XMLDoc

Description

TXMLNodeClass describes a class that descends from TXMLNode (see page 7348).

82.5.6 XMLDoc.TXMLNodeCollectionClass Type

TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7396) descendant.

Pascal

```
TXMLNodeCollectionClass = class of TXMLNodeCollection;
```

C++

```
TXMLNodeCollection TXMLNodeCollectionClass;
```

File

XMLDoc

Description

TXMLNodeCollectionClass describes a class that descends from TXMLNodeCollection (see page 7396).

82.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
UseXSDBooleanStrings (see page 7453)	This is variable XMLDoc.UseXSDBooleanStrings.

82.6.1 XMLDoc.UseXSDBooleanStrings Variable

Pascal

```
UseXSDBooleanStrings: Boolean = True;
```

C++

```
Boolean UseXSDBooleanStrings = True;
```

File

XMLDoc

Description

This is variable XMLDoc.UseXSDBooleanStrings.

83 xmldom Namespace

This is namespace xmldom.

Topics

Name	Description
Classes (see page 7458)	The following table lists classes in this documentation.
Interfaces (see page 7473)	The following table lists interfaces in this documentation.
Functions (see page 7557)	The following table lists functions in this documentation.
Types (see page 7565)	The following table lists types in this documentation.
Variables (see page 7568)	The following table lists variables in this documentation.
Constants (see page 7569)	The following table lists constants in this documentation.

Classes

	Name	Description
	DOMException (see page 7458)	This is class xmldom.DOMException.
	EDOMParseError (see page 7460)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 7466)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 7468)	TDOMVendorList represents a collection of available DOM implementations.

Constants

Name	Description
ATTRIBUTE_NODE (see page 7569)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 7570)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 7570)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 7570)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 7571)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 7571)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 7571)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 7572)	This is constant xmldom.DOMWrapperVersion.
ELEMENT_NODE (see page 7572)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 7572)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 7573)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 7573)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 7573)	This is constant xmldom.INDEX_SIZE_ERR.

INUSE_ATTRIBUTE_ERR (see page 7574)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 7574)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 7574)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 7575)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 7575)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 7575)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 7576)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 7576)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 7576)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 7577)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 7577)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 7577)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 7578)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 7578)	This is constant xmldom.SEncoding.
SHttp (see page 7578)	This is constant xmldom.SHttp.
SStandalone (see page 7579)	This is constant xmldom.SStandalone.
SVersion (see page 7579)	This is constant xmldom.SVersion.
SXML (see page 7579)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 7580)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 7580)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 7580)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 7581)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 7581)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 7581)	This is constant xmldom.WRONG_DOCUMENT_ERR.

Functions

	Name	Description
	DOMVendorNotSupported (see page 7557)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 7557)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 7558)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 7559)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 7559)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 7560)	Returns the TDOMVendor (see page 7466) object for a specified DOM vendor.
	IsPrefixed (see page 7560)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 7561)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 7561)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 7562)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 7562)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 7562)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 7563)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7562) .

Interfaces

	Name	Description
	IDOMAttr (see page 7473)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 7477)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 7478)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 7482)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 7482)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 7489)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 7490)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 7494)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 7501)	This is class xmldom.IDOMEntity.
	IDOMEntityReference (see page 7504)	This is class xmldom.IDOMEntityReference.

	IDOMImplementation (see page 7504)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 7506)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 7509)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 7523)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 7526)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 7528)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 7530)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 7532)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 7537)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 7541)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 7545)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 7548)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 7550)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 7553)	This is class xmldom.IDOMXSLProcessor.

Types

Name	Description
DOMNodeType (see page 7565)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 7565)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 7566)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 7566)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 7567)	This is type xmldom.TDOMVendorArray.

Variables

Name	Description
DefaultDOMVendor (see page 7568)	This is variable xmldom.DefaultDOMVendor.
DOMVendors (see page 7568)	This is variable xmldom.DOMVendors.

83.1 Classes

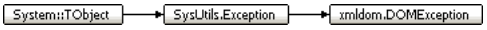
The following table lists classes in this documentation.

Classes

	Name	Description
	DOMException (see page 7458)	This is class xmldom.DOMException.
	EDOMParseError (see page 7460)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 7466)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 7468)	TDOMVendorList represents a collection of available DOM implementations.

83.1.1 DOMException Class

Class Hierarchy



Pascal

```
DOMException = class(Exception);
```

C++

```
class DOMException : public Exception;
```

File

xmldom

Description

This is class xmldom.DOMException.

83.1.1.1 DOMException Members

The following tables list the members exposed by DOMException.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	code (see page 7460)	This is code, a member of class DOMException.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3366)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4018)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
	Destroy (see page 3366)	Disposes of an object instance.
	Destroy (see page 4026)	Disposes of an object instance.
	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.

83.1.1.2 DOMException Fields

The fields of the DOMException class are listed here.

Public Fields

	Name	Description
	code (see page 7460)	This is code, a member of class DOMException.

83.1.1.2.1 DOMException.code Field

Pascal

```
code: Word;
```

C++

```
Word code;
```

Description

This is code, a member of class DOMException.

83.1.2 EDOMParseError Class

EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.

Class Hierarchy

Pascal

```
EDOMParseError = class(Exception);
```

C++

```
class EDOMParseError : public Exception;
```

File

xmldom

Description

EDOMParseError is raised when TXMLDocument encounters an error while parsing an XML document. It wraps the information that is available from the DOM implementation's IDOMParseError (see page 7532) interface.

See Also

TXMLDocument (see page 7299)

83.1.2.1 EDOMParseError Members

The following tables list the members exposed by EDOMParseError.

Protected Methods

	Name	Description
	GetStackTrace (see page 4028)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 4028)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 4028)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 4029)	This is SetStackInfo, a member of class Exception.

Protected Properties

	Name	Description
R	ParseError (see page 7464)	This is ParseError, a member of class EDOMParseError.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 4026)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 4027)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 4027)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
V	AfterConstruction (see page 3367)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3368)	Responds before the first destructor executes.
	ClassInfo (see page 3368)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3369)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3371)	Determines whether an object is of a specific type.
	ClassParent (see page 3372)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3374)	Returns the class reference for the object's class.
	CleanupInstance (see page 3377)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7462)	This is Create, a member of class EDOMParseError.
	CreateFmt (see page 4020)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 4021)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 4021)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 4022)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 4022)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 4023)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 4025)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 3377)	Provides the interface for a method that processes message records.
V	Destroy (see page 3366)	Disposes of an object instance.
V	Destroy (see page 4026)	Disposes of an object instance.
V	Dispatch (see page 3379)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3381)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3382)	This is the overview for the FieldAddress method overload.
	Free (see page 3382)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3386)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3391) method.
	GetBaseException (see page 4027)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3386)	Returns an integer containing the hash code.
	GetInterface (see page 3387)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3387)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3388)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3388)	Determines the relationship of two object types.
	InitInstance (see page 3389)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3389)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3389)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3390)	Returns the name of a class method by address.
	NewInstance (see page 3391)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 4028)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3391)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 4029)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3391)	Returns a string containing the class name.
	ToString (see page 4029)	Returns a string containing the class name.
	UnitName (see page 3392)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 4030)	This is BaseException, a member of class Exception.
	ErrorCode (see page 7463)	Indicates the error code that the DOM parser returned.
	FilePos (see page 7463)	Indicates the offset in the XML document where the parser encountered an error.
	HelpContext (see page 4030)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 4031)	This is InnerException, a member of class Exception.
	Line (see page 7464)	Indicates the line of the XML document where the parser encountered an error.
	LinePos (see page 7464)	Indicates the position within the problem line where the parser encountered an error.
	Message (see page 4031)	Message contains the text string to display in the exception dialog box when the exception is raised.
	Reason (see page 7465)	Indicates the error code that the DOM parser returned.
	SrcText (see page 7465)	Indicates the text of the line where the parser encountered an error.
	StackInfo (see page 4031)	This is StackInfo, a member of class Exception.
	StackTrace (see page 4032)	This is StackTrace, a member of class Exception.
	URL (see page 7465)	Identifies the source that could not be parsed by its URL.

83.1.2.2 EDOMParseError.Create Constructor

Pascal

```
constructor Create(const ParseError: IDOMParseError; const Msg: string);
```

C++

```
__fastcall EDOMParseError(const IDOMParseError ParseError, const UnicodeString Msg);
```

Description

This is Create, a member of class EDOMParseError.

83.1.2.3 EDOMParseError Properties

The properties of the EDOMParseError class are listed here.

Protected Properties

	Name	Description
	ParseError (see page 7464)	This is ParseError, a member of class EDOMParseError.

Public Properties

	Name	Description
	ErrorCode (see page 7463)	Indicates the error code that the DOM parser returned.
	FilePos (see page 7463)	Indicates the offset in the XML document where the parser encountered an error.
	Line (see page 7464)	Indicates the line of the XML document where the parser encountered an error.
	LinePos (see page 7464)	Indicates the position within the problem line where the parser encountered an error.
	Reason (see page 7465)	Indicates the error code that the DOM parser returned.
	SrcText (see page 7465)	Indicates the text of the line where the parser encountered an error.
	URL (see page 7465)	Identifies the source that could not be parsed by its URL.

83.1.2.3.1 EDOMParseError.ErrorCode Property

Indicates the error code that the DOM parser returned.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

ErrorCode is the error code supplied by the DOM parser when it encountered a problem parsing the XML document.

See Also

Reason (see page 7465)

83.1.2.3.2 EDOMParseError.FilePos Property

Indicates the offset in the XML document where the parser encountered an error.

Pascal

```
property FilePos: Integer;
```

C++

```
__property int FilePos;
```

Description

FilePos is the offset in the XML document where the parser detected the error.

See Also

Line (see page 7464)

LinePos (see page 7464)

SrcText (see page 7465)

URL (see page 7465)

83.1.2.3.3 EDOMParseError.Line Property

Indicates the line of the XML document where the parser encountered an error.

Pascal

```
property Line: Integer;
```

C++

```
__property int Line;
```

Description

Line is the line of the XML document where the parser detected the error.

See Also

FilePos (see page 7463)

LinePos (see page 7464)

SrcText (see page 7465)

URL (see page 7465)

83.1.2.3.4 EDOMParseError.LinePos Property

Indicates the position within the problem line where the parser encountered an error.

Pascal

```
property LinePos: Integer;
```

C++

```
__property int LinePos;
```

Description

LinePos is the offset in the line specified by the Line (see page 7464) property where the DOM parser detected an error.

See Also

Line (see page 7464)

FilePos (see page 7463)

SrcText (see page 7465)

URL (see page 7465)

83.1.2.3.5 EDOMParseError.ParseError Property

Pascal

```
property ParseError: IDOMParseError;
```

C++

```
__property IDOMParseError ParseError;
```

Description

This is ParseError, a member of class EDOMParseError.

83.1.2.3.6 EDOMParseError.Reason Property

Indicates the error code that the DOM parser returned.

Pascal

```
property Reason: DOMString;
```

C++

```
__property DOMString Reason;
```

Description

Reason is a short text string that describes the error that was encountered. It corresponds to the error code specified by ErrorCode (see page 7463).

See Also

ErrorCode (see page 7463)

Message (see page 4031)

83.1.2.3.7 EDOMParseError.SrcText Property

Indicates the text of the line where the parser encountered an error.

Pascal

```
property SrcText: DOMString;
```

C++

```
__property DOMString SrcText;
```

Description

SrcText is the text from the XML document that the DOM parser could not interpret.

See Also

Line (see page 7464)

83.1.2.3.8 EDOMParseError.URL Property

Identifies the source that could not be parsed by its URL.

Pascal

```
property URL: DOMString;
```

C++

```
__property DOMString URL;
```

Description

URL identifies the problem the DOM parser encountered by its URL.

See Also

SrcText (see page 7465)

83.1.3 TDOMVendor Class

TDOMVendor is the base class for components that provide access to a DOM implementation.

Class Hierarchy

Pascal

```
TDOMVendor = class;
```

C++

```
class TDOMVendor;
```

File

xmldom

Description

Note: Use TDOMVendor as a base class for a class that can be used to represent a DOM implementation. TDOMVendor introduces two abstract (pure virtual) methods, which provide information about a specific DOM implementation. One returns the IDOMImplementation (see page 7504) interface that acts as the top-level interface for a DOM vendor's implementation. The other returns a unique string that identifies the DOM implementation.

Once a TDOMVendor object has been registered, TXMLDocument objects can assign the DOM implementation as the value of their DOMVendor property by choosing the identifying string in the Object Inspector.

See Also

RegisterDOMVendor

UnregisterDOMVendor

TXMLDocument (see page 7299)

GetDOMVendor

Using the Document Object Model

83.1.3.1 TDOMVendor Members

The following tables list the members exposed by TDOMVendor.

Public Methods

	Name	Description
	Description (see page 7467)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 7467)	Returns the IDOMImplementation (see page 7504) interface for the associated DOM implementation.

83.1.3.2 TDOMVendor Methods

The methods of the TDOMVendor class are listed here.

Public Methods

	Name	Description
	Description (see page 7467)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 7467)	Returns the IDOMImplementation (see page 7504) interface for the associated DOM implementation.

83.1.3.2.1 TDOMVendor.Description Method

Returns a string that uniquely identifies the associated DOM implementation.

Pascal

```
function Description: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString Description() = 0;
```

Description

Description returns the name, or description, of the associated DOM implementation. This is the string that appears in the Object Inspector for a TXMLDocument object's DOMVendor property.

Description is an abstract (pure virtual) method in TDOMVendor (see page 7466). Descendant classes override this method to provide a unique identifier for the DOM implementation.

See Also

DOMVendor (see page 7337)

GetDOMVendor

83.1.3.2.2 TDOMVendor.DOMImplementation Method

Returns the IDOMImplementation (see page 7504) interface for the associated DOM implementation.

Pascal

```
function DOMImplementation: IDOMImplementation; virtual; abstract;
```

C++

```
virtual __fastcall IDOMImplementation DOMImplementation() = 0;
```

Description

DOMImplementation returns the top-level interface for a DOM implementation. This interface is the starting point for any application that uses the DOM implementation to parse and edit an XML document.

DOMImplementation is an abstract (pure virtual) method in TDOMVendor (see page 7466). Descendant classes override this method to return the IDOMImplementation (see page 7504) interface for the associated DOM implementation.

Note: For more information about the IDOMImplementation (see page 7504), see the W3C Document Object Model (DOM) Level 2 Specification or the documentation provided by your DOM vendor.

See Also

domimplementation (see page 7337)

Description (see page 7467)

83.1.4 TDOMVendorList Class

TDOMVendorList represents a collection of available DOM implementations.

Class Hierarchy

Pascal

```
TDOMVendorList = class;
```

C++

```
class TDOMVendorList;
```

File

xmldom

Description

TDOMVendorList is the type of the global DOMVendors (see page 7568) variable. It represents a collection of DOM implementations (TDOMVendor (see page 7466) objects).

TDOMVendorList introduces a single property, Vendors (see page 7471), that can be used to iterate through all the DOM vendors in the collection. It also introduces methods that can be used to add, delete, or locate vendors.

Note: Do not explicitly use TDOMVendorList methods to add or delete vendors from the global DOMVendors (see page 7568) variable. Instead, add or remove vendors from this global list by calling the global RegisterDOMVendor (see page 7562) and UnRegisterDOMVendor (see page 7563) procedures.

See Also

DOMVendors

TXMLDocument (see page 7299)

RegisterDOMVendor

TDOMVendor (see page 7466)

UnregisterDOMVendor

83.1.4.1 TDOMVendorList Members

The following tables list the members exposed by TDOMVendorList.

Protected Methods

	Name	Description
	GetVendors (see page 7470)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 7469)	Adds a new DOM implementation to the end of the list.
	Count (see page 7469)	Returns the number of DOM vendors in the collection.
	Find (see page 7470)	Returns a vendor from the list, given its description.
	Remove (see page 7471)	Removes a specified vendor from the list.

Public Properties

	Name	Description
	Vendors (see page 7471)	Lists the DOM vendors maintained by this TDOMVendorList (see page 7468) object.

83.1.4.2 TDOMVendorList Methods

The methods of the TDOMVendorList class are listed here.

Protected Methods

	Name	Description
	GetVendors (see page 7470)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 7469)	Adds a new DOM implementation to the end of the list.
	Count (see page 7469)	Returns the number of DOM vendors in the collection.
	Find (see page 7470)	Returns a vendor from the list, given its description.
	Remove (see page 7471)	Removes a specified vendor from the list.

83.1.4.2.1 TDOMVendorList.Add Method

Adds a new DOM implementation to the end of the list.

Pascal

```
procedure Add(const Vendor: TDOMVendor);
```

C++

```
__fastcall Add(const TDOMVendor Vendor);
```

Description

Call Add to add a DOM vendor to the end of the list.

Vendor is an object that provides access to the IDOMImplementation (see page 7504) interface for the new DOM vendor.

If the list already includes a DOM vendor with the same description as that supplied by Vendor, Add raises an exception.

See Also

Count

Vendors

Find

Remove

Description (see page 7467)

83.1.4.2.2 TDOMVendorList.Count Method

Returns the number of DOM vendors in the collection.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Call Count to determine the number of DOM implementations stored in the list. Count returns a value that can be used as an upper bound when iterating over the vendors using the Vendors (see page 7471) property.

Count increases when the Add (see page 7469) method adds a vendor to the list and decreases when the Remove (see page 7471) method removes a vendor from the list.

See Also

Vendors

Add

Remove

83.1.4.2.3 TDOMVendorList.Find Method

Returns a vendor from the list, given its description.

Pascal

```
function Find(const VendorDesc: string): TDOMVendor;
```

C++

```
__fastcall TDOMVendor Find(const UnicodeString VendorDesc);
```

Description

Call Find to access a vendor in the list, given its description. If there is no vendor in the list with the specified description, Find returns nil (Delphi) or NULL (C++).

VendorDesc is the string that is returned by the TDOMVendor (see page 7466) object's Description method.

Find returns a TDOMVendor (see page 7466) object, which can be used as a value for the DOMVendor property of a TXMLDocument component.

See Also

Vendors

Description (see page 7467)

DOMVendor (see page 7337)

83.1.4.2.4 TDOMVendorList.GetVendors Method

Pascal

```
function GetVendors(Index: Integer): TDOMVendor;
```

C++

```
__fastcall TDOMVendor GetVendors(int Index);
```

Description

This is GetVendors, a member of class TDOMVendorList.

83.1.4.2.5 TDOMVendorList.Remove Method

Removes a specified vendor from the list.

Pascal

```
procedure Remove(const Vendor: TDOMVendor);
```

C++

```
__fastcall Remove(const TDOMVendor Vendor);
```

Description

Remove removes the specified vendor from the list.

Vendor is the DOM vendor to remove from the list. If the list does not include a vendor with the same description as this parameter, Remove does nothing.

See Also

Count

Vendors

Find

Add

Description (see page 7467)

83.1.4.3 TDOMVendorList Properties

The properties of the TDOMVendorList class are listed here.

Public Properties

	Name	Description
	Vendors (see page 7471)	Lists the DOM vendors maintained by this TDOMVendorList (see page 7468) object.

83.1.4.3.1 TDOMVendorList.Vendors Property

Lists the DOM vendors maintained by this TDOMVendorList (see page 7468) object.

Pascal

```
property Vendors [Index: Integer]: TDOMVendor;
```

C++

```
__property TDOMVendor Vendors[int Index];
```

Description

Use Vendors to access the DOM vendors in this collection. Index is the index of the vendor. Its value ranges from 0 to one less than the value returned by the Count (see page 7469) method.

Note: In Delphi, Vendors is the default property for TDOMVendorList (see page 7468). This means that the property name can be omitted in code. That is, the statement

for i := 0 to DOMVendors. Count (see page 7469) do

StringList1.Add (see page 7469)(DOMVendors.Vendors[i].Description);

Can be written

for i := 0 to DOMVendors.Count (see page 7469) do

StringList1.Add (see page 7469)(DOMVendors (see page 7568)[i].Description);

Note: In C++, Vendors can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Count

Add

Find

Remove

operator_sb

83.2 Interfaces

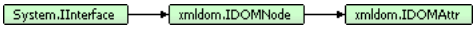
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDOMAttr (see page 7473)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 7477)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 7478)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 7482)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 7482)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 7489)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 7490)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 7494)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 7501)	This is class xmldom.IDOMEntity.
	IDOMEntityReference (see page 7504)	This is class xmldom.IDOMEntityReference.
	IDOMImplementation (see page 7504)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 7506)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 7509)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 7523)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 7526)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 7528)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 7530)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 7532)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 7537)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 7541)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 7545)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 7548)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 7550)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 7553)	This is class xmldom.IDOMXSLProcessor.

83.2.1 IDOMAttr Interface

Class Hierarchy



Pascal

```
IDOMAttr = interface(IDOMNode);
```

C++

```
__interface IDOMAttr : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMAttr.

83.2.1.1 IDOMAttr Members

The following tables list the members exposed by IDOMAttr.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_name (see page 7475)	This is get_name, a member of class IDOMAttr.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_ownerElement (see page 7475)	This is get_ownerElement, a member of class IDOMAttr.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_specified (see page 7475)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 7476)	This is get_value, a member of class IDOMAttr.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	set_value (see page 7476)	This is set_value, a member of class IDOMAttr.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	name (see page 7476)	This is name, a member of class IDOMAttr.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.

	ownerElement (see page 7476)	This is ownerElement, a member of class IDOMAttr.
	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
	specified (see page 7477)	This is specified, a member of class IDOMAttr.
	value (see page 7477)	This is value, a member of class IDOMAttr.

83.2.1.2 IDOMAttr Methods

The methods of the IDOMAttr class are listed here.

Public Methods

	Name	Description
	get_name (see page 7475)	This is get_name, a member of class IDOMAttr.
	get_ownerElement (see page 7475)	This is get_ownerElement, a member of class IDOMAttr.
	get_specified (see page 7475)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 7476)	This is get_value, a member of class IDOMAttr.
	set_value (see page 7476)	This is set_value, a member of class IDOMAttr.

83.2.1.2.1 IDOMAttr.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMAttr.

83.2.1.2.2 IDOMAttr.get_ownerElement Method

Pascal

```
function get_ownerElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_ownerElement();
```

Description

This is get_ownerElement, a member of class IDOMAttr.

83.2.1.2.3 IDOMAttr.get_specified Method

Pascal

```
function get_specified: WordBool; safecall;
```

C++

```
__fastcall WordBool get_specified();
```

Description

This is get_specified, a member of class IDOMAttr.

83.2.1.2.4 IDOMAttr.get_value Method

Pascal

```
function get_value: DOMString; safecall;
```

C++

```
__fastcall DOMString get_value();
```

Description

This is get_value, a member of class IDOMAttr.

83.2.1.2.5 IDOMAttr.set_value Method

Pascal

```
procedure set_value(const attributeValue: DOMString); safecall;
```

C++

```
__fastcall set_value(const DOMString attributeValue);
```

Description

This is set_value, a member of class IDOMAttr.

83.2.1.3 IDOMAttr Properties

The properties of the IDOMAttr class are listed here.

Public Properties

	Name	Description
R	name (see page 7476)	This is name, a member of class IDOMAttr.
R	ownerElement (see page 7476)	This is ownerElement, a member of class IDOMAttr.
R	specified (see page 7477)	This is specified, a member of class IDOMAttr.
	value (see page 7477)	This is value, a member of class IDOMAttr.

83.2.1.3.1 IDOMAttr.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMAttr.

83.2.1.3.2 IDOMAttr.ownerElement Property

Pascal

```
property ownerElement: IDOMElement;
```

C++

```
__property IDOMElement ownerElement;
```

Description

This is ownerElement, a member of class IDOMAttr.

83.2.1.3.3 IDOMAttr.specified Property

Pascal

```
property specified: WordBool;
```

C++

```
__property WordBool specified;
```

Description

This is specified, a member of class IDOMAttr.

83.2.1.3.4 IDOMAttr.value Property

Pascal

```
property value: DOMString;
```

C++

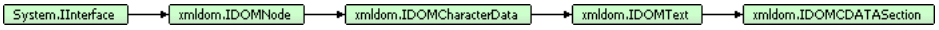
```
__property DOMString value;
```

Description

This is value, a member of class IDOMAttr.

83.2.2 IDOMCDATASection Interface

Class Hierarchy



Pascal

```
IDOMCDATASection = interface(IDOMText);
```

C++

```
__interface IDOMCDATASection : public IDOMText;
```

File

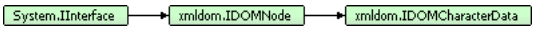
xmldom

Description

This is class xmldom.IDOMCDATASection.

83.2.3 IDOMCharacterData Interface

Class Hierarchy



Pascal

```
IDOMCharacterData = interface(IDOMNode);
```

C++

```
__interface IDOMCharacterData : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMCharacterData.

83.2.3.1 IDOMCharacterData Members

The following tables list the members exposed by IDOMCharacterData.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	appendData (see page 7479)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	deleteData (see page 7480)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_data (see page 7480)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_length (see page 7480)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	insertData (see page 7480)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	replaceData (see page 7480)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7481)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 7519)	Sets the value of the node.
	substringData (see page 7481)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
	data (see page 7481)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	length (see page 7481)	This is length, a member of class IDOMCharacterData.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.

83.2.3.2 IDOMCharacterData Methods

The methods of the IDOMCharacterData class are listed here.

Public Methods

	Name	Description
	appendData (see page 7479)	This is appendData, a member of class IDOMCharacterData.
	deleteData (see page 7480)	This is deleteData, a member of class IDOMCharacterData.
	get_data (see page 7480)	This is get_data, a member of class IDOMCharacterData.
	get_length (see page 7480)	This is get_length, a member of class IDOMCharacterData.
	insertData (see page 7480)	This is insertData, a member of class IDOMCharacterData.
	replaceData (see page 7480)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7481)	This is set_data, a member of class IDOMCharacterData.
	substringData (see page 7481)	This is substringData, a member of class IDOMCharacterData.

83.2.3.2.1 IDOMCharacterData.appendData Method

Pascal

```
procedure appendData(const data: DOMString); safecall;
```

C++

```
__fastcall appendData(const DOMString data);
```

Description

This is appendData, a member of class IDOMCharacterData.

83.2.3.2.2 IDOMCharacterData.deleteData Method

Pascal

```
procedure deleteData(offset: Integer; count: Integer); safecall;
```

C++

```
__fastcall deleteData(int offset, int count);
```

Description

This is deleteData, a member of class IDOMCharacterData.

83.2.3.2.3 IDOMCharacterData.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMCharacterData.

83.2.3.2.4 IDOMCharacterData.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMCharacterData.

83.2.3.2.5 IDOMCharacterData.insertData Method

Pascal

```
procedure insertData(offset: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall insertData(int offset, const DOMString data);
```

Description

This is insertData, a member of class IDOMCharacterData.

83.2.3.2.6 IDOMCharacterData.replaceData Method

Pascal

```
procedure replaceData(offset: Integer; count: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall replaceData(int offset, int count, const DOMString data);
```

Description

This is replaceData, a member of class IDOMCharacterData.

83.2.3.2.7 IDOMCharacterData.set_data Method

Pascal

```
procedure set_data(const data: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString data);
```

Description

This is set_data, a member of class IDOMCharacterData.

83.2.3.2.8 IDOMCharacterData.substringData Method

Pascal

```
function substringData(offset: Integer; count: Integer): DOMString; safecall;
```

C++

```
__fastcall DOMString substringData(int offset, int count);
```

Description

This is substringData, a member of class IDOMCharacterData.

83.2.3.3 IDOMCharacterData Properties

The properties of the IDOMCharacterData class are listed here.

Public Properties

	Name	Description
	data (see page 7481)	This is data, a member of class IDOMCharacterData.
R	length (see page 7481)	This is length, a member of class IDOMCharacterData.

83.2.3.3.1 IDOMCharacterData.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMCharacterData.

83.2.3.3.2 IDOMCharacterData.length Property

Pascal

```
property length: Integer;
```

C++

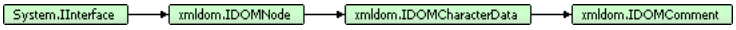
```
__property int length;
```

Description

This is length, a member of class IDOMCharacterData.

83.2.4 IDOMComment Interface

Class Hierarchy



Pascal

```
IDOMComment = interface(IDOMCharacterData);
```

C++

```
__interface IDOMComment : public IDOMCharacterData;
```

File

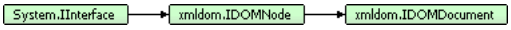
xmldom

Description

This is class xmldom.IDOMComment.

83.2.5 IDOMDocument Interface

Class Hierarchy



Pascal

```
IDOMDocument = interface(IDOMNode);
```

C++

```
__interface IDOMDocument : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocument.

83.2.5.1 IDOMDocument Members

The following tables list the members exposed by IDOMDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	createAttribute (see page 7484)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 7485)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 7485)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 7485)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 7485)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 7485)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 7486)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 7486)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 7486)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 7486)	This is createTextNode, a member of class IDOMDocument.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_doctype (see page 7487)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 7487)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 7487)	This is get_domImplementation, a member of class IDOMDocument.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	getElementById (see page 7487)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 7488)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 7488)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	importNode (see page 7488)	This is importNode, a member of class IDOMDocument.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_documentElement (see page 7488)	This is set_documentElement, a member of class IDOMDocument.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	doctype (see page 7489)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 7489)	This is documentElement, a member of class IDOMDocument.

	domImplementation (see page 7489)	This is domImplementation, a member of class IDOMDocument.
	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
	localName (see page 7521)	This is localName, a member of class IDOMNode.
	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.

83.2.5.2 IDOMDocument Methods

The methods of the IDOMDocument class are listed here.

Public Methods

	Name	Description
	createAttribute (see page 7484)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 7485)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 7485)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 7485)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 7485)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 7485)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 7486)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 7486)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 7486)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 7486)	This is createTextNode, a member of class IDOMDocument.
	get_doctype (see page 7487)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 7487)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 7487)	This is get_domImplementation, a member of class IDOMDocument.
	getElementById (see page 7487)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 7488)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 7488)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	importNode (see page 7488)	This is importNode, a member of class IDOMDocument.
	set_documentElement (see page 7488)	This is set_documentElement, a member of class IDOMDocument.

83.2.5.2.1 IDOMDocument.createAttribute Method

Pascal

```
function createAttribute(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttribute(const DOMString name);
```

Description

This is createAttribute, a member of class IDOMDocument.

83.2.5.2.2 IDOMDocument.createAttributeNS Method

Pascal

```
function createAttributeNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttributeNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createAttributeNS, a member of class IDOMDocument.

83.2.5.2.3 IDOMDocument.createCDATASection Method

Pascal

```
function createCDATASection(const data: DOMString): IDOMCDATASection; safecall;
```

C++

```
__fastcall IDOMCDATASection createCDATASection(const DOMString data);
```

Description

This is createCDATASection, a member of class IDOMDocument.

83.2.5.2.4 IDOMDocument.createComment Method

Pascal

```
function createComment(const data: DOMString): IDOMComment; safecall;
```

C++

```
__fastcall IDOMComment createComment(const DOMString data);
```

Description

This is createComment, a member of class IDOMDocument.

83.2.5.2.5 IDOMDocument.createDocumentFragment Method

Pascal

```
function createDocumentFragment: IDOMDocumentFragment; safecall;
```

C++

```
__fastcall IDOMDocumentFragment createDocumentFragment();
```

Description

This is createDocumentFragment, a member of class IDOMDocument.

83.2.5.2.6 IDOMDocument.createElement Method

Pascal

```
function createElement(const tagName: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElement(const DOMString tagName);
```

Description

This is createElement, a member of class IDOMDocument.

83.2.5.2.7 IDOMDocument.createElementNS Method

Pascal

```
function createElementNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElementNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createElementNS, a member of class IDOMDocument.

83.2.5.2.8 IDOMDocument.createEntityReference Method

Pascal

```
function createEntityReference(const name: DOMString): IDOMEntityReference; safecall;
```

C++

```
__fastcall IDOMEntityReference createEntityReference(const DOMString name);
```

Description

This is createEntityReference, a member of class IDOMDocument.

83.2.5.2.9 IDOMDocument.createProcessingInstruction Method

Pascal

```
function createProcessingInstruction(const target: DOMString; const data: DOMString):
IDOMProcessingInstruction; safecall;
```

C++

```
__fastcall IDOMProcessingInstruction createProcessingInstruction(const DOMString target, const
DOMString data);
```

Description

This is createProcessingInstruction, a member of class IDOMDocument.

83.2.5.2.10 IDOMDocument.createTextNode Method

Pascal

```
function createTextNode(const data: DOMString): IDOMText; safecall;
```

C++

```
__fastcall IDOMText createTextNode(const DOMString data);
```

Description

This is createTextNode, a member of class IDOMDocument.

83.2.5.2.11 IDOMDocument.get_doctype Method

Pascal

```
function get_doctype: IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType get_doctype();
```

Description

This is get_doctype, a member of class IDOMDocument.

83.2.5.2.12 IDOMDocument.get_documentElement Method

Pascal

```
function get_documentElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_documentElement();
```

Description

This is get_documentElement, a member of class IDOMDocument.

83.2.5.2.13 IDOMDocument.get_domImplementation Method

Pascal

```
function get_domImplementation: IDOMImplementation; safecall;
```

C++

```
__fastcall IDOMImplementation get_domImplementation();
```

Description

This is get_domImplementation, a member of class IDOMDocument.

83.2.5.2.14 IDOMDocument.getElementById Method

Pascal

```
function getElementById(const elementId: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement getElementById(const DOMString elementId);
```

Description

This is getElementById, a member of class IDOMDocument.

83.2.5.2.15 IDOMDocument.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const tagName: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString tagName);
```

Description

This is getElementsByTagName, a member of class IDOMDocument.

83.2.5.2.16 IDOMDocument.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMDocument.

83.2.5.2.17 IDOMDocument.importNode Method

Pascal

```
function importNode(importedNode: IDOMNode; deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode importNode(IDOMNode importedNode, WordBool deep);
```

Description

This is importNode, a member of class IDOMDocument.

83.2.5.2.18 IDOMDocument.set_documentElement Method

Pascal

```
procedure set_documentElement(const Element: IDOMElement); safecall;
```

C++

```
__fastcall set_documentElement(const IDOMElement Element);
```

Description

This is set_documentElement, a member of class IDOMDocument.

83.2.5.3 IDOMDocument Properties

The properties of the IDOMDocument class are listed here.

Public Properties

	Name	Description
	doctype (see page 7489)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 7489)	This is documentElement, a member of class IDOMDocument.
	domImplementation (see page 7489)	This is domImplementation, a member of class IDOMDocument.

83.2.5.3.1 IDOMDocument.doctype Property

Pascal

```
property doctype: IDOMDocumentType;
```

C++

```
__property IDOMDocumentType doctype;
```

Description

This is doctype, a member of class IDOMDocument.

83.2.5.3.2 IDOMDocument.documentElement Property

Pascal

```
property documentElement: IDOMElement;
```

C++

```
__property IDOMElement documentElement;
```

Description

This is documentElement, a member of class IDOMDocument.

83.2.5.3.3 IDOMDocument.domImplementation Property

Pascal

```
property domImplementation: IDOMImplementation;
```

C++

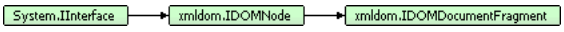
```
__property IDOMImplementation domImplementation;
```

Description

This is domImplementation, a member of class IDOMDocument.

83.2.6 IDOMDocumentFragment Interface

Class Hierarchy



Pascal

```
IDOMDocumentFragment = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentFragment : public IDOMNode;
```

File

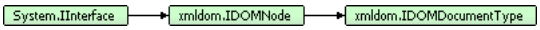
xmldom

Description

This is class xmldom.IDOMDocumentFragment.

83.2.7 IDOMDocumentType Interface

Class Hierarchy



Pascal

```
IDOMDocumentType = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentType : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocumentType.

83.2.7.1 IDOMDocumentType Members

The following tables list the members exposed by IDOMDocumentType.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_entities (see page 7492)	This is get_entities, a member of class IDOMDocumentType.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_internalSubset (see page 7492)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_name (see page 7492)	This is get_name, a member of class IDOMDocumentType.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_notations (see page 7492)	This is get_notations, a member of class IDOMDocumentType.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.

	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7492)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 7493)	This is get_systemId, a member of class IDOMDocumentType.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	entities (see page 7493)	This is entities, a member of class IDOMDocumentType.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	internalSubset (see page 7493)	This is internalSubset, a member of class IDOMDocumentType.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	name (see page 7494)	This is name, a member of class IDOMDocumentType.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	notations (see page 7494)	This is notations, a member of class IDOMDocumentType.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 7494)	This is publicId, a member of class IDOMDocumentType.
R	systemId (see page 7494)	This is systemId, a member of class IDOMDocumentType.

83.2.7.2 IDOMDocumentType Methods

The methods of the IDOMDocumentType class are listed here.

Public Methods

	Name	Description
	get_entities (see page 7492)	This is get_entities, a member of class IDOMDocumentType.
	get_internalSubset (see page 7492)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_name (see page 7492)	This is get_name, a member of class IDOMDocumentType.
	get_notations (see page 7492)	This is get_notations, a member of class IDOMDocumentType.
	get_publicId (see page 7492)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 7493)	This is get_systemId, a member of class IDOMDocumentType.

83.2.7.2.1 IDOMDocumentType.get_entities Method

Pascal

```
function get_entities: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_entities();
```

Description

This is get_entities, a member of class IDOMDocumentType.

83.2.7.2.2 IDOMDocumentType.get_internalSubset Method

Pascal

```
function get_internalSubset: DOMString; safecall;
```

C++

```
__fastcall DOMString get_internalSubset();
```

Description

This is get_internalSubset, a member of class IDOMDocumentType.

83.2.7.2.3 IDOMDocumentType.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMDocumentType.

83.2.7.2.4 IDOMDocumentType.get_notations Method

Pascal

```
function get_notations: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_notations();
```

Description

This is get_notations, a member of class IDOMDocumentType.

83.2.7.2.5 IDOMDocumentType.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMDocumentType.

83.2.7.2.6 IDOMDocumentType.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMDocumentType.

83.2.7.3 IDOMDocumentType Properties

The properties of the IDOMDocumentType class are listed here.

Public Properties

	Name	Description
	entities (see page 7493)	This is entities, a member of class IDOMDocumentType.
	internalSubset (see page 7493)	This is internalSubset, a member of class IDOMDocumentType.
	name (see page 7494)	This is name, a member of class IDOMDocumentType.
	notations (see page 7494)	This is notations, a member of class IDOMDocumentType.
	publicId (see page 7494)	This is publicId, a member of class IDOMDocumentType.
	systemId (see page 7494)	This is systemId, a member of class IDOMDocumentType.

83.2.7.3.1 IDOMDocumentType.entities Property

Pascal

```
property entities: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap entities;
```

Description

This is entities, a member of class IDOMDocumentType.

83.2.7.3.2 IDOMDocumentType.internalSubset Property

Pascal

```
property internalSubset: DOMString;
```

C++

```
__property DOMString internalSubset;
```

Description

This is internalSubset, a member of class IDOMDocumentType.

83.2.7.3.3 IDOMDocumentType.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMDocumentType.

83.2.7.3.4 IDOMDocumentType.notations Property

Pascal

```
property notations: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap notations;
```

Description

This is notations, a member of class IDOMDocumentType.

83.2.7.3.5 IDOMDocumentType.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMDocumentType.

83.2.7.3.6 IDOMDocumentType.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

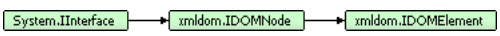
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMDocumentType.

83.2.8 IDOMElement Interface

Class Hierarchy



Pascal

```
IDOMElement = interface(IDOMNode);
```

C++

```
__interface IDOMElement : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMElement.

83.2.8.1 IDOMElement Members

The following tables list the members exposed by IDOMElement.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_tagName (see page 7497)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 7497)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 7497)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 7497)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 7497)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 7498)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 7498)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 7498)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 7498)	This is hasAttributeNS, a member of class IDOMElement.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeAttribute (see page 7499)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 7499)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 7499)	This is removeAttributeNS, a member of class IDOMElement.

	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	setAttribute (see page 7499)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 7499)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 7500)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 7500)	This is setAttributeNS, a member of class IDOMElement.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
R	tagName (see page 7500)	This is tagName, a member of class IDOMElement.

83.2.8.2 IDOMElement Methods

The methods of the IDOMElement class are listed here.

Public Methods

	Name	Description
	get_tagName (see page 7497)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 7497)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 7497)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 7497)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 7497)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 7498)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 7498)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 7498)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 7498)	This is hasAttributeNS, a member of class IDOMElement.
	removeAttribute (see page 7499)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 7499)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 7499)	This is removeAttributeNS, a member of class IDOMElement.
	setAttribute (see page 7499)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 7499)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 7500)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 7500)	This is setAttributeNS, a member of class IDOMElement.

83.2.8.2.1 IDOMElement.get_tagName Method

Pascal

```
function get_tagName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_tagName();
```

Description

This is get_tagName, a member of class IDOMElement.

83.2.8.2.2 IDOMElement.getAttribute Method

Pascal

```
function getAttribute(const name: DOMString): DOMString; safecall;
```

C++

```
__fastcall DOMString getAttribute(const DOMString name);
```

Description

This is getAttribute, a member of class IDOMElement.

83.2.8.2.3 IDOMElement.getAttributeNode Method

Pascal

```
function getAttributeNode(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNode(const DOMString name);
```

Description

This is getAttributeNode, a member of class IDOMElement.

83.2.8.2.4 IDOMElement.getAttributeNodeNS Method

Pascal

```
function getAttributeNodeNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNodeNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getAttributeNodeNS, a member of class IDOMElement.

83.2.8.2.5 IDOMElement.getAttributeNS Method

Pascal

```
function getAttributeNS(const namespaceURI: DOMString; const localName: DOMString): DOMString;
safecall;
```

C++

```
__fastcall DOMString getAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getAttributeNS, a member of class IDOMElement.

83.2.8.2.6 IDOMElement.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const name: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString name);
```

Description

This is getElementsByTagName, a member of class IDOMElement.

83.2.8.2.7 IDOMElement.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMElement.

83.2.8.2.8 IDOMElement.hasAttribute Method

Pascal

```
function hasAttribute(const name: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool hasAttribute(const DOMString name);
```

Description

This is hasAttribute, a member of class IDOMElement.

83.2.8.2.9 IDOMElement.hasAttributeNS Method

Pascal

```
function hasAttributeNS(const namespaceURI: DOMString; const localName: DOMString): WordBool;
safecall;
```

C++

```
__fastcall WordBool hasAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is hasAttributeNS, a member of class IDOMElement.

83.2.8.2.10 IDOMElement.removeAttribute Method

Pascal

```
procedure removeAttribute(const name: DOMString); safecall;
```

C++

```
__fastcall removeAttribute(const DOMString name);
```

Description

This is removeAttribute, a member of class IDOMElement.

83.2.8.2.11 IDOMElement.removeAttributeNode Method

Pascal

```
function removeAttributeNode(const oldAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr removeAttributeNode(const IDOMAttr oldAttr);
```

Description

This is removeAttributeNode, a member of class IDOMElement.

83.2.8.2.12 IDOMElement.removeAttributeNS Method

Pascal

```
procedure removeAttributeNS(const namespaceURI: DOMString; const localName: DOMString);
safecall;
```

C++

```
__fastcall removeAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeAttributeNS, a member of class IDOMElement.

83.2.8.2.13 IDOMElement.setAttribute Method

Pascal

```
procedure setAttribute(const name: DOMString; const value: DOMString); safecall;
```

C++

```
__fastcall setAttribute(const DOMString name, const DOMString value);
```

Description

This is setAttribute, a member of class IDOMElement.

83.2.8.2.14 IDOMElement.setAttributeNode Method

Pascal

```
function setAttributeNode(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNode(const IDOMAttr newAttr);
```

Description

This is setAttributeNode, a member of class IDOMElement.

83.2.8.2.15 IDOMElement.setAttributeNodeNS Method

Pascal

```
function setAttributeNodeNS(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNodeNS(const IDOMAttr newAttr);
```

Description

This is setAttributeNodeNS, a member of class IDOMElement.

83.2.8.2.16 IDOMElement.setAttributeNS Method

Pascal

```
procedure setAttributeNS(const namespaceURI: DOMString; const qulifiedName: DOMString; const
value: DOMString); safecall;
```

C++

```
__fastcall setAttributeNS(const DOMString namespaceURI, const DOMString qulifiedName, const
DOMString value);
```

Description

This is setAttributeNS, a member of class IDOMElement.

83.2.8.3 IDOMElement Properties

The properties of the IDOMElement class are listed here.

Public Properties

	Name	Description
	tagName (see page 7500)	This is tagName, a member of class IDOMElement.

83.2.8.3.1 IDOMElement.tagName Property

Pascal

```
property tagName: DOMString;
```

C++

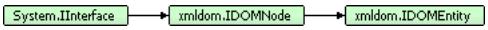
```
__property DOMString tagName;
```

Description

This is tagName, a member of class IDOMElement.

83.2.9 IDOMEntity Interface

Class Hierarchy



Pascal

```
IDOMEntity = interface(IDOMNode);
```

C++

```
__interface IDOMEntity : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMEntity.

83.2.9.1 IDOMEntity Members

The following tables list the members exposed by IDOMEntity.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_notationName (see page 7502)	This is get_notationName, a member of class IDOMEntity.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7502)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 7503)	This is get_systemId, a member of class IDOMEntity.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.

	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	notationName (see page 7503)	This is notationName, a member of class IDOMEntity.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 7503)	This is publicId, a member of class IDOMEntity.
R	systemId (see page 7503)	This is systemId, a member of class IDOMEntity.

83.2.9.2 IDOMEntity Methods

The methods of the IDOMEntity class are listed here.

Public Methods

	Name	Description
	get_notationName (see page 7502)	This is get_notationName, a member of class IDOMEntity.
	get_publicId (see page 7502)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 7503)	This is get_systemId, a member of class IDOMEntity.

83.2.9.2.1 IDOMEntity.get_notationName Method

Pascal

```
function get_notationName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_notationName();
```

Description

This is get_notationName, a member of class IDOMEntity.

83.2.9.2.2 IDOMEntity.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMEntity.

83.2.9.2.3 IDOMEntity.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMEntity.

83.2.9.3 IDOMEntity Properties

The properties of the IDOMEntity class are listed here.

Public Properties

	Name	Description
	notationName (see page 7503)	This is notationName, a member of class IDOMEntity.
	publicId (see page 7503)	This is publicId, a member of class IDOMEntity.
	systemId (see page 7503)	This is systemId, a member of class IDOMEntity.

83.2.9.3.1 IDOMEntity.notationName Property

Pascal

```
property notationName: DOMString;
```

C++

```
__property DOMString notationName;
```

Description

This is notationName, a member of class IDOMEntity.

83.2.9.3.2 IDOMEntity.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMEntity.

83.2.9.3.3 IDOMEntity.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

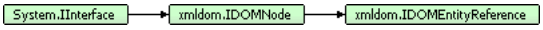
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMEntity.

83.2.10 IDOMEntityReference Interface

Class Hierarchy



Pascal

```
IDOMEntityReference = interface(IDOMNode);
```

C++

```
__interface IDOMEntityReference : public IDOMNode;
```

File

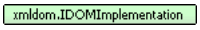
xmldom

Description

This is class xmldom.IDOMEntityReference.

83.2.11 IDOMImplementation Interface

Class Hierarchy



Pascal

```
IDOMImplementation = interface;
```

C++

```
__interface IDOMImplementation;
```

File

xmldom

Description

This is class xmldom.IDOMImplementation.

83.2.11.1 IDOMImplementation Members

The following tables list the members exposed by IDOMImplementation.

Public Methods

	Name	Description
	createDocument (see page 7505)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 7505)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 7505)	This is hasFeature, a member of class IDOMImplementation.

83.2.11.2 IDOMImplementation Methods

The methods of the IDOMImplementation class are listed here.

Public Methods

	Name	Description
	createDocument (see page 7505)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 7505)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 7505)	This is hasFeature, a member of class IDOMImplementation.

83.2.11.2.1 IDOMImplementation.createDocument Method

Pascal

```
function createDocument(const namespaceURI: DOMString; const qualifiedName: DOMString;
doctype: IDOMDocumentType): IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument createDocument(const DOMString namespaceURI, const DOMString
qualifiedName, IDOMDocumentType doctype);
```

Description

This is createDocument, a member of class IDOMImplementation.

83.2.11.2.2 IDOMImplementation.createDocumentType Method

Pascal

```
function createDocumentType(const qualifiedName: DOMString; const publicId: DOMString; const
systemId: DOMString): IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType createDocumentType(const DOMString qualifiedName, const DOMString
publicId, const DOMString systemId);
```

Description

This is createDocumentType, a member of class IDOMImplementation.

83.2.11.2.3 IDOMImplementation.hasFeature Method

Pascal

```
function hasFeature(const feature: DOMString; const version: DOMString): WordBool;
```

C++

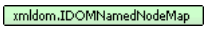
```
__fastcall WordBool hasFeature(const DOMString feature, const DOMString version);
```

Description

This is hasFeature, a member of class IDOMImplementation.

83.2.12 IDOMNamedNodeMap Interface

Class Hierarchy



Pascal

```
IDOMNamedNodeMap = interface;
```

C++

```
__interface IDOMNamedNodeMap;
```

File

xmldom

Description

This is class xmldom.IDOMNamedNodeMap.

83.2.12.1 IDOMNamedNodeMap Members

The following tables list the members exposed by IDOMNamedNodeMap.

Public Methods

	Name	Description
	get_item (see page 7507)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 7507)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 7507)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 7507)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 7508)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 7508)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.
	setNamedItem (see page 7508)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 7508)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

Public Properties

	Name	Description
R	item (see page 7509)	This is item, a member of class IDOMNamedNodeMap.
R	length (see page 7509)	This is length, a member of class IDOMNamedNodeMap.

83.2.12.2 IDOMNamedNodeMap Methods

The methods of the IDOMNamedNodeMap class are listed here.

Public Methods

	Name	Description
	get_item (see page 7507)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 7507)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 7507)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 7507)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 7508)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 7508)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

	setNamedItem (see page 7508)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 7508)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

83.2.12.2.1 IDOMNamedNodeMap.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNamedNodeMap.

83.2.12.2.2 IDOMNamedNodeMap.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNamedNodeMap.

83.2.12.2.3 IDOMNamedNodeMap.getNamedItem Method

Pascal

```
function getNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode getNamedItem(const DOMString name);
```

Description

This is getNamedItem, a member of class IDOMNamedNodeMap.

83.2.12.2.4 IDOMNamedNodeMap.getNamedItemNS Method

Pascal

```
function getNamedItemNS(const namespaceURI: DOMString; const localName: DOMString): IDOMNode;
safecall;
```

C++

```
__fastcall IDOMNode getNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getNamedItemNS, a member of class IDOMNamedNodeMap.

83.2.12.2.5 IDOMNamedNodeMap.removeNamedItem Method

Pascal

```
function removeNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItem(const DOMString name);
```

Description

This is removeNamedItem, a member of class IDOMNamedNodeMap.

83.2.12.2.6 IDOMNamedNodeMap.removeNamedItemNS Method

Pascal

```
function removeNamedItemNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

83.2.12.2.7 IDOMNamedNodeMap.setNamedItem Method

Pascal

```
function setNamedItem(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItem(const IDOMNode arg);
```

Description

This is setNamedItem, a member of class IDOMNamedNodeMap.

83.2.12.2.8 IDOMNamedNodeMap.setNamedItemNS Method

Pascal

```
function setNamedItemNS(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItemNS(const IDOMNode arg);
```

Description

This is setNamedItemNS, a member of class IDOMNamedNodeMap.

83.2.12.3 IDOMNamedNodeMap Properties

The properties of the IDOMNamedNodeMap class are listed here.

Public Properties

	Name	Description
	item (see page 7509)	This is item, a member of class IDOMNamedNodeMap.
	length (see page 7509)	This is length, a member of class IDOMNamedNodeMap.

83.2.12.3.1 IDOMNamedNodeMap.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNamedNodeMap.

83.2.12.3.2 IDOMNamedNodeMap.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

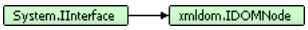
Description

This is length, a member of class IDOMNamedNodeMap.

83.2.13 IDOMNode Interface

IDOMNode is the DOM interface for a node in an XML document.

Class Hierarchy



Pascal

```
IDOMNode = interface(IInterface);
```

C++

```
__interface IDOMNode : public IInterface;
```

File

xmldom

Description

IDOMNode is a vendor-neutral version of the DOM interface for a node in an XML document. It conforms to the W3CXML DOM level 2 specification. Use IDOMNode to work with a node in an XML document when you are using standard DOM interfaces rather than the IXMLNode interface. (IXMLNode is the node interface used by TXMLDocument document.)

IDOMNode represents an element in an XML document that has been parsed by a DOM parser. It includes properties and methods for navigating the node hierarchy of an XML document, for accessing the attributes (see page 7520) of a node, and

for drilling down to obtain the node's value.

For more information about IDOMNode and the other standard DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

Note: C++ method declarations that use IDOMNode use the _di_IDOMNode type instead. This type is a DelphiInterface (see page 3263) wrapper around the IDOMNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMNode >  _di_IDOMNode;
```

See Also

IXMLNode (see page 7615)

TXMLDocument (see page 7299)

CreateDOMNode

83.2.13.1 IDOMNode Members

The following tables list the members exposed by IDOMNode.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.

	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
	localName (see page 7521)	This is localName, a member of class IDOMNode.
	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.

83.2.13.2 IDOMNode Methods

The methods of the IDOMNode class are listed here.

Public Methods

	Name	Description
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

83.2.13.2.1 IDOMNode.appendChild Method

Pascal

```
function appendChild(const newChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode appendChild(const IDOMNode newChild);
```

Description

This is appendChild, a member of class IDOMNode.

83.2.13.2.2 IDOMNode.cloneNode Method

Pascal

```
function cloneNode(deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode cloneNode(WordBool deep);
```

Description

This is cloneNode, a member of class IDOMNode.

83.2.13.2.3 IDOMNode.get_attributes Method

Returns the value of the get_attributes property.

Pascal

```
function get_attributes: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_attributes();
```

Description

get_get_attributes is the read implementation of the get_attributes property.

See Also

get_attributes

83.2.13.2.4 IDOMNode.get_childNodes Method

Returns the value of the get_childNodes property.

Pascal

```
function get_childNodes: IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList get_childNodes();
```

Description

get_get_childNodes is the read implementation of the get_childNodes property.

See Also

get_childNodes

83.2.13.2.5 IDOMNode.get_firstChild Method

Returns the value of the get_firstChild property.

Pascal

```
function get_firstChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_firstChild();
```

Description

get_get_firstChild is the read implementation of the get_firstChild property.

See Also

get_firstChild

83.2.13.2.6 IDOMNode.get_lastChild Method

Returns the value of the get_lastChild property.

Pascal

```
function get_lastChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_lastChild();
```

Description

get_get_lastChild is the read implementation of the get_lastChild property.

See Also

get_lastChild

83.2.13.2.7 IDOMNode.get_localName Method

Returns the value of the get_localName property.

Pascal

```
function get_localName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_localName();
```

Description

get_get_localName is the read implementation of the get_localName property.

See Also

get_localName

83.2.13.2.8 IDOMNode.get_namespaceURI Method

Returns the value of the get_namespaceURI property.

Pascal

```
function get_namespaceURI: DOMString; safecall;
```

C++

```
__fastcall DOMString get_namespaceURI();
```

Description

get_get_namespaceURI is the read implementation of the get_namespaceURI property.

See Also

get_namespaceURI

83.2.13.2.9 IDOMNode.get_nextSibling Method

Returns the value of the get_nextSibling property.

Pascal

```
function get_nextSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_nextSibling();
```

Description

get_get_nextSibling is the read implementation of the get_nextSibling property.

See Also

get_nextSibling

83.2.13.2.10 IDOMNode.get_nodeName Method

Returns the value of the get_nodeName property.

Pascal

```
function get_nodeName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeName();
```

Description

get_get_nodeName is the read implementation of the get_nodeName property.

See Also

get_nodeName

83.2.13.2.11 IDOMNode.get_nodeType Method

Returns the value of the get_nodeType property.

Pascal

```
function get_nodeType: DOMNodeType; safecall;
```

C++

```
__fastcall DOMNodeType get_nodeType();
```

Description

get_get_nodeType is the read implementation of the get_nodeType property.

See Also

get_nodeType

83.2.13.2.12 IDOMNode.get_nodeValue Method

Returns the value of the get_nodeValue property.

Pascal

```
function get_nodeValue: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeValue();
```

Description

get_get_nodeValue is the read implementation of the get_nodeValue property.

See Also

get_nodeValue

83.2.13.2.13 IDOMNode.get_ownerDocument Method

Returns the value of the get_ownerDocument property.

Pascal

```
function get_ownerDocument: IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument get_ownerDocument();
```

Description

get_get_ownerDocument is the read implementation of the get_ownerDocument property.

See Also

get_ownerDocument

83.2.13.2.14 IDOMNode.get_parentNode Method

Returns the value of the get_parentNode property.

Pascal

```
function get_parentNode: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_parentNode();
```

Description

get_get_parentNode is the read implementation of the get_parentNode property.

See Also

get_parentNode

83.2.13.2.15 IDOMNode.get_prefix Method

Returns the value of the get_prefix property.

Pascal

```
function get_prefix: DOMString; safecall;
```

C++

```
__fastcall DOMString get_prefix();
```

Description

get_get_prefix is the read implementation of the get_prefix property.

See Also

get_prefix

83.2.13.2.16 IDOMNode.get_previousSibling Method

Returns the value of the get_previousSibling property.

Pascal

```
function get_previousSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_previousSibling();
```

Description

get_get_previousSibling is the read implementation of the get_previousSibling property.

See Also

get_previousSibling

83.2.13.2.17 IDOMNode.hasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function hasChildNodes: WordBool; safecall;
```

C++

```
__fastcall WordBool hasChildNodes();
```

Description

hasChildNodes returns true if this node has any child nodes, false if it has no children.

See Also

ChildNodes (see page 7520)

FirstChild (see page 7520)

LastChild (see page 7520)

83.2.13.2.18 IDOMNode.insertBefore Method

Inserts a new child node before a specified child of the node.

Pascal

```
function insertBefore(const newChild: IDOMNode; const refChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode insertBefore(const IDOMNode newChild, const IDOMNode refChild);
```

Description

Use insertBefore to add a child node at a specified position in this node's list of children. If the new node is already in the document's node hierarchy, it is first removed from its parent node. If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 7521) that is not permitted as a child of this node, insertBefore raises an exception.

newChild is the child node to add. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 7521) DOCUMENT_FRAGMENT_NODE (see page 7570), the entire contents of the document fragment are moved into this node's childNodes (see page 7520) property list.

refChild is a child of this node before which to insert the new node. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

insertBefore returns the interface for the newly added child node.

See Also

AppendChild (see page 7511)

ChildNodes (see page 7520)

replaceChild (see page 7518)

removeChild (see page 7518)

NodeType (see page 7521)

OwnerDocument (see page 7522)

83.2.13.2.19 IDOMNode.normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure normalize; safecall;
```

C++

```
__fastcall normalize();
```

Description

Call normalize to consolidate adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes (see page 7520), only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes. This is useful for operations (such as XPointer lookups) that depend on a particular structure in the node hierarchy.

See Also

ChildNodes (see page 7520)

NodeType (see page 7521)

Attributes (see page 7520)

83.2.13.2.20 IDOMNode.removeChild Method

Removes a child node from this node.

Pascal

```
function removeChild(const childNode: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeChild(const IDOMNode childNode);
```

Description

removeChild removes a specified child node from this node, and returns its interface. If this node is read-only, or if the specified child node is not a child of this node, removeChild raises an exception.

childNode the node to remove.

See Also

AppendChild (see page 7511)

ChildNodes (see page 7520)

replaceChild (see page 7518)

83.2.13.2.21 IDOMNode.replaceChild Method

Replaces one of this node's children with a specified node.

Pascal

```
function replaceChild(const newChild: IDOMNode; const oldChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode replaceChild(const IDOMNode newChild, const IDOMNode oldChild);
```

Description

Use replaceChild to remove a child of this node and replace it with the node specified by newChild. If the new node is already in the document's node hierarchy, it is first removed from its parent node.

If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 7521) that is not permitted as a child of this node, replaceChild raises an exception.

newChild is the child node that replaces oldChild. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 7521) DOCUMENT_FRAGMENT_NODE (see page 7570), the entire contents of the document fragment are moved into this node's childNodes (see page 7520) property list.

refChild is a child that newChild replaces. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

replaceChild returns the interface for refChild (the replaced node).

See Also

AppendChild (see page 7511)

ChildNodes (see page 7520)

replaceChild

insertBefore (see page 7516)

83.2.13.2.22 IDOMNode.set_nodeValue Method

Sets the value of the node.

Pascal

```
procedure set_nodeValue(value: DOMString); safecall;
```

C++

```
__fastcall set_nodeValue(DOMString value);
```

Description

set_set_nodeValue is the write implementation of the set_nodeValue property.

value is the new value for the node.

See Also

set_nodeValue

83.2.13.2.23 IDOMNode.supports Method

Indicates whether the node supports a specified feature.

Pascal

```
function supports(const feature: DOMString; const version: DOMString): WordBool;
```

C++

```
__fastcall WordBool supports(const DOMString feature, const DOMString version);
```

Description

supports returns true if the DOM implementation implements the specified feature and this node supports that feature. Otherwise, supports returns false.

feature is the name of the feature in question.

version is the version of the feature. If no version is specified, supports returns true if any version of the feature is supported.

83.2.13.3 IDOMNode Properties

The properties of the IDOMNode class are listed here.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.

R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.

83.2.13.3.1 IDOMNode.attributes Property

Pascal

```
property attributes: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap attributes;
```

Description

This is attributes, a member of class IDOMNode.

83.2.13.3.2 IDOMNode.childNodes Property

Pascal

```
property childNodes: IDOMNodeList;
```

C++

```
__property IDOMNodeList childNodes;
```

Description

This is childNodes, a member of class IDOMNode.

83.2.13.3.3 IDOMNode.firstChild Property

Pascal

```
property firstChild: IDOMNode;
```

C++

```
__property IDOMNode firstChild;
```

Description

This is firstChild, a member of class IDOMNode.

83.2.13.3.4 IDOMNode.lastChild Property

Pascal

```
property lastChild: IDOMNode;
```

C++

```
__property IDOMNode lastChild;
```

Description

This is lastChild, a member of class IDOMNode.

83.2.13.3.5 IDOMNode.localName Property

Pascal

```
property localName: DOMString;
```

C++

```
__property DOMString localName;
```

Description

This is localName, a member of class IDOMNode.

83.2.13.3.6 IDOMNode.namespaceURI Property

Pascal

```
property namespaceURI: DOMString;
```

C++

```
__property DOMString namespaceURI;
```

Description

This is namespaceURI, a member of class IDOMNode.

83.2.13.3.7 IDOMNode.nextSibling Property

Pascal

```
property nextSibling: IDOMNode;
```

C++

```
__property IDOMNode nextSibling;
```

Description

This is nextSibling, a member of class IDOMNode.

83.2.13.3.8 IDOMNode.nodeName Property

Pascal

```
property nodeName: DOMString;
```

C++

```
__property DOMString nodeName;
```

Description

This is nodeName, a member of class IDOMNode.

83.2.13.3.9 IDOMNode.nodeType Property

Pascal

```
property nodeType: DOMNodeType;
```

C++

```
__property DOMNodeType nodeType;
```

Description

This is nodeType, a member of class IDOMNode.

83.2.13.3.10 IDOMNode.nodeValue Property

Pascal

```
property nodeValue: DOMString;
```

C++

```
__property DOMString nodeValue;
```

Description

This is nodeValue, a member of class IDOMNode.

83.2.13.3.11 IDOMNode.ownerDocument Property

Pascal

```
property ownerDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ownerDocument;
```

Description

This is ownerDocument, a member of class IDOMNode.

83.2.13.3.12 IDOMNode.parentNode Property

Pascal

```
property parentNode: IDOMNode;
```

C++

```
__property IDOMNode parentNode;
```

Description

This is parentNode, a member of class IDOMNode.

83.2.13.3.13 IDOMNode.prefix Property

Pascal

```
property prefix: DOMString;
```

C++

```
__property DOMString prefix;
```

Description

This is prefix, a member of class IDOMNode.

83.2.13.3.14 IDOMNode.previousSibling Property

Pascal

```
property previousSibling: IDOMNode;
```

C++

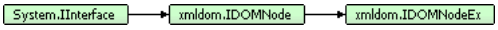
```
__property IDOMNode previousSibling;
```

Description

This is previousSibling, a member of class IDOMNode.

83.2.14 IDOMNodeEx Interface

Class Hierarchy



Pascal

```
IDOMNodeEx = interface(IDOMNode);
```

C++

```
__interface IDOMNodeEx : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNodeEx.

83.2.14.1 IDOMNodeEx Members

The following tables list the members exposed by IDOMNodeEx.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.

	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_text (see page 7524)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 7525)	This is get_xml, a member of class IDOMNodeEx.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7519)	Sets the value of the node.
	set_text (see page 7525)	This is set_text, a member of class IDOMNodeEx.
	supports (see page 7519)	Indicates whether the node supports a specified feature.
	transformNode (see page 7525)	This is the overview for the transformNode method overload.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
	text (see page 7526)	This is text, a member of class IDOMNodeEx.
R	xml (see page 7526)	This is xml, a member of class IDOMNodeEx.

83.2.14.2 IDOMNodeEx Methods

The methods of the IDOMNodeEx class are listed here.

Public Methods

	Name	Description
	get_text (see page 7524)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 7525)	This is get_xml, a member of class IDOMNodeEx.
	set_text (see page 7525)	This is set_text, a member of class IDOMNodeEx.
	transformNode (see page 7525)	This is the overview for the transformNode method overload.

83.2.14.2.1 IDOMNodeEx.get_text Method

Pascal

```
function get_text: DOMString; safecall;
```

C++

```
__fastcall DOMString get_text();
```

Description

This is get_text, a member of class IDOMNodeEx.

83.2.14.2.2 IDOMNodeEx.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMNodeEx.

83.2.14.2.3 IDOMNodeEx.set_text Method

Pascal

```
procedure set_text(const Value: DOMString); safecall;
```

C++

```
__fastcall set_text(const DOMString Value);
```

Description

This is set_text, a member of class IDOMNodeEx.

83.2.14.2.4 transformNode Method

This is the overview for the transformNode method overload.

Overload List

	Name	Description
	IDOMNodeEx.transformNode (IDOMNode, IDOMDocument) (see page 7525)	This is transformNode, a member of class IDOMNodeEx.
	IDOMNodeEx.transformNode (IDOMNode, WideString) (see page 7526)	This is transformNode, a member of class IDOMNodeEx.

83.2.14.2.4.1 IDOMNodeEx.transformNode Method (IDOMNode, IDOMDocument)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; const output: IDOMDocument); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, const IDOMDocument output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

83.2.14.2.4.2 IDOMNodeEx.transformNode Method (IDOMNode, WideString)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; var output: WideString); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, BSTR output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

83.2.14.3 IDOMNodeEx Properties

The properties of the IDOMNodeEx class are listed here.

Public Properties

	Name	Description
	text (see page 7526)	This is text, a member of class IDOMNodeEx.
R	xml (see page 7526)	This is xml, a member of class IDOMNodeEx.

83.2.14.3.1 IDOMNodeEx.text Property

Pascal

```
property text: DOMString;
```

C++

```
__property DOMString text;
```

Description

This is text, a member of class IDOMNodeEx.

83.2.14.3.2 IDOMNodeEx.xml Property

Pascal

```
property xml: DOMString;
```

C++

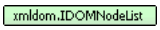
```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMNodeEx.

83.2.15 IDOMNodeList Interface

Class Hierarchy



Pascal

```
IDOMNodeList = interface;
```

C++

```
__interface IDOMNodeList;
```

File

xmldom

Description

This is class xmldom.IDOMNodeList.

83.2.15.1 IDOMNodeList Members

The following tables list the members exposed by IDOMNodeList.

Public Methods

	Name	Description
	get_item (see page 7527)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 7527)	This is get_length, a member of class IDOMNodeList.

Public Properties

	Name	Description
R	item (see page 7528)	This is item, a member of class IDOMNodeList.
R	length (see page 7528)	This is length, a member of class IDOMNodeList.

83.2.15.2 IDOMNodeList Methods

The methods of the IDOMNodeList class are listed here.

Public Methods

	Name	Description
	get_item (see page 7527)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 7527)	This is get_length, a member of class IDOMNodeList.

83.2.15.2.1 IDOMNodeList.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNodeList.

83.2.15.2.2 IDOMNodeList.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNodeList.

83.2.15.3 IDOMNodeList Properties

The properties of the IDOMNodeList class are listed here.

Public Properties

	Name	Description
	item (see page 7528)	This is item, a member of class IDOMNodeList.
	length (see page 7528)	This is length, a member of class IDOMNodeList.

83.2.15.3.1 IDOMNodeList.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNodeList.

83.2.15.3.2 IDOMNodeList.length Property

Pascal

```
property length: Integer;
```

C++

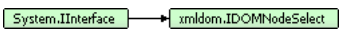
```
__property int length;
```

Description

This is length, a member of class IDOMNodeList.

83.2.16 IDOMNodeSelect Interface

Class Hierarchy



Pascal

```
IDOMNodeSelect = interface(IInterface);
```

C++

```
__interface IDOMNodeSelect : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMNodeSelect.

83.2.16.1 IDOMNodeSelect Members

The following tables list the members exposed by IDOMNodeSelect.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	selectNode (see page 7529)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 7529)	This is selectNodes, a member of class IDOMNodeSelect.

83.2.16.2 IDOMNodeSelect Methods

The methods of the IDOMNodeSelect class are listed here.

Public Methods

	Name	Description
	selectNode (see page 7529)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 7529)	This is selectNodes, a member of class IDOMNodeSelect.

83.2.16.2.1 IDOMNodeSelect.selectNode Method

Pascal

```
function selectNode(const nodePath: WideString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode selectNode(const BSTR nodePath);
```

Description

This is selectNode, a member of class IDOMNodeSelect.

83.2.16.2.2 IDOMNodeSelect.selectNodes Method

Pascal

```
function selectNodes(const nodePath: WideString): IDOMNodeList; safecall;
```

C++

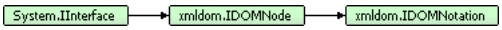
```
__fastcall IDOMNodeList selectNodes(const BSTR nodePath);
```

Description

This is selectNodes, a member of class IDOMNodeSelect.

83.2.17 IDOMNotation Interface

Class Hierarchy



Pascal

```
IDOMNotation = interface(IDOMNode);
```

C++

```
__interface IDOMNotation : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNotation.

83.2.17.1 IDOMNotation Members

The following tables list the members exposed by IDOMNotation.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7531)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 7531)	This is get_systemId, a member of class IDOMNotation.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.

	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 7532)	This is publicId, a member of class IDOMNotation.
R	systemId (see page 7532)	This is systemId, a member of class IDOMNotation.

83.2.17.2 IDOMNotation Methods

The methods of the IDOMNotation class are listed here.

Public Methods

	Name	Description
	get_publicId (see page 7531)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 7531)	This is get_systemId, a member of class IDOMNotation.

83.2.17.2.1 IDOMNotation.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMNotation.

83.2.17.2.2 IDOMNotation.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMNotation.

83.2.17.3 IDOMNotation Properties

The properties of the IDOMNotation class are listed here.

Public Properties

	Name	Description
	publicId (see page 7532)	This is publicId, a member of class IDOMNotation.
	systemId (see page 7532)	This is systemId, a member of class IDOMNotation.

83.2.17.3.1 IDOMNotation.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMNotation.

83.2.17.3.2 IDOMNotation.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

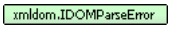
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMNotation.

83.2.18 IDOMParseError Interface

Class Hierarchy



Pascal

```
IDOMParseError = interface;
```

C++

```
__interface IDOMParseError;
```

File

xmldom

Description

This is class xmldom.IDOMParseError.

83.2.18.1 IDOMParseError Members

The following tables list the members exposed by IDOMParseError.

Public Methods

	Name	Description
	get_errorCode (see page 7533)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 7534)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 7534)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 7534)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 7534)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 7534)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 7535)	This is get_url, a member of class IDOMParseError.

Public Properties

	Name	Description
R	errorCode (see page 7535)	This is errorCode, a member of class IDOMParseError.
R	filePos (see page 7535)	This is filePos, a member of class IDOMParseError.
R	line (see page 7536)	This is line, a member of class IDOMParseError.
R	linePos (see page 7536)	This is linePos, a member of class IDOMParseError.
R	reason (see page 7536)	This is reason, a member of class IDOMParseError.
R	srcText (see page 7536)	This is srcText, a member of class IDOMParseError.
R	url (see page 7536)	This is url, a member of class IDOMParseError.

83.2.18.2 IDOMParseError Methods

The methods of the IDOMParseError class are listed here.

Public Methods

	Name	Description
	get_errorCode (see page 7533)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 7534)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 7534)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 7534)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 7534)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 7534)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 7535)	This is get_url, a member of class IDOMParseError.

83.2.18.2.1 IDOMParseError.get_errorCode Method

Pascal

```
function get_errorCode: Integer; safecall;
```

C++

```
__fastcall int get_errorCode();
```

Description

This is get_errorCode, a member of class IDOMParseError.

83.2.18.2.2 IDOMParseError.get_filepos Method

Pascal

```
function get_filepos: Integer; safecall;
```

C++

```
__fastcall int get_filepos();
```

Description

This is get_filepos, a member of class IDOMParseError.

83.2.18.2.3 IDOMParseError.get_line Method

Pascal

```
function get_line: Integer; safecall;
```

C++

```
__fastcall int get_line();
```

Description

This is get_line, a member of class IDOMParseError.

83.2.18.2.4 IDOMParseError.get_linepos Method

Pascal

```
function get_linepos: Integer; safecall;
```

C++

```
__fastcall int get_linepos();
```

Description

This is get_linepos, a member of class IDOMParseError.

83.2.18.2.5 IDOMParseError.get_reason Method

Pascal

```
function get_reason: DOMString; safecall;
```

C++

```
__fastcall DOMString get_reason();
```

Description

This is get_reason, a member of class IDOMParseError.

83.2.18.2.6 IDOMParseError.get_srcText Method

Pascal

```
function get_srcText: DOMString; safecall;
```

C++

```
__fastcall DOMString get_srcText();
```

Description

This is get_srcText, a member of class IDOMParseError.

83.2.18.2.7 IDOMParseError.get_url Method

Pascal

```
function get_url: DOMString; safecall;
```

C++

```
__fastcall DOMString get_url();
```

Description

This is get_url, a member of class IDOMParseError.

83.2.18.3 IDOMParseError Properties

The properties of the IDOMParseError class are listed here.

Public Properties

	Name	Description
	errorCode (see page 7535)	This is errorCode, a member of class IDOMParseError.
	filePos (see page 7535)	This is filePos, a member of class IDOMParseError.
	line (see page 7536)	This is line, a member of class IDOMParseError.
	linePos (see page 7536)	This is linePos, a member of class IDOMParseError.
	reason (see page 7536)	This is reason, a member of class IDOMParseError.
	srcText (see page 7536)	This is srcText, a member of class IDOMParseError.
	url (see page 7536)	This is url, a member of class IDOMParseError.

83.2.18.3.1 IDOMParseError.errorCode Property

Pascal

```
property errorCode: Integer;
```

C++

```
__property int errorCode;
```

Description

This is errorCode, a member of class IDOMParseError.

83.2.18.3.2 IDOMParseError.filePos Property

Pascal

```
property filePos: Integer;
```

C++

```
__property int filePos;
```

Description

This is filePos, a member of class IDOMParseError.

83.2.18.3.3 IDOMParseError.line Property

Pascal

```
property line: Integer;
```

C++

```
__property int line;
```

Description

This is line, a member of class IDOMParseError.

83.2.18.3.4 IDOMParseError.linePos Property

Pascal

```
property linePos: Integer;
```

C++

```
__property int linePos;
```

Description

This is linePos, a member of class IDOMParseError.

83.2.18.3.5 IDOMParseError.reason Property

Pascal

```
property reason: DOMString;
```

C++

```
__property DOMString reason;
```

Description

This is reason, a member of class IDOMParseError.

83.2.18.3.6 IDOMParseError.srcText Property

Pascal

```
property srcText: DOMString;
```

C++

```
__property DOMString srcText;
```

Description

This is srcText, a member of class IDOMParseError.

83.2.18.3.7 IDOMParseError.url Property

Pascal

```
property url: DOMString;
```

C++

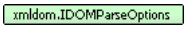
```
__property DOMString url;
```

Description

This is url, a member of class IDOMParseError.

83.2.19 IDOMParseOptions Interface

Class Hierarchy



Pascal

```
IDOMParseOptions = interface;
```

C++

```
__interface IDOMParseOptions;
```

File

xmldom

Description

This is class xmldom.IDOMParseOptions.

83.2.19.1 IDOMParseOptions Members

The following tables list the members exposed by IDOMParseOptions.

Public Methods

	Name	Description
	get_async (see page 7538)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 7538)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 7538)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 7538)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 7539)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 7539)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 7539)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 7539)	This is set_validate, a member of class IDOMParseOptions.

Public Properties

	Name	Description
	async (see page 7540)	This is async, a member of class IDOMParseOptions.
	preserveWhiteSpace (see page 7540)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 7540)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 7540)	This is validate, a member of class IDOMParseOptions.

83.2.19.2 IDOMParseOptions Methods

The methods of the IDOMParseOptions class are listed here.

Public Methods

	Name	Description
	get_async (see page 7538)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 7538)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 7538)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 7538)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 7539)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 7539)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 7539)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 7539)	This is set_validate, a member of class IDOMParseOptions.

83.2.19.2.1 IDOMParseOptions.get_async Method

Pascal

```
function get_async: Boolean; safecall;
```

C++

```
__fastcall Boolean get_async();
```

Description

This is get_async, a member of class IDOMParseOptions.

83.2.19.2.2 IDOMParseOptions.get_preserveWhiteSpace Method

Pascal

```
function get_preserveWhiteSpace: Boolean; safecall;
```

C++

```
__fastcall Boolean get_preserveWhiteSpace();
```

Description

This is get_preserveWhiteSpace, a member of class IDOMParseOptions.

83.2.19.2.3 IDOMParseOptions.get_resolveExternals Method

Pascal

```
function get_resolveExternals: Boolean; safecall;
```

C++

```
__fastcall Boolean get_resolveExternals();
```

Description

This is get_resolveExternals, a member of class IDOMParseOptions.

83.2.19.2.4 IDOMParseOptions.get_validate Method

Pascal

```
function get_validate: Boolean; safecall;
```

C++

```
__fastcall Boolean get_validate();
```

Description

This is get_validate, a member of class IDOMParseOptions.

83.2.19.2.5 IDOMParseOptions.set_async Method

Pascal

```
procedure set_async(Value: Boolean); safecall;
```

C++

```
__fastcall set_async(Boolean Value);
```

Description

This is set_async, a member of class IDOMParseOptions.

83.2.19.2.6 IDOMParseOptions.set_preserveWhiteSpace Method

Pascal

```
procedure set_preserveWhiteSpace(Value: Boolean); safecall;
```

C++

```
__fastcall set_preserveWhiteSpace(Boolean Value);
```

Description

This is set_preserveWhiteSpace, a member of class IDOMParseOptions.

83.2.19.2.7 IDOMParseOptions.set_resolveExternals Method

Pascal

```
procedure set_resolveExternals(Value: Boolean); safecall;
```

C++

```
__fastcall set_resolveExternals(Boolean Value);
```

Description

This is set_resolveExternals, a member of class IDOMParseOptions.

83.2.19.2.8 IDOMParseOptions.set_validate Method

Pascal

```
procedure set_validate(Value: Boolean); safecall;
```

C++

```
__fastcall set_validate(Boolean Value);
```

Description

This is set_validate, a member of class IDOMParseOptions.

83.2.19.3 IDOMParseOptions Properties

The properties of the IDOMParseOptions class are listed here.

Public Properties

	Name	Description
	async (see page 7540)	This is async, a member of class IDOMParseOptions.
	preserveWhiteSpace (see page 7540)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 7540)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 7540)	This is validate, a member of class IDOMParseOptions.

83.2.19.3.1 IDOMParseOptions.async Property

Pascal

```
property async: Boolean;
```

C++

```
__property Boolean async;
```

Description

This is async, a member of class IDOMParseOptions.

83.2.19.3.2 IDOMParseOptions.preserveWhiteSpace Property

Pascal

```
property preserveWhiteSpace: Boolean;
```

C++

```
__property Boolean preserveWhiteSpace;
```

Description

This is preserveWhiteSpace, a member of class IDOMParseOptions.

83.2.19.3.3 IDOMParseOptions.resolveExternals Property

Pascal

```
property resolveExternals: Boolean;
```

C++

```
__property Boolean resolveExternals;
```

Description

This is resolveExternals, a member of class IDOMParseOptions.

83.2.19.3.4 IDOMParseOptions.validate Property

Pascal

```
property validate: Boolean;
```

C++

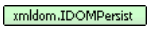
```
__property Boolean validate;
```

Description

This is validate, a member of class IDOMParseOptions.

83.2.20 IDOMPersist Interface

Class Hierarchy



Pascal

```
IDOMPersist = interface;
```

C++

```
__interface IDOMPersist;
```

File

xmldom

Description

This is class xmldom.IDOMPersist.

83.2.20.1 IDOMPersist Members

The following tables list the members exposed by IDOMPersist.

Public Methods

	Name	Description
	asyncLoadState (see page 7542)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 7542)	This is get_xml, a member of class IDOMPersist.
	load (see page 7542)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 7542)	This is the overview for the loadFromStream method overload.
	loadxml (see page 7543)	This is loadxml, a member of class IDOMPersist.
	save (see page 7543)	This is save, a member of class IDOMPersist.
	saveToStream (see page 7543)	This is the overview for the saveToStream method overload.
	set_OnAsyncLoad (see page 7544)	This is set_OnAsyncLoad, a member of class IDOMPersist.

Public Properties

	Name	Description
	xml (see page 7544)	This is xml, a member of class IDOMPersist.

83.2.20.2 IDOMPersist Methods

The methods of the IDOMPersist class are listed here.

Public Methods

	Name	Description
	asyncLoadState (see page 7542)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 7542)	This is get_xml, a member of class IDOMPersist.
	load (see page 7542)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 7542)	This is the overview for the loadFromStream method overload.
	loadxml (see page 7543)	This is loadxml, a member of class IDOMPersist.
	save (see page 7543)	This is save, a member of class IDOMPersist.
	saveToStream (see page 7543)	This is the overview for the saveToStream method overload.

	set_OnAsyncLoad (see page 7544)	This is set_OnAsyncLoad, a member of class IDOMPersist.

83.2.20.2.1 IDOMPersist.asyncLoadState Method

Pascal

```
function asyncLoadState: Integer; safecall;
```

C++

```
__fastcall int asyncLoadState();
```

Description

This is asyncLoadState, a member of class IDOMPersist.

83.2.20.2.2 IDOMPersist.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMPersist.

83.2.20.2.3 IDOMPersist.load Method

Pascal

```
function load(source: OleVariant): WordBool; safecall;
```

C++

```
__fastcall WordBool load(OleVariant source);
```

Description

This is load, a member of class IDOMPersist.

83.2.20.2.4 loadFromStream Method

This is the overview for the loadFromStream method overload.

Overload List

	Name	Description
	IDOMPersist.loadFromStream (IStream) (see page 7542)	This is loadFromStream, a member of class IDOMPersist.
	IDOMPersist.loadFromStream (TStream) (see page 7543)	This is loadFromStream, a member of class IDOMPersist.

83.2.20.2.4.1 IDOMPersist.loadFromStream Method (IStream)

Pascal

```
function loadFromStream(const stream: IStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const IStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

83.2.20.2.4.2 IDOMPersist.loadFromStream Method (TStream)

Pascal

```
function loadFromStream(const stream: TStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const TStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

83.2.20.2.5 IDOMPersist.loadxml Method

Pascal

```
function loadxml(const Value: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool loadxml(const DOMString Value);
```

Description

This is loadxml, a member of class IDOMPersist.

83.2.20.2.6 IDOMPersist.save Method

Pascal

```
procedure save(destination: OleVariant); safecall;
```

C++

```
__fastcall save(OleVariant destination);
```

Description

This is save, a member of class IDOMPersist.

83.2.20.2.7 saveToStream Method

This is the overview for the saveToStream method overload.

Overload List

	Name	Description
	IDOMPersist.saveToStream (IStream) (see page 7544)	This is saveToStream, a member of class IDOMPersist.
	IDOMPersist.saveToStream (TStream) (see page 7544)	This is saveToStream, a member of class IDOMPersist.

83.2.20.2.7.1 IDOMPersist.saveToStream Method (IStream)

Pascal

```
procedure saveToStream(const stream: IStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const IStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

83.2.20.2.7.2 IDOMPersist.saveToStream Method (TStream)

Pascal

```
procedure saveToStream(const stream: TStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const TStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

83.2.20.2.8 IDOMPersist.set_OnAsyncLoad Method

Pascal

```
procedure set_OnAsyncLoad(const Sender: TObject; EventHandler: TAsyncEventHandler); safecall;
```

C++

```
__fastcall set_OnAsyncLoad(const TObject * Sender, TAsyncEventHandler EventHandler);
```

Description

This is set_OnAsyncLoad, a member of class IDOMPersist.

83.2.20.3 IDOMPersist Properties

The properties of the IDOMPersist class are listed here.

Public Properties

	Name	Description
	xml (see page 7544)	This is xml, a member of class IDOMPersist.

83.2.20.3.1 IDOMPersist.xml Property

Pascal

```
property xml: DOMString;
```

C++

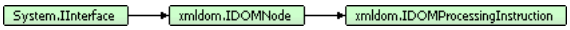
```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMPersist.

83.2.21 IDOMProcessingInstruction Interface

Class Hierarchy



Pascal

```
IDOMProcessingInstruction = interface(IDOMNode);
```

C++

```
__interface IDOMProcessingInstruction : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMProcessingInstruction.

83.2.21.1 IDOMProcessingInstruction Members

The following tables list the members exposed by IDOMProcessingInstruction.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_data (see page 7546)	This is get_data, a member of class IDOMProcessingInstruction.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	get_target (see page 7546)	This is get_target, a member of class IDOMProcessingInstruction.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.

	set_data (see page 7547)	This is set_data, a member of class IDOMProcessingInstruction.
	set_nodeValue (see page 7519)	Sets the value of the node.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
	data (see page 7547)	This is data, a member of class IDOMProcessingInstruction.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.
R	target (see page 7547)	This is target, a member of class IDOMProcessingInstruction.

83.2.21.2 IDOMProcessingInstruction Methods

The methods of the IDOMProcessingInstruction class are listed here.

Public Methods

	Name	Description
	get_data (see page 7546)	This is get_data, a member of class IDOMProcessingInstruction.
	get_target (see page 7546)	This is get_target, a member of class IDOMProcessingInstruction.
	set_data (see page 7547)	This is set_data, a member of class IDOMProcessingInstruction.

83.2.21.2.1 IDOMProcessingInstruction.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMProcessingInstruction.

83.2.21.2.2 IDOMProcessingInstruction.get_target Method

Pascal

```
function get_target: DOMString; safecall;
```

C++

```
__fastcall DOMString get_target();
```

Description

This is get_target, a member of class IDOMProcessingInstruction.

83.2.21.2.3 IDOMProcessingInstruction.set_data Method

Pascal

```
procedure set_data(const value: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString value);
```

Description

This is set_data, a member of class IDOMProcessingInstruction.

83.2.21.3 IDOMProcessingInstruction Properties

The properties of the IDOMProcessingInstruction class are listed here.

Public Properties

	Name	Description
	data (see page 7547)	This is data, a member of class IDOMProcessingInstruction.
R	target (see page 7547)	This is target, a member of class IDOMProcessingInstruction.

83.2.21.3.1 IDOMProcessingInstruction.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMProcessingInstruction.

83.2.21.3.2 IDOMProcessingInstruction.target Property

Pascal

```
property target: DOMString;
```

C++

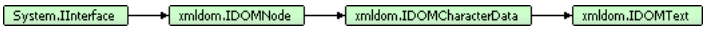
```
__property DOMString target;
```

Description

This is target, a member of class IDOMProcessingInstruction.

83.2.22 IDOMText Interface

Class Hierarchy



Pascal

```
IDOMText = interface(IDOMCharacterData);
```

C++

```
__interface IDOMText : public IDOMCharacterData;
```

File

xmldom

Description

This is class xmldom.IDOMText.

83.2.22.1 IDOMText Members

The following tables list the members exposed by IDOMText.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	appendChild (see page 7511)	This is appendChild, a member of class IDOMNode.
	appendData (see page 7479)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 7512)	This is cloneNode, a member of class IDOMNode.
	deleteData (see page 7480)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 7512)	Returns the value of the get_attributes property.
	get_childNodes (see page 7512)	Returns the value of the get_childNodes property.
	get_data (see page 7480)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 7512)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7513)	Returns the value of the get_lastChild property.
	get_length (see page 7480)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 7513)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7513)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7514)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7514)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7514)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7515)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7515)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7515)	Returns the value of the get_parentNode property.
	get_prefix (see page 7515)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7516)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7516)	Indicates whether the node has any child nodes.
	insertBefore (see page 7516)	Inserts a new child node before a specified child of the node.
	insertData (see page 7480)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 7517)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7518)	Removes a child node from this node.
	replaceChild (see page 7518)	Replaces one of this node's children with a specified node.
	replaceData (see page 7480)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7481)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 7519)	Sets the value of the node.
	splitText (see page 7549)	This is splitText, a member of class IDOMText.
	substringData (see page 7481)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 7519)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7520)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 7520)	This is childNodes, a member of class IDOMNode.
	data (see page 7481)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 7520)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 7520)	This is lastChild, a member of class IDOMNode.
R	length (see page 7481)	This is length, a member of class IDOMCharacterData.
R	localName (see page 7521)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 7521)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 7521)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 7521)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 7521)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 7522)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 7522)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 7522)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 7522)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 7523)	This is previousSibling, a member of class IDOMNode.

83.2.22.2 IDOMText Methods

The methods of the IDOMText class are listed here.

Public Methods

	Name	Description
	splitText (see page 7549)	This is splitText, a member of class IDOMText.

83.2.22.2.1 IDOMText.splitText Method

Pascal

```
function splitText(offset: Integer): IDOMText; safecall;
```

C++

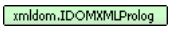
```
__fastcall IDOMText splitText(int offset);
```

Description

This is splitText, a member of class IDOMText.

83.2.23 IDOMXMLProlog Interface

Class Hierarchy



Pascal

```
IDOMXMLProlog = interface;
```

C++

```
__interface IDOMXMLProlog;
```

File

xmldom

Description

This is class xmldom.IDOMXMLProlog.

83.2.23.1 IDOMXMLProlog Members

The following tables list the members exposed by IDOMXMLProlog.

Public Methods

	Name	Description
	get_Encoding (see page 7551)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 7551)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 7551)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 7551)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 7551)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 7552)	This is set_Version, a member of class IDOMXMLProlog.

Public Properties

	Name	Description
	Encoding (see page 7552)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 7552)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 7552)	This is Version, a member of class IDOMXMLProlog.

83.2.23.2 IDOMXMLProlog Methods

The methods of the IDOMXMLProlog class are listed here.

Public Methods

	Name	Description
	get_Encoding (see page 7551)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 7551)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 7551)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 7551)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 7551)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 7552)	This is set_Version, a member of class IDOMXMLProlog.

83.2.23.2.1 IDOMXMLProlog.get_Encoding Method

Pascal

```
function get_Encoding: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Encoding();
```

Description

This is get_Encoding, a member of class IDOMXMLProlog.

83.2.23.2.2 IDOMXMLProlog.get_Standalone Method

Pascal

```
function get_Standalone: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Standalone();
```

Description

This is get_Standalone, a member of class IDOMXMLProlog.

83.2.23.2.3 IDOMXMLProlog.get_Version Method

Pascal

```
function get_Version: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Version();
```

Description

This is get_Version, a member of class IDOMXMLProlog.

83.2.23.2.4 IDOMXMLProlog.set_Encoding Method

Pascal

```
procedure set_Encoding(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Encoding(const DOMString Value);
```

Description

This is set_Encoding, a member of class IDOMXMLProlog.

83.2.23.2.5 IDOMXMLProlog.set_Standalone Method

Pascal

```
procedure set_Standalone(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Standalone(const DOMString Value);
```

Description

This is set_Standalone, a member of class IDOMXMLProlog.

83.2.23.2.6 IDOMXMLProlog.set_Version Method

Pascal

```
procedure set_Version(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Version(const DOMString Value);
```

Description

This is set_Version, a member of class IDOMXMLProlog.

83.2.23.3 IDOMXMLProlog Properties

The properties of the IDOMXMLProlog class are listed here.

Public Properties

	Name	Description
	Encoding (see page 7552)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 7552)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 7552)	This is Version, a member of class IDOMXMLProlog.

83.2.23.3.1 IDOMXMLProlog.Encoding Property

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

This is Encoding, a member of class IDOMXMLProlog.

83.2.23.3.2 IDOMXMLProlog.Standalone Property

Pascal

```
property Standalone: DOMString;
```

C++

```
__property DOMString Standalone;
```

Description

This is Standalone, a member of class IDOMXMLProlog.

83.2.23.3.3 IDOMXMLProlog.Version Property

Pascal

```
property Version: DOMString;
```

C++

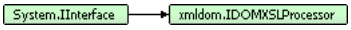
```
__property DOMString Version;
```

Description

This is Version, a member of class IDOMXMLProlog.

83.2.24 IDOMXSLProcessor Interface

Class Hierarchy



Pascal

```
IDOMXSLProcessor = interface(IInterface);
```

C++

```
__interface IDOMXSLProcessor : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMXSLProcessor.

83.2.24.1 IDOMXSLProcessor Members

The following tables list the members exposed by IDOMXSLProcessor.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	Get_input (see page 7554)	This is Get_input, a member of class IDOMXSLProcessor.
	Get_output (see page 7554)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 7554)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	reset (see page 7555)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 7555)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 7555)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 7555)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 7555)	This is transform, a member of class IDOMXSLProcessor.

Public Properties

	Name	Description
	input (see page 7556)	This is input, a member of class IDOMXSLProcessor.
	output (see page 7556)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 7556)	This is stylesheet, a member of class IDOMXSLProcessor.

83.2.24.2 IDOMXSLProcessor Methods

The methods of the IDOMXSLProcessor class are listed here.

Public Methods

	Name	Description
	Get_input (see page 7554)	This is Get_input, a member of class IDOMXSLProcessor.
	Get_output (see page 7554)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 7554)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	reset (see page 7555)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 7555)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 7555)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 7555)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 7555)	This is transform, a member of class IDOMXSLProcessor.

83.2.24.2.1 IDOMXSLProcessor.Get_input Method

Pascal

```
function Get_input: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_input();
```

Description

This is Get_input, a member of class IDOMXSLProcessor.

83.2.24.2.2 IDOMXSLProcessor.Get_output Method

Pascal

```
function Get_output: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_output();
```

Description

This is Get_output, a member of class IDOMXSLProcessor.

83.2.24.2.3 IDOMXSLProcessor.Get_stylesheet Method

Pascal

```
function Get_stylesheet: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode Get_stylesheet();
```

Description

This is Get_stylesheet, a member of class IDOMXSLProcessor.

83.2.24.2.4 IDOMXSLProcessor.reset Method

Pascal

```
procedure reset; safecall;
```

C++

```
__fastcall reset();
```

Description

This is reset, a member of class IDOMXSLProcessor.

83.2.24.2.5 IDOMXSLProcessor.Set_input Method

Pascal

```
procedure Set_input(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_input(const OleVariant value);
```

Description

This is Set_input, a member of class IDOMXSLProcessor.

83.2.24.2.6 IDOMXSLProcessor.Set_output Method

Pascal

```
procedure Set_output(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_output(const OleVariant value);
```

Description

This is Set_output, a member of class IDOMXSLProcessor.

83.2.24.2.7 IDOMXSLProcessor.setParameter Method

Pascal

```
procedure setParameter(const Name: DOMString; Value: OleVariant; const namespaceURI:
DOMString); safecall;
```

C++

```
__fastcall setParameter(const DOMString Name, OleVariant Value, const DOMString namespaceURI);
```

Description

This is setParameter, a member of class IDOMXSLProcessor.

83.2.24.2.8 IDOMXSLProcessor.transform Method

Pascal

```
function transform: WordBool; safecall;
```

C++

```
__fastcall WordBool transform();
```

Description

This is transform, a member of class IDOMXSLProcessor.

83.2.24.3 IDOMXSLProcessor Properties

The properties of the IDOMXSLProcessor class are listed here.

Public Properties

	Name	Description
	input (see page 7556)	This is input, a member of class IDOMXSLProcessor.
	output (see page 7556)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 7556)	This is stylesheet, a member of class IDOMXSLProcessor.

83.2.24.3.1 IDOMXSLProcessor.input Property

Pascal

```
property input: OleVariant;
```

C++

```
__property OleVariant input;
```

Description

This is input, a member of class IDOMXSLProcessor.

83.2.24.3.2 IDOMXSLProcessor.output Property

Pascal

```
property output: OleVariant;
```

C++

```
__property OleVariant output;
```

Description

This is output, a member of class IDOMXSLProcessor.

83.2.24.3.3 IDOMXSLProcessor.stylesheet Property

Pascal

```
property stylesheet: IDOMNode;
```

C++

```
__property IDOMNode stylesheet;
```

Description

This is stylesheet, a member of class IDOMXSLProcessor.

83.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DOMVendorNotSupported (see page 7557)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 7557)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 7558)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 7559)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 7559)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 7560)	Returns the TDOMVendor (see page 7466) object for a specified DOM vendor.
	IsPrefixed (see page 7560)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 7561)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 7561)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 7562)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 7562)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 7562)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 7563)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7562) .

83.3.1 xmldom.DOMVendorNotSupported Function

Pascal

```
procedure DOMVendorNotSupported(const PropOrMethod: string; const VendorName: string);
```

C++

```
DOMVendorNotSupported(const UnicodeString PropOrMethod, const UnicodeString VendorName);
```

File

xmldom

Description

This is function xmldom.DOMVendorNotSupported.

83.3.2 xmldom.ExtractLocalName Function

Strips the namespace prefix, if present, from the tag name of an XML node.

Pascal

```
function ExtractLocalName(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractLocalName(const DOMString AName);
```

File

xmldom

Description

Call ExtractLocalName to convert a full tag name for an XML node into the corresponding local name.

AName is the tag name to be converted.

ExtractLocalName returns the value of AName with any namespace prefix stripped away.

See Also

ExtractPrefix (see page 7558)

IsPrefixed (see page 7560)

NodeMatches (see page 7561)

IXMLNode

SameNamespace (see page 7562)

83.3.3 xmldom.ExtractPrefix Function

Returns the namespace prefix of an XML node's tag name.

Pascal

```
function ExtractPrefix(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractPrefix(const DOMString AName);
```

File

xmldom

Description

Call ExtractPrefix to determine the namespace prefix, if any, of a full tag name for an XML node.

AName is the tag name of the XML node.

ExtractPrefix returns the namespace prefix, not including the colon (:) that separates the namespace from the local name of the XML node. If the tag name does not include a namespace prefix, ExtractPrefix returns an empty string.

See Also

ExtractLocalName (see page 7557)

IsPrefixed (see page 7560)

SameNamespace (see page 7562)

NodeMatches (see page 7561)

83.3.4 xmldom.GetDOM Function

Returns the top-level DOM interface for a registered DOM implementation.

Pascal

```
function GetDOM(const VendorDesc: string = ''): IDOMImplementation;
```

C++

```
IDOMImplementation GetDOM(const UnicodeString VendorDesc = '');
```

File

xmldom

Description

Call GetDOM to obtain an IDOMImplementation (see page 7504) interface that you can use for parsing and editing XML documents. The IDOMImplementation (see page 7504) interface is the top-level DOM interface, which you can use to check for the presence of a specific feature or to obtain an interface for working with a DOM document.

VendorDesc is a short descriptive name that identifies the DOM implementation. This name is returned by the Description method of the DOM implementation factory that is used to register the implementation.

Note: The DOM implementation must be registered before you can access it using GetDOM. To register a DOM implementation, use the RegisterDOMVendor (see page 7562) procedure.

See Also

RegisterDOMVendor (see page 7562)

DOMVendors (see page 7568)

GetDOMVendor (see page 7560)

TXMLDocument

Using the Document Object Model

83.3.5 xmldom.GetDOMNodeEx Function

Pascal

```
function GetDOMNodeEx(const Node: IDOMNode): IDOMNodeEx;
```

C++

```
IDOMNodeEx GetDOMNodeEx(const IDOMNode Node);
```

File

xmldom

Description

This is function xmldom.GetDOMNodeEx.

83.3.6 xmldom.GetDOMVendor Function

Returns the TDOMVendor (see page 7466) object for a specified DOM vendor.

Pascal

```
function GetDOMVendor(VendorDesc: string): TDOMVendor;
```

C++

```
TDOMVendor GetDOMVendor(UnicodeString VendorDesc);
```

File

xmldom

Description

Call GetDOMVendor to obtain the TDOMVendor (see page 7466) object that represents a registered DOM implementation. You can then assign this return value as the DOMVendor property of TXMLDocument, or call this object's DOMImplementation method to access the IDOMImplementation (see page 7504) interface for the DOM implementation.

VendorDesc is the descriptive string that identifies the particular DOM implementation that is desired. This is the value returned by the TDOMVendor (see page 7466) object's Description method.

See Also

RegisterDOMVendor (see page 7562)

DOMVendors (see page 7568)

TXMLDocument

83.3.7 xmldom.IsPrefixed Function

Indicates whether an XML tag name includes a namespace prefix.

Pascal

```
function IsPrefixed(const AName: DOMString): Boolean;
```

C++

```
Boolean IsPrefixed(const DOMString AName);
```

File

xmldom

Description

Call IsPrefixed to determine whether a tag name includes a namespace prefix.

AName is the tag name of the XML node.

IsPrefixed returns true if AName includes a namespace prefix followed by the colon (:) separator character, false otherwise.

See Also

ExtractLocalName (see page 7557)

ExtractPrefix (see page 7558)

IXMLNode

83.3.8 xmldom.MakeNodeName Function

Assembles a full tag name from a local name and namespace prefix.

Pascal

```
function MakeNodeName(const Prefix: DOMString; const LocalName: DOMString): DOMString;
```

C++

```
DOMString MakeNodeName(const DOMString Prefix, const DOMString LocalName);
```

File

xmldom

Description

Call MakeNodeName to create a tag name from a namespace prefix and a local tag name.

Prefix is the namespace prefix.

LocalName is the local name for the node.

MakeNodeName returns the full tag name that consists of the namespace prefix, followed by a colon and then the local name.

See Also

ExtractPrefix (see page 7558)

ExtractLocalName (see page 7557)

83.3.9 xmldom.NodeMatches Function

Indicates whether a specified node matches a given namespace and tag name.

Pascal

```
function NodeMatches(const Node: IDOMNode; const TagName: DOMString; const NamespaceURI:
DOMString): Boolean; overload;
```

C++

```
Boolean NodeMatches(const IDOMNode Node, const DOMString TagName, const DOMString
NamespaceURI);
```

File

xmldom

Description

Call NodeMatches to determine whether the node with the interface specified by Node refers to the same node as is described by the TagName and NamespaceURI parameters.

NodeMatches returns true if

The value of Node's NamespaceURI property is the same as the NamespaceURI parameter.

The local name or tag name of Node is the same as the TagName parameter.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 7509) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 7557)

ExtractPrefix (see page 7558)

IXMLNode

SameNamespace (see page 7562)

IXMLNode

83.3.10 xmldom.RegisterDOMVendor Function

Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.

Pascal

```
procedure RegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
RegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call RegisterDOMVendor to register a DOM implementation that you want to use for parsing XML documents. A DOM (Document Object Model) implementation is a set of objects that implement the standard DOM interfaces.

If you are using another DOM implementation, you must create a TDOMVendor (see page 7466) descendant that can supply the IDOMImplementation (see page 7504) interface and description string for the DOM implementation. Pass an instance of this object as the Vendor parameter to RegisterDOMVendor.

See Also

UnRegisterDOMVendor (see page 7563)

TXMLDocument

TXMLDocument

DOMVendors (see page 7568)

Using the Document Object Model

83.3.11 xmldom.SameNamespace Function

Indicates whether a specified node is defined within a specified namespace.

Pascal

```
function SameNamespace(const Node: IDOMNode; const namespaceURI: WideString): Boolean;
overload;
```

```
function SameNamespace(const URI1: WideString; const URI2: WideString): Boolean; overload;
```

C++

```
Boolean SameNamespace(const IDOMNode Node, const BSTR namespaceURI);
Boolean SameNamespace(const BSTR URI1, const BSTR URI2);
```

File

xmldom

Description

Call SameNamespace to determine whether the two parameters identify the same namespace.

Using the first syntax, the Node parameter identifies a node and the NamespaceURI parameter a namespace. SameNamespace checks whether the node is defined in the specified namespace.

Using the second syntax, both parameters are namespace URI values. SameNamespace checks whether they both identify the same namespace.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 7509) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 7557)

ExtractPrefix (see page 7558)

IXMLNode

SameNamespace

IXMLNode

IXMLNode

83.3.12 xmldom.UnRegisterDOMVendor Function

Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7562) .

Pascal

```
procedure UnRegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
UnRegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call UnRegisterDOMVendor to unregister a DOM implementation when it is no longer available. After unregistering a DOM implementation, it can't be used by TXMLDocument for parsing XML documents.

Vendor is the TDOMVendor (see page 7466) descendant the represents the DOM implementation that is to be unregistered. It must be the same object used when registering the DOM implementation when you called RegisterDOMVendor (see page 7562).

See Also

RegisterDOMVendor (see page 7562)

TXMLDocument

TXMLDocument

DOMVendors (see page 7568)

Using the Document Object Model

83.4 Types

The following table lists types in this documentation.

Types

Name	Description
DOMNodeType (see page 7565)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 7565)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 7566)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 7566)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 7567)	This is type xmldom.TDOMVendorArray.

83.4.1 xmldom.DOMNodeType Type

DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.

Pascal

```
DOMNodeType = Word;
```

C++

```
Word DOMNodeType;
```

File

xmldom

Description

DOMNodeType is the type of the nodeType property on the IDOMNode (see page 7509) interface. It is a flag, defined by W3C, that indicates the role played by a node in the parsed version of an XML document.

See Also

IDOMNode

83.4.2 xmldom.DOMString Type

DOMString is the type for strings that are used by a DOM parser.

Pascal

```
DOMString = UnicodeString;
```

C++

```
UnicodeString DOMString;
```

File

xmldom

Description

DOM implementations work with wide characters. The strings that are used, therefore, when working with DOM interfaces, are of type DOMString, which simply maps to the WideString (see page 3476) type.

83.4.3 xmldom.DOMTimeStamp Type

Pascal

```
DOMTimeStamp = Int64;
```

C++

```
Int64 DOMTimeStamp;
```

File

xmldom

Description

This is type xmldom.DOMTimeStamp.

83.4.4 xmldom.TAsyncEventHandler Type

TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.

Pascal

```
TAsyncEventHandler = procedure (Sender: TObject; AsyncLoadState: Integer) of object;
```

C++

```
(Sender: TObject; AsyncLoadState: Integer) ( TAsyncEventHandler)();
```

File

xmldom

Description

TAsyncEventHandler is the type for event handlers that respond at various stages during the asynchronous parsing of an XML document.

Sender is the TXMLDocument object that receives the event.

AsyncLoadState is an integer that indicates the current stage of the parsing process. Its value is vendor-specific, and corresponds to the document's AsyncLoadState property.

See Also

TXMLDocument

IXMLDocument

AsyncLoadState

83.4.5 xmldom.TDOMVendorArray Type

Pascal

```
TDOMVendorArray = array of TDOMVendor;
```

C++

```
array of TDOMVendor TDOMVendorArray;
```

File

xmldom

Description

This is type xmldom.TDOMVendorArray.

83.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefaultDOMVendor (see page 7568)	This is variable xmldom.DefaultDOMVendor.
DOMVendors (see page 7568)	This is variable xmldom.DOMVendors.

83.5.1 xmldom.DefaultDOMVendor Variable

Pascal

```
DefaultDOMVendor: string;
```

C++

```
UnicodeString DefaultDOMVendor;
```

File

xmldom

Description

This is variable xmldom.DefaultDOMVendor.

83.5.2 xmldom.DOMVendors Variable

Pascal

```
DOMVendors: TDOMVendorList;
```

C++

```
TDOMVendorList DOMVendors;
```

File

xmldom

Description

This is variable xmldom.DOMVendors.

83.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ATTRIBUTE_NODE (see page 7569)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 7570)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 7570)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 7570)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 7571)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 7571)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 7571)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 7572)	This is constant xmldom.DOMWrapperVersion.
ELEMENT_NODE (see page 7572)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 7572)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 7573)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 7573)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 7573)	This is constant xmldom.INDEX_SIZE_ERR.
INUSE_ATTRIBUTE_ERR (see page 7574)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 7574)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 7574)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 7575)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 7575)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 7575)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 7576)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 7576)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 7576)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 7577)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 7577)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 7577)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 7578)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 7578)	This is constant xmldom.SEncoding.
SHttp (see page 7578)	This is constant xmldom.SHttp.
SStandalone (see page 7579)	This is constant xmldom.SStandalone.
SVersion (see page 7579)	This is constant xmldom.SVersion.
SXML (see page 7579)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 7580)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 7580)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 7580)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 7581)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 7581)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 7581)	This is constant xmldom.WRONG_DOCUMENT_ERR.

83.6.1 xmldom.ATTRIBUTE_NODE Constant

Pascal

```
ATTRIBUTE_NODE = 2;
```

C++

```
ATTRIBUTE_NODE = 2;
```

File

xmldom

Description

This is constant xmldom.ATTRIBUTE_NODE.

83.6.2 xmldom.CDATA_SECTION_NODE Constant

Pascal

```
CDATA_SECTION_NODE = 4;
```

C++

```
CDATA_SECTION_NODE = 4;
```

File

xmldom

Description

This is constant xmldom.CDATA_SECTION_NODE.

83.6.3 xmldom.COMMENT_NODE Constant

Pascal

```
COMMENT_NODE = 8;
```

C++

```
COMMENT_NODE = 8;
```

File

xmldom

Description

This is constant xmldom.COMMENT_NODE.

83.6.4 xmldom.DOCUMENT_FRAGMENT_NODE Constant

Pascal

```
DOCUMENT_FRAGMENT_NODE = 11;
```

C++

```
DOCUMENT_FRAGMENT_NODE = 11;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_FRAGMENT_NODE.

83.6.5 xmldom.DOCUMENT_NODE Constant

Pascal

```
DOCUMENT_NODE = 9;
```

C++

```
DOCUMENT_NODE = 9;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_NODE.

83.6.6 xmldom.DOCUMENT_TYPE_NODE Constant

Pascal

```
DOCUMENT_TYPE_NODE = 10;
```

C++

```
DOCUMENT_TYPE_NODE = 10;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_TYPE_NODE.

83.6.7 xmldom.DOMSTRING_SIZE_ERR Constant

Pascal

```
DOMSTRING_SIZE_ERR = 2;
```

C++

```
DOMSTRING_SIZE_ERR = 2;
```

File

xmldom

Description

This is constant xmldom.DOMSTRING_SIZE_ERR.

83.6.8 xmldom.DOMWrapperVersion Constant

Pascal

```
DOMWrapperVersion = 1.3;
```

C++

```
DOMWrapperVersion = 1.3;
```

File

xmldom

Description

This is constant xmldom.DOMWrapperVersion.

83.6.9 xmldom.ELEMENT_NODE Constant

Pascal

```
ELEMENT_NODE = 1;
```

C++

```
ELEMENT_NODE = 1;
```

File

xmldom

Description

This is constant xmldom.ELEMENT_NODE.

83.6.10 xmldom.ENTITY_NODE Constant

Pascal

```
ENTITY_NODE = 6;
```

C++

```
ENTITY_NODE = 6;
```

File

xmldom

Description

This is constant xmldom.ENTITY_NODE.

83.6.11 xmldom.ENTITY_REFERENCE_NODE Constant

Pascal

```
ENTITY_REFERENCE_NODE = 5;
```

C++

```
ENTITY_REFERENCE_NODE = 5;
```

File

xmldom

Description

This is constant xmldom.ENTITY_REFERENCE_NODE.

83.6.12 xmldom.HIERARCHY_REQUEST_ERR Constant

Pascal

```
HIERARCHY_REQUEST_ERR = 3;
```

C++

```
HIERARCHY_REQUEST_ERR = 3;
```

File

xmldom

Description

This is constant xmldom.HIERARCHY_REQUEST_ERR.

83.6.13 xmldom.INDEX_SIZE_ERR Constant

Pascal

```
INDEX_SIZE_ERR = 1;
```

C++

```
INDEX_SIZE_ERR = 1;
```

File

xmldom

Description

This is constant xmldom.INDEX_SIZE_ERR.

83.6.14 xmldom.INUSE_ATTRIBUTE_ERR Constant

Pascal

```
INUSE_ATTRIBUTE_ERR = 10;
```

C++

```
INUSE_ATTRIBUTE_ERR = 10;
```

File

xmldom

Description

This is constant xmldom.INUSE_ATTRIBUTE_ERR.

83.6.15 xmldom.INVALID_ACCESS_ERR Constant

Pascal

```
INVALID_ACCESS_ERR = 15;
```

C++

```
INVALID_ACCESS_ERR = 15;
```

File

xmldom

Description

This is constant xmldom.INVALID_ACCESS_ERR.

83.6.16 xmldom.INVALID_CHARACTER_ERR Constant

Pascal

```
INVALID_CHARACTER_ERR = 5;
```

C++

```
INVALID_CHARACTER_ERR = 5;
```

File

xmldom

Description

This is constant xmldom.INVALID_CHARACTER_ERR.

83.6.17 xmldom.INVALID_MODIFICATION_ERR Constant

Pascal

```
INVALID_MODIFICATION_ERR = 13;
```

C++

```
INVALID_MODIFICATION_ERR = 13;
```

File

xmldom

Description

This is constant xmldom.INVALID_MODIFICATION_ERR.

83.6.18 xmldom.INVALID_STATE_ERR Constant

Pascal

```
INVALID_STATE_ERR = 11;
```

C++

```
INVALID_STATE_ERR = 11;
```

File

xmldom

Description

This is constant xmldom.INVALID_STATE_ERR.

83.6.19 xmldom.NAMESPACE_ERR Constant

Pascal

```
NAMESPACE_ERR = 14;
```

C++

```
NAMESPACE_ERR = 14;
```

File

xmldom

Description

This is constant xmldom.NAMESPACE_ERR.

83.6.20 xmldom.NO_DATA_ALLOWED_ERR Constant

Pascal

```
NO_DATA_ALLOWED_ERR = 6;
```

C++

```
NO_DATA_ALLOWED_ERR = 6;
```

File

xmldom

Description

This is constant xmldom.NO_DATA_ALLOWED_ERR.

83.6.21 xmldom.NO_MODIFICATION_ALLOWED_ERR Constant

Pascal

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

C++

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

File

xmldom

Description

This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.

83.6.22 xmldom.NOT_FOUND_ERR Constant

Pascal

```
NOT_FOUND_ERR = 8;
```

C++

```
NOT_FOUND_ERR = 8;
```

File

xmldom

Description

This is constant xmldom.NOT_FOUND_ERR.

83.6.23 xmldom.NOT_SUPPORTED_ERR Constant

Pascal

```
NOT_SUPPORTED_ERR = 9;
```

C++

```
NOT_SUPPORTED_ERR = 9;
```

File

xmldom

Description

This is constant xmldom.NOT_SUPPORTED_ERR.

83.6.24 xmldom.NOTATION_NODE Constant

Pascal

```
NOTATION_NODE = 12;
```

C++

```
NOTATION_NODE = 12;
```

File

xmldom

Description

This is constant xmldom.NOTATION_NODE.

83.6.25 xmldom.NSDelim Constant

Pascal

```
NSDelim = ':';
```

C++

```
NSDelim = ':';
```

File

xmldom

Description

This is constant xmldom.NSDelim.

83.6.26 xmldom.PROCESSING_INSTRUCTION_NODE Constant

Pascal

```
PROCESSING_INSTRUCTION_NODE = 7;
```

C++

```
PROCESSING_INSTRUCTION_NODE = 7;
```

File

xmldom

Description

This is constant xmldom.PROCESSING_INSTRUCTION_NODE.

83.6.27 xmldom.SEncoding Constant

Pascal

```
SEncoding = 'encoding';
```

C++

```
SEncoding = 'encoding';
```

File

xmldom

Description

This is constant xmldom.SEncoding.

83.6.28 xmldom.SHttp Constant

Pascal

```
SHttp = 'http:/';
```

C++

```
SHttp = 'http:/';
```

File

xmldom

Description

This is constant xmldom.SHttp.

83.6.29 xmldom.SStandalone Constant

Pascal

```
SStandalone = 'standalone';
```

C++

```
SStandalone = 'standalone';
```

File

xmldom

Description

This is constant xmldom.SStandalone.

83.6.30 xmldom.SVersion Constant

Pascal

```
SVersion = 'version';
```

C++

```
SVersion = 'version';
```

File

xmldom

Description

This is constant xmldom.SVersion.

83.6.31 xmldom.SXML Constant

Pascal

```
SXML = 'xml';
```

C++

```
SXML = 'xml';
```

File

xmldom

Description

This is constant xmldom.SXML.

83.6.32 xmldom.SXMLNamespaceURI Constant

Pascal

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

C++

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

File

xmldom

Description

This is constant xmldom.SXMLNamespaceURI.

83.6.33 xmldom.SXMLNS Constant

Pascal

```
SXMLNS = 'xmlns';
```

C++

```
SXMLNS = 'xmlns';
```

File

xmldom

Description

This is constant xmldom.SXMLNS.

83.6.34 xmldom.SXMLPrefixNamespaceURI Constant

Pascal

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

C++

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

File

xmldom

Description

This is constant xmldom.SXMLPrefixNamespaceURI.

83.6.35 xmldom.SYNTAX_ERR Constant

Pascal

```
SYNTAX_ERR = 12;
```

C++

```
SYNTAX_ERR = 12;
```

File

xmldom

Description

This is constant xmldom.SYNTAX_ERR.

83.6.36 xmldom.TEXT_NODE Constant

Pascal

```
TEXT_NODE = 3;
```

C++

```
TEXT_NODE = 3;
```

File

xmldom

Description

This is constant xmldom.TEXT_NODE.

83.6.37 xmldom.WRONG_DOCUMENT_ERR Constant

Pascal

```
WRONG_DOCUMENT_ERR = 4;
```

C++

```
WRONG_DOCUMENT_ERR = 4;
```

File

xmldom

Description

This is constant xmldom.WRONG_DOCUMENT_ERR.

84 XMLIntf Namespace

This is namespace XMLIntf.

Topics

Name	Description
Classes (see page 7585)	The following table lists classes in this documentation.
Interfaces (see page 7586)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 7662)	The following table lists structs, records, enums in this documentation.
Types (see page 7667)	The following table lists types in this documentation.
Constants (see page 7668)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 7585)	EXMLDocError is the exception class for errors editing or parsing an XML document.

Constants

Name	Description
NodeTypeNames (see page 7668)	This is constant XMLIntf.NodeTypeNames.

Interfaces

	Name	Description
	IXMLDocument (see page 7586)	IXMLDocument represents an XML (see page 7614) document.
	IXMLNode (see page 7615)	IXMLNode is the interface for a node in an XML (see page 7643) document.
	IXMLNodeCollection (see page 7644)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 7649)	IXMLNodeList represents a set of nodes in an XML document.

Structs, Records, Enums

	Name	Description
	TNodeType (see page 7662)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 7664)	TParseOption and TParseOptions (see page 7667) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 7664)	TXMLDocOption and TXMLDocOptions (see page 7667) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 7665)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

Types

Name	Description
TParseOptions (see page 7667)	This is type XMLIntf.TParseOptions.
TXMLDocOptions (see page 7667)	This is type XMLIntf.TXMLDocOptions.

84.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 7585)	EXMLDocError is the exception class for errors editing or parsing an XML document.

84.1.1 EXMLDocError Class

EXMLDocError is the exception class for errors editing or parsing an XML document.

Class Hierarchy

Pascal

```
EXMLDocError = class(Exception);
```

C++

```
class EXMLDocError : public Exception;
```

File

XMLIntf

Description

EXMLDocError is raised by TXMLDocument or TXMLNode when

An error occurs when loading an XML document.

An error occurs when creating a node in an XML document.

A required DOM interface is not found.

An option is requested that is not supported by the DOM implementation.

An attempt is made to perform an action that is not consistent with the specified node type.

See Also

XMLDocError

TXMLDocument (see page 7299)

TXMLNode (see page 7348)

84.2 Interfaces

The following table lists interfaces in this documentation.

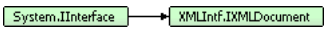
Interfaces

	Name	Description
	IXMLDocument (see page 7586)	IXMLDocument represents an XML (see page 7614) document.
	IXMLNode (see page 7615)	IXMLNode is the interface for a node in an XML (see page 7643) document.
	IXMLNodeCollection (see page 7644)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 7649)	IXMLNodeList represents a set of nodes in an XML document.

84.2.1 IXMLDocument Interface

IXMLDocument represents an XML (see page 7614) document.

Class Hierarchy



Pascal

```
IXMLDocument = interface(IInterface);
```

C++

```
__interface IXMLDocument : public IInterface;
```

File

XMLIntf

Description

Use IXMLDocument to work with an XML (see page 7614) document. Applications can obtain an IXMLDocument interface through a TXMLDocument component, using the global LoadXMLDocument function, or through the OwnerDocument property of any node in the document.

Like the related DOM interface IDOMDocument (see page 7482), you can use IXMLDocument to access the contents of an XML (see page 7614) document and to create new nodes. In addition, IXMLDocument lets you load an XML (see page 7614) document that is stored on disk or assign a string of raw (unparsed) XML (see page 7614) as the contents of the document.

TXMLDocument implements the IXMLDocument interface. Typically, the TXMLDocument component is the starting point for working with an XML (see page 7614) document.

Note: C++ method declarations that use IXMLDocument use the _di_IXMLDocument type instead. This type is a DelphiInterface (see page 3263) wrapper around the IXMLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLDocument >  _di_IXMLDocument;
```

See Also

IXMLNode (see page 7615)

EXMLDocError (see page 7585)

TXMLDocument (see page 7299)

IXMLNode (see page 7615)

LoadXMLDocument

84.2.1.1 IXMLDocument Members

The following tables list the members exposed by IXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.

	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.
	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.
	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

84.2.1.2 IXMLDocument Methods

The methods of the IXMLDocument class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.

	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.
	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

84.2.1.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLDocument.AddChild (DOMString) (see page 7589)	Creates and adds a node to the child nodes of this document.
	IXMLDocument.AddChild (DOMString, DOMString) (see page 7589)	Creates and adds a node to the child nodes of this document.

84.2.1.2.1.1 IXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 7609) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 7614) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 7618)

CreateNode (see page 7591)

CreateElement (see page 7590)

ChildNodes (see page 7609)

DocumentElement (see page 7609)

84.2.1.2.2 IXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7609) list of any node in the document (including this document's ChildNodes (see page 7609) property). It can also be assigned to the DocumentElement (see page 7609) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7589) method.

See Also

AddChild (see page 7618)

ChildNodes (see page 7609)

DocumentElement (see page 7609)

AddChild (see page 7589)

CreateNode (see page 7591)

84.2.1.2.3 IXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 7614) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7609) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 7609) property).

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.

ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7589) method.

See Also

AddChild (see page 7618)

ChildNodes (see page 7609)

AttributeNodes (see page 7634)

DocumentElement (see page 7609)

CreateElement (see page 7590)

AddChild (see page 7589)

84.2.1.2.4 IXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

The AddChild (see page 7589) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node calls GeneratePrefix to generate a unique string to act as a namespace prefix.

Node (see page 7611) is the interface for the node which will use the newly-generated namespace prefix.

GeneratePrefix returns a unique string that is not already used in the document to represent a namespace URI. The generated namespace prefixes are only unique within a single document.

See Also

AddChild (see page 7618)

84.2.1.2.5 IXMLDocument.GetActive Method

Returns the value of the Active (see page 7607) property.

Pascal

```
function GetActive: Boolean;
```

C++

```
__fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active (see page 7607) property.

See Also

Active (see page 7607)

84.2.1.2.6 IXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState (see page 7608) property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState (see page 7608) property.

See Also

AsyncLoadState (see page 7608)

84.2.1.2.7 IXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes (see page 7609) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 7609) property.

See Also

ChildNodes (see page 7609)

84.2.1.2.8 IXMLDocument.GetDocBinding Method

Replaces a specified child node with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class for one of the child nodes of the document node. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7609)

RegisterDocBinding (see page 7600)

ChildNodes (see page 7609)

84.2.1.2.9 IXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement (see page 7609) property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement (see page 7609) property.

See Also

DocumentElement (see page 7609)

84.2.1.2.10 IXMLDocument.GetDocumentNode Method

Returns the value of the Node (see page 7611) property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node (see page 7611) property.

See Also

Node (see page 7611)

84.2.1.2.11 IXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument (see page 7609) property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument (see page 7609) property.

See Also

DOMDocument (see page 7609)

84.2.1.2.12 IXMLDocument.GetEncoding Method

Returns the value of the Encoding (see page 7610) property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding (see page 7610) property.

See Also

Encoding (see page 7610)

84.2.1.2.13 IXMLDocument.GetFileName Method

Returns the value of the FileName (see page 7610) property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName (see page 7610) property.

See Also

FileName (see page 7610)

84.2.1.2.14 IXMLDocument.GetModified Method

Returns the value of the Modified (see page 7611) property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified (see page 7611) property.

See Also

Modified (see page 7611)

84.2.1.2.15 IXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr (see page 7612) property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr (see page 7612) property.

See Also

NodeIndentStr (see page 7612)

84.2.1.2.16 IXMLDocument.GetOptions Method

Returns the value of the Options (see page 7612) property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options (see page 7612) property.

See Also

Options (see page 7612)

84.2.1.2.17 IXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions (see page 7613) property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions (see page 7613) property.

See Also

ParseOptions (see page 7613)

84.2.1.2.18 IXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef (see page 7613) property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef (see page 7613) property.

See Also

SchemaRef (see page 7613)

84.2.1.2.19 IXMLDocument.GetStandAlone Method

Returns the value of the StandAlone (see page 7614) property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone (see page 7614) property.

See Also

StandAlone (see page 7614)

84.2.1.2.20 IXMLDocument.GetVersion Method

Returns the value of the Version (see page 7614) property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version (see page 7614) property.

See Also

Version (see page 7614)

84.2.1.2.21 IXMLDocument.GetXML Method

Returns the value of the XML (see page 7614) property.

Pascal

```
function GetXML: TStrings;
```

C++

```
__fastcall TStrings GetXML();
```

Description

GetXML is the read implementation of the XML (see page 7614) property.

See Also

XML (see page 7614)

84.2.1.2.22 IXMLDocument.IsEmptyDoc Method

Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the IXMLDocument (see page 7586) instance represents an empty document. IsEmptyDoc returns true if the DOMDocument (see page 7609) property is not set (for example, if there is no DOM implementation) or if this interface represents a document with no child nodes.

See Also

XML (see page 7614)

ChildNodes (see page 7609)

DOMDocument (see page 7609)

84.2.1.2.23 IXMLDocument.LoadFromFile Method

Loads an XML (see page 7614) document from disk.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString);
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName);
```

Description

Call LoadFromFile to load the XML (see page 7614) document specified by AFileName so that you can examine or modify it.

AFileName is the name of the file to load.

Once you have loaded an XML (see page 7614) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7601) method.

See Also

SaveToFile (see page 7601)

XML (see page 7614)

FileName (see page 7610)

84.2.1.2.24 IXMLDocument.LoadFromStream Method

Loads an XML (see page 7614) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 7614) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 7614) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 7614) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 7607) property to true.

See Also

SaveToStream (see page 7602)

LoadFromFile (see page 7598)

LoadFromXML (see page 7599)

Active (see page 7607)

84.2.1.2.25 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	IXMLDocument.LoadFromXML (AnsiString) (see page 7599)	Loads a string representation of an XML (see page 7614) document and activates it.
	IXMLDocument.LoadFromXML (DOMString) (see page 7599)	Loads a string representation of an XML (see page 7614) document and activates it.

84.2.1.2.25.1 IXMLDocument.LoadFromXML Method (AnsiString)

Loads a string representation of an XML (see page 7614) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: AnsiString); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 7614) document. Unlike the XML (see page 7614)

property, which lets you assign XML (see page 7614) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 7614) document as a whole.

After assigning the XML (see page 7614) property as the contents of the document, LoadFromXML sets the Active (see page 7607) property to true.

See Also

SaveToXML (see page 7602)

LoadFromFile (see page 7598)

Active (see page 7607)

LoadFromStream (see page 7599)

XML (see page 7614)

84.2.1.2.26 IXMLDocument.Refresh Method

Updates the parsed XML (see page 7614) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 7614) document specified by FileName (see page 7610) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 7614) document (which can be traversed starting with DocumentElement (see page 7609)) to reflect the current contents of the file specified by FileName (see page 7610).

The Active (see page 7607) property must be true when you call Refresh.

Note: Do not call Refresh after making changes directly to the XML (see page 7614) property, because assigning a value to XML (see page 7614) automatically closes the XML (see page 7614) document. To refresh the XML (see page 7614) document after assigning a new value to XML (see page 7614), set the Active (see page 7607) property to true.

See Also

FileName (see page 7610)

Active (see page 7607)

DOMDocument (see page 7609)

DocumentElement (see page 7609)

Resync (see page 7601)

84.2.1.2.27 IXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7609)

GetDocBinding (see page 7593)

ChildNodes (see page 7609)

84.2.1.2.28 IXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7609)

84.2.1.2.29 IXMLDocument.SaveToFile Method

Saves the XML (see page 7614) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString);
```

C++

```
__fastcall SaveToFile(const DOMString AFileName);
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 7614) document.

AFileName is the name of the file to save.

Tip: To save the modifications back to the file from which the document was originally read, use the value of the FileName (see page 7610) property for the AFileName parameter.

See Also

LoadFromFile (see page 7598)

XML (see page 7614)

FileName (see page 7610)

84.2.1.2.30 IXMLDocument.SaveToStream Method

Saves the XML (see page 7614) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 7614) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 7614) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 7599)

SaveToXML (see page 7602)

SaveToFile (see page 7601)

84.2.1.2.31 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	IXMLDocument.SaveToXML (DOMString) (see page 7602)	Saves the XML (see page 7614) document to a string-type variable.
	IXMLDocument.SaveToXML (UTF8String) (see page 7602)	Saves the XML (see page 7614) document to a string-type variable.
	IXMLDocument.SaveToXML (WideString) (see page 7602)	Saves the XML (see page 7614) document to a string-type variable.

84.2.1.2.31.1 IXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 7614) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: WideString); overload;
procedure SaveToXML(var XML: UTF8String); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
```

```
__fastcall SaveToXML(BSTR XML);
__fastcall SaveToXML(UTF8String XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 7614) document to the string-type variable specified by XML (see page 7614). SaveToXML writes the contents of XML (see page 7614) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 7614) parameter.

Unlike the XML (see page 7614) property, which lets you write individual lines from the XML (see page 7614) document, SaveToXML writes the entire text of the XML (see page 7614) document.

See Also

LoadFromXML (see page 7599)

SaveToStream (see page 7602)

SaveToFile (see page 7601)

XML (see page 7614)

84.2.1.2.32 IXMLDocument.SetActive Method

Sets the value of the Active (see page 7607) property.

Pascal

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

C++

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

Description

SetActive is the write implementation of the Active (see page 7607) property.

Value is the new value to set.

See Also

Active (see page 7607)

84.2.1.2.33 IXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement (see page 7609) property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement (see page 7609) property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 7609)

84.2.1.2.34 IXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument (see page 7609) property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument (see page 7609) property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 7609)

84.2.1.2.35 IXMLDocument.SetEncoding Method

Sets the value of the Encoding (see page 7610) property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding (see page 7610) property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 7610)

84.2.1.2.36 IXMLDocument.SetFileName Method

Sets the value of the FileName (see page 7610) property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName (see page 7610) property.

Value is the name of an XML (see page 7614) document or, if the vendor supports it, an URL for the XML (see page 7614) document.

See Also

FileName (see page 7610)

84.2.1.2.37 IXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr (see page 7612) property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr (see page 7612) property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 7612)

84.2.1.2.38 IXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML (see page 7614) document progresses to a certain point.

Value is a method that is called when the AsyncLoadState (see page 7608) property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 7608)

84.2.1.2.39 IXMLDocument.SetOptions Method

Sets the value of the Options (see page 7612) property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions (see page 7596) is the write implementation of the Options (see page 7612) property.

Value is the set of options to assign.

See Also

Options (see page 7612)

84.2.1.2.40 IXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions (see page 7613) property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions (see page 7596) is the write implementation of the ParseOptions (see page 7613) property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 7613)

84.2.1.2.41 IXMLDocument.SetStandAlone Method

Sets the value of the StandAlone (see page 7614) property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone (see page 7614) property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 7614)

84.2.1.2.42 IXMLDocument.SetVersion Method

Sets the value of the Version (see page 7614) property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version (see page 7614) property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 7614)

84.2.1.2.43 IXMLDocument.SetXML Method

Sets the value of the XML (see page 7614) property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML (see page 7614) property.

Value contains the raw (unparsed) XML (see page 7614) to assign.

See Also

XML (see page 7614)

84.2.1.3 IXMLDocument Properties

The properties of the IXMLDocument class are listed here.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.
	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

84.2.1.3.1 IXMLDocument.Active Property

Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use IXMLDocument (see page 7586) to examine or change the contents of an XML (see page 7614) document, it must be active. Setting Active to true results in the parsing of the XML (see page 7614) document specified by the FileName (see page 7610) property or by the XML (see page 7614) property. If both FileName (see page 7610) and XML (see page 7614) are empty strings, Setting Active to true causes IXMLDocument (see page 7586) to represent an empty document.

Active must be true before you can access the DOMDocument (see page 7609) or DocumentElement (see page 7609) properties.

See Also

DOMDocument (see page 7609)

DocumentElement (see page 7609)

FileName (see page 7610)

XML (see page 7614)

84.2.1.3.2 IXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 7614) document asynchronously. The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 7614) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 7614) document.
2	Loaded: The parser has finished reading and parsing the structure of the XML (see page 7614) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 7614) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 7614) document.

See Also

SetOnAsyncLoad (see page 7605)

84.2.1.3.3 IXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 7609), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

See Also

DocumentElement (see page 7609)

Node (see page 7611)

84.2.1.3.4 IXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 7614) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 7614) document. Because XML (see page 7614) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 7614) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

The IXMLNode (see page 7615) interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

See Also

DOMDocument (see page 7609)

ChildNodes (see page 7609)

84.2.1.3.5 IXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 7614) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument (see page 7482) interface for the parsed XML (see page 7614) document. This is the interface supported by the DOM parser for the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 7609). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Specification.

See Also

DocumentElement (see page 7609)

84.2.1.3.6 IXMLDocument.Encoding Property

Indicates the character set that the XML (see page 7614) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 7614) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 7610)

DOMDocument (see page 7609)

XML (see page 7614)

84.2.1.3.7 IXMLDocument.FileName Property

Indicates the XML (see page 7614) document file, if any, that this interface represents.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Use FileName to get or set the name of the XML (see page 7614) file that this document represents. FileName is set automatically when an application calls the LoadFromFile (see page 7598) method or when this interface is obtained from the global LoadXMLDocument function.

When you set the value of FileName, the XML (see page 7614) property automatically changes (at runtime) to represent the contents of that file.

Note: The content of this XML (see page 7614) document, which is available as the value of the XML (see page 7614) property, may not be the same as the content of the specified file. If an application uses IXMLDocument (see page 7586) to

edit the document, those changes are not saved back to the XML (see page 7614) document file until you call the SaveToFile (see page 7601) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

SaveToFile (see page 7601)

LoadFromFile (see page 7598)

LoadXMLDocument (see page 7442)

XML (see page 7614)

84.2.1.3.8 IXMLDocument.Modified Property

Indicates whether the XML (see page 7614) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read (see page 3727) Modified to determine whether the application has made any changes to the nodes in the XML (see page 7614) document. When Modified is true, the value of the XML (see page 7614) property may differ from the contents of the file specified by FileName (see page 7610). You can call the SaveToFile (see page 7601) method to update the XML (see page 7614) document file to reflect the changes.

See Also

Resync (see page 7601)

SaveToFile (see page 7601)

XML (see page 7614)

FileName (see page 7610)

84.2.1.3.9 IXMLDocument.Node Property

Provides access to the document node for the XML (see page 7614) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read (see page 3727) Node to access the document node for the XML (see page 7614) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 7609) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 7609)

ChildNodes (see page 7609)

84.2.1.3.10 IXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how the document formats nodes that are added. When the Options (see page 7612) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 7614) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, IXMLDocument (see page 7586) inserts two spaces to represent each nesting level.

See Also

Options (see page 7612)

NestingLevel (see page 7380)

FormatXMLData (see page 7441)

XML (see page 7614)

84.2.1.3.11 IXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 7614) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7649) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 7614) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.

doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (see page 3430) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild (see page 7589) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 7589) creates the new namespace but does not add an attribute to declare it.
doAutoSave	When you close the document (set Active (see page 7607) to false), any changes are automatically saved back to the file specified by FileName (see page 7610) or to the XML (see page 7614) property.

See Also

Nodes (see page 7660)

Attributes (see page 7634)

AddChild (see page 7618)

84.2.1.3.12 IXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 7614) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, IXMLDocument (see page 7586) raises an EXMLDocError (see page 7585) exception.

See Also

domimplementation (see page 7337)

EXMLDocError (see page 7585)

84.2.1.3.13 IXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 7614) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read (see page 3727) SchemaRef to determine the schema associated with this XML (see page 7614) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 7609)

84.2.1.3.14 IXMLDocument.StandAlone Property

Indicates whether the XML (see page 7614) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 7614) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 7609)

84.2.1.3.15 IXMLDocument.Version Property

Indicates the XML (see page 7614) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 7614) version number that the document uses. An XML (see page 7614) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 7614)

84.2.1.3.16 IXMLDocument.XML Property

Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read (see page 3727) XML to obtain the contents of the associated XML document. Set XML to specify the raw (unparsed)

XML you want to examine or manipulate.

XML is a TStrings object. This means you can manipulate the XML line by line. To read or write the XML in the document as a single string, use the LoadFromXML (see page 7599) or SaveToXML (see page 7602) method.

Note: If you call the LoadFromFile (see page 7598), LoadFromStream (see page 7599), or LoadFromXML (see page 7599) method, XML is automatically updated to reflect the contents of that file.

See Also

LoadFromFile (see page 7598)

DocumentElement (see page 7609)

TXMLDocument (see page 7299)

LoadFromXML (see page 7599)

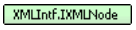
SaveToXML (see page 7602)

LoadFromStream (see page 7599)

84.2.2 IXMLNode Interface

IXMLNode is the interface for a node in an XML (see page 7643) document.

Class Hierarchy



Pascal

```
IXMLNode = interface;
```

C++

```
__interface IXMLNode;
```

File

XMLIntf

Description

Use IXMLNode to work with a node in an XML (see page 7643) document. The IXMLNode interface is designed to work with TXMLDocument and the code generated by the Data Binding wizard.

IXMLNode is similar to the IDOMNode (see page 7509) interface that is defined as part of the Document Object Model (DOM) specification. However, IXMLNode is designed to simplify the task of navigating and editing XML (see page 7643) documents by

including some of the properties and methods from the IDOMElement (see page 7494) interface, resulting in a simpler model for examining an XML (see page 7643) node.

including additional properties and methods to simplify access to data in common XML (see page 7643) structures.

The Data Binding wizard generates descendants of IXMLNode that further simplify the task of working with XML (see page 7643) documents by converting the attributes and child nodes of a node into properties of the node interface.

Note: C++ method declarations that use IXMLNode use the _di_IXMLNode type instead. This type is a DelphiInterface (see page 3263) wrapper around the IXMLNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNode >  _di_IXMLNode;
```

See Also

TXMLNode (see page 7348)

TXMLDocument (see page 7299)

IXMLNodeCollection (see page 7644)

IDOMNode (see page 7509)

IXMLNodeList (see page 7649)

Working with XML Nodes

84.2.2.1 IXMLNode Members

The following tables list the members exposed by IXMLNode.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

84.2.2.2 IXMLNode Methods

The methods of the IXMLNode class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.

	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

84.2.2.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 7618)	Adds a new child node to this node.
	IXMLNode.AddChild (DOMString, Integer) (see page 7618)	Adds a new child node to this node.

84.2.2.2.1.1 IXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
```

Description

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

NamespaceURI (see page 7639) identifies the namespace that includes the new node's definition. If NamespaceURI (see page 7639) is omitted (the second syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI (see page 7639) parameter, the namespace is the

same as this node's NamespaceURI (see page 7639) property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI (see page 7639) parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML (see page 7643) document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 7635) property array.

See Also

Delete (see page 7653)

ChildNodes (see page 7635)

CreateNode (see page 7591)

NamespaceURI (see page 7639)

Options (see page 7341)

Working with XML Nodes

84.2.2.2.2 IXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

After calling CloneNode to create a new node, you can add that node into the ChildNodes (see page 7635) property of another node (or, if this node represents an attribute, into the AttributeNodes (see page 7634) property) by calling ChildNodes (see page 7635). ->Add or ChildNodes (see page 7635). ->Insert (see page 3698).

See Also

ChildNodes (see page 7635)

AddChild (see page 7618)

ParentNode (see page 7641)

CloneNode (see page 7512)

Add (see page 7651)

Insert (see page 7658)

84.2.2.2.3 IXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix (see page 7642) is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 7612)

Attributes (see page 7634)

FindNamespaceDecl (see page 7620)

84.2.2.2.4 IXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI identified by NamespaceURI (see page 7639). If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI (see page 7639) is the namespace URI to locate.

See Also

FindNamespaceURI (see page 7620)

DeclareNamespace (see page 7620)

84.2.2.2.5 IXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

FindNamespaceDecl (see page 7620) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 7625)

GetPrefix (see page 7626)

FindNamespaceDecl (see page 7620)

84.2.2.2.6 IXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

GetAttribute is the read implementation of the Attributes (see page 7634) property.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, GetAttribute returns a Null Variant (see page 3430).

Note: Do not call GetAttribute unless this node is an element node. Only element nodes have attributes.

See Also

Attributes (see page 7634)

GetAttributeNS (see page 7622)

84.2.2.2.7 IXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 7634) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 7634) property.

See Also

AttributeNodes (see page 7634)

84.2.2.2.8 IXMLNode.GetAttributeNS Method

Returns the value of a specified attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Call GetAttributeNS to determine the value of an attribute when you must explicitly specify the namespace for the attribute.

AttrName is the name of the attribute whose value you want to fetch.

NamespaceURI (see page 7639) identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the attribute. If AttrName and NamespaceURI (see page 7639) do not identify an attribute of this node, GetAttributeNS returns a Null Variant (see page 3430).

See Also

AttributeNodes (see page 7634)

Attributes (see page 7634)

NamespaceURI (see page 7639)

SetAttributeNS (see page 7630)

84.2.2.2.9 IXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 7635) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 7635) property.

See Also

ChildNodes (see page 7635)

84.2.2.2.10 IXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

GetChildValue is the read implementation of the ChildValues (see page 7636) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

The return value depends on the type of the child node, as indicated in the following table:

NodeType (see page 7640)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, reading the child's value causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

ChildValues (see page 7636)

84.2.2.2.11 IXMLNode.GetCollection Method

Returns the value of the Collection (see page 7636) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

GetCollection is the read implementation of the Collection (see page 7636) property.

See Also

Collection (see page 7636)

84.2.2.2.12 IXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 7637) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the read implementation of the DOMNode (see page 7637) property.

See Also

DOMNode (see page 7637)

84.2.2.2.13 IXMLNode.GetHasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

GetHasChildNodes is the read implementation of the HasChildNodes (see page 7637) property.

See Also

HasChildNodes (see page 7637)

84.2.2.2.14 IXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

GetIsTextElement is the read implementation of the IsTextElement (see page 7638) property.

See Also

IsTextElement (see page 7638)

84.2.2.2.15 IXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

GetLocalName is the read implementation of the LocalName (see page 7638) property.

See Also

LocalName (see page 7638)

84.2.2.2.16 IXMLNode.GetNamespaceURI Method

Returns the namespace URI for the node's qualified name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

GetNamespaceURI is the read implementation of the NamespaceURI (see page 7639) property.

See Also

NamespaceURI (see page 7639)

84.2.2.2.17 IXMLNode.GetNodeName Method

Returns the name of the node.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

GetNodeName is the read implementation of the NodeName (see page 7639) property.

See Also

NodeName (see page 7639)

84.2.2.2.18 IXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

GetNodeType is the read implementation of the NodeType (see page 7640) property.

See Also

NodeType (see page 7640)

84.2.2.2.19 IXMLNode.GetNodeValue Method

Returns the value of the node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

GetNodeValue is the read implementation of the NodeValue (see page 7641) property.

See Also

NodeValue (see page 7641)

84.2.2.2.20 IXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

GetOwnerDocument is the read implementation of the OwnerDocument (see page 7641) property.

See Also

OwnerDocument (see page 7641)

84.2.2.2.21 IXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

GetParentNode is the read implementation of the ParentNode (see page 7641) property.

See Also

ParentNode (see page 7641)

84.2.2.2.22 IXMLNode.GetPrefix Method

Returns the namespace prefix from the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

GetPrefix is the read implementation of the Prefix (see page 7642) property.

See Also

Prefix (see page 7642)

84.2.2.2.23 IXMLNode.GetReadOnly Method

Returns the value of the ReadOnly (see page 7642) property.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly is the read implementation of the ReadOnly (see page 7642) property.

See Also

ReadOnly (see page 7642)

84.2.2.2.24 IXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

GetText is the read implementation for the Text (see page 7643) property.

See Also

Text (see page 7643)

84.2.2.2.25 IXMLNode.GetXML Method

Returns the value of the XML (see page 7643) property.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

GetXML is the read implementation of this node's XML (see page 7643) property.

See Also

XML (see page 7643)

84.2.2.2.26 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	IXMLNode.HasAttribute (DOMString) (see page 7628)	Indicates whether the node has an attribute with a specified name.
	IXMLNode.HasAttribute (DOMString, DOMString) (see page 7628)	Indicates whether the node has an attribute with a specified name.

84.2.2.2.26.1 IXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call HasAttribute to determine whether this node has a specified attribute.

Name is the name of the attribute about which you want to know.

NamespaceURI (see page 7639) is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 7634)

Attributes (see page 7634)

84.2.2.2.27 IXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

NextSibling returns the node that follows this one in the parent node's ChildNodes (see page 7635) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 7641)

PreviousSibling (see page 7629)

ChildNodes (see page 7635)

84.2.2.2.28 IXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode (see page 7615) interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 7635)

AttributeNodes (see page 7634)

84.2.2.2.29 IXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 7635) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 7641)

NextSibling (see page 7628)

ChildNodes (see page 7635)

84.2.2.2.30 IXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7635)

AttributeNodes (see page 7634)

Refresh (see page 7323)

84.2.2.2.31 IXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

SetAttribute is the write implementation for the Attributes (see page 7634) property.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

Attributes (see page 7634)

SetAttributeNS (see page 7630)

84.2.2.2.32 IXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI (see page 7639) is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 7630)

GetAttributeNS (see page 7622)

AttributeNodes (see page 7634)

84.2.2.2.33 IXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

SetChildValue is the write implementation of the ChildValues (see page 7636) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

Value is the value to assign.

See Also

ChildValues (see page 7636)

84.2.2.2.34 IXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

SetNodeValue is the write implementation of the NodeValue (see page 7641) property.

Value is the value to assign.

See Also

NodeValue (see page 7641)

84.2.2.2.35 IXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly is the write implementation of the ReadOnly (see page 7642) property.

Value is the new read-only status of the node.

See Also

ReadOnly (see page 7642)

84.2.2.2.36 IXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

SetText is the write implementation for the Text (see page 7643) property.

Value is the new text value for the node.

See Also

Text (see page 7643)

84.2.2.2.37 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	IXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 7633)	Transforms the subtree rooted at this node, using a specified XSL interface.
	IXMLNode.TransformNode (IXMLNode, WideString) (see page 7633)	Transforms the subtree rooted at this node, using a specified XSL interface.

84.2.2.2.37.1 IXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Call TransformNode to transform the XML (see page 7643) that is the value of the XML (see page 7643) property using the transformation specified by an XSL document.

stylesheet is a node in an XML (see page 7643) document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString (see page 3476), TransformNode returns a string of XML (see page 7643). If output is the interface to an XML (see page 7643) document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML (see page 7643) document is MSXML.

See Also

XML (see page 7643)

84.2.2.3 IXMLNode Properties

The properties of the IXMLNode class are listed here.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.
R	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7639)	Indicates the node name.
R	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
R	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
R	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
R	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
R	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

84.2.2.3.1 IXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

Use AttributeNodes to access the attributes of this node using an IXMLNode (see page 7615) interface. Attributes (see page 7634) modify a node in an XML (see page 7643) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

If an attribute consists of a simple Name=Value pair, it is simpler to use the Attributes (see page 7634) property to read or set attribute values. AttributeNodes is an IXMLNodeList (see page 7649) interface, which can be used to

Read (see page 3727) the count of attributes before iterating over all the attributes of this node.

Add or delete attribute nodes from this node.

Access attributes through an XML (see page 7643) node interface (for example, to read the attribute names from the nodes).

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes (see page 7634) are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 7635) property.

See Also

ChildNodes (see page 7635)

Attributes (see page 7634)

84.2.2.3.2 IXMLNode.Attributes Property

Represents the values of this node's attributes.

Pascal

```
property Attributes [const AttrName: DOMString]: OleVariant;
```

C++

```
__property OleVariant Attributes[DOMString const AttrName];
```

Description

Use Attributes to read or set the values of this node's attributes. Attributes modify a node in an XML (see page 7643) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

The AttrName index gives the name of the attribute to access. In the previous example, "genre" and "title" are the names of the attributes. When setting Attributes, if you specify the name of an attribute that already exists for a node, you change that attribute's value. If you specify an attribute name that does not correspond to an existing attribute of the node, a new attribute is

created.

When reading Attributes, if you specify the name of an attribute that does not exist for the node, the value depends on the Options property of the document that contains the node. If Options includes doAttrNull, reading an attribute that does not exist for the node results in a Null Variant (see page 3430). When Options does not include doAttrNull, reading an attribute that does not exist results in an empty string.

The value of Attributes is the text of the specified attribute. When setting attributes, the value string is not parsed. Any markup in the supplied value (for example, an entity reference) is treated as literal text. If you set an attribute to nil (Delphi) or NULL (C++), it is removed from this node. Note that you can set an attribute's value to an empty string.

See Also

AttributeNodes (see page 7634)

GetAttributeNS (see page 7622)

SetAttributeNS (see page 7630)

Options (see page 7341)

Working with XML Nodes

84.2.2.3.3 IXMLNode.ChildNodes Property

Represents the child nodes of the node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access the children of a node. ChildNodes is an IXMLNodeList (see page 7649) interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Tip: To get or set the values of the child nodes rather than their interfaces, use the ChildValues (see page 7636) property instead.

Note: Attributes (see page 7634) are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes (see page 7634) or AttributeNodes (see page 7634) property instead.

See Also

ChildValues (see page 7636)

AttributeNodes (see page 7634)

ParentNode (see page 7641)

Attributes (see page 7634)

Working with XML Nodes

84.2.2.3.4 IXMLNode.ChildValues Property

Represents the values of this node's child nodes.

Pascal

```
property ChildValues [const IndexOrName: OleVariant]: OleVariant;
```

C++

```
__property OleVariant ChildValues[OleVariant const IndexOrName];
```

Description

Use ChildValues to access the values of this node's children.

IndexOrName identifies the desired child node either by its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, trying to get or set ChildValues causes an exception.

The value of a child node depends on the type of the child node, as indicated in the following table:

NodeType (see page 7640)	Value
ntElement	If the element contains only text, this is that text value. Otherwise, it is the Value of the first child node.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, reading or setting ChildValues causes an exception.	

Note: In Delphi, ChildValues is the default property for IXMLNode (see page 7615). That means that you can omit the property name when using this property. For example

```
Edit1.Text := XMLDocument1.DocumentElement.ChildValues['Name'];
```

can also be written

```
Edit1.Text := XMLDocument1.DocumentElement['Name'];
```

Note: Attributes (see page 7634) are not considered child nodes of a node. To get or set the values of this node's attributes, use the Attributes (see page 7634) property instead.

See Also

ChildNodes (see page 7635)

ParentNode (see page 7641)

Attributes (see page 7634)

84.2.2.3.5 IXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: IXMLNodeCollection;
```

C++

```
__property IXMLNodeCollection Collection;
```

Description

Read (see page 3727) Collection to determine whether this node is one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML (see page 7643) document as this node's ParentNode (see page 7641) property, but it is specialized to work with this node and any other child nodes of the same type.

For example, consider the following XML (see page 7643) fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 7641) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

If this node does not represent one of a set of repeating child elements, Collection is nil (Delphi) or NULL (C++).

See Also

ParentNode (see page 7641)

84.2.2.3.6 IXMLNode.DOMNode Property

Provides access to the IDOMNode (see page 7509) interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode (see page 7509) is the interface for this node provided by the DOM implementation that parsed the XML (see page 7643) document. You can use this interface to work with the node using the lower-level properties and methods of the DOM.

Typically, applications use the IXMLNode (see page 7615) interface to work with a node rather than the IDOMNode (see page 7509) interface. For more information on DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

84.2.2.3.7 IXMLNode.HasChildNodes Property

Indicates whether this node has any child nodes.

Pascal

```
property HasChildNodes: Boolean;
```

C++

```
__property Boolean HasChildNodes;
```

Description

HasChildNodes is true if the underlying DOM node has any child nodes, false if it has no children.

Note that in the underlying DOM implementation, given the following XML (see page 7643)

```
<Title> Understanding XML </Title>
```

the "Title" node has a single child (a text node with the value 'Understanding XML (see page 7643)').

See Also

ChildNodes (see page 7635)

DOMNode (see page 7637)

84.2.2.3.8 IXMLNode.IsTextElement Property

Indicates whether the node has a single text value.

Pascal

```
property IsTextElement: Boolean;
```

C++

```
__property Boolean IsTextElement;
```

Description

Read (see page 3727) IsTextElement to determine whether this node represents a single text value. For example, if the node represents the following:

```
<Title> Understanding XML </Title>
```

IsTextElement is true, because the node represents a tagged element and the text that is its value.

Note: In the underlying DOM implementation, nodes for which IsTextElement is true are element nodes that have a single child node that is a text node. IXMLNode (see page 7615) flattens this out so that you can use the element node to work with the text value (through the Text (see page 7643) property), rather than requiring you to use a separate node for the text value.

See Also

NodeType (see page 7640)

Text (see page 7643)

Working with XML Nodes

84.2.2.3.9 IXMLNode.LocalName Property

Indicates the name of the node without any namespace prefix.

Pascal

```
property LocalName: DOMString;
```

C++

```
__property DOMString LocalName;
```

Description

Read (see page 3727) LocalName to determine the local name of the underlying DOM node. LocalName is only defined for

nodes of type ntElement or ntAttrribute.

For element nodes, LocalName is derived from the name that appears in the XML (see page 7643) tag. For example, the local name for <xsi:Name First="John" Last="Doe"> is 'Name'. (The string "xsi" is called the prefix).

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example, the element <xsi:Name xsi:First="John" xsi:Last="Doe"> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, localName is nil (Delphi) or NULL (C++).

To get the name of a node when localName is not defined, use the NodeName (see page 7639) property instead. If the node's name has no namespace prefix, LocalName has the same value as NodeName (see page 7639).

See Also

NamespaceURI (see page 7639)

Prefix (see page 7642)

NodeName (see page 7639)

84.2.2.3.10 IXMLNode.NamespaceURI Property

Identifies the URI for the namespace used in the qualified node name.

Pascal

```
property NamespaceURI: DOMString;
```

C++

```
__property DOMString NamespaceURI;
```

Description

NamespaceURI is the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, NamespaceURI is an empty string.

The namespace URI is only available if it is explicitly assigned when the node is created. By default, IXMLNode (see page 7615) does not assign namespaces to the new nodes it creates.

See Also

NodeName (see page 7639)

NodeType (see page 7640)

Prefix (see page 7642)

LocalName (see page 7638)

CreateDOMNode (see page 7439)

NamespaceURI (see page 7521)

84.2.2.3.11 IXMLNode.NodeName Property

Indicates the node name.

Pascal

```
property NodeName: DOMstring;
```

C++

```
__property DOMstring NodeName;
```

Description

NodeName is the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType (see page 7640)	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'
ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If Prefix (see page 7642) and LocalName (see page 7638) are both defined for a node, the node name is the combination of these two values.

See Also

NodeType (see page 7640)

Prefix (see page 7642)

LocalName (see page 7638)

84.2.2.3.12 IXMLNode.NodeType Property

Indicates the type of the node.

Pascal

```
property NodeType: TNodeType;
```

C++

```
__property TNodeType NodeType;
```

Description

Read (see page 3727) NodeType to determine the type of the node. The type reflects the way the node is used in the XML (see page 7643) document.

See Also

IsTextElement (see page 7638)

84.2.2.3.13 IXMLNode.NodeValue Property

Specifies the value of this node.

Pascal

```
property NodeValue: OleVariant;
```

C++

```
__property OleVariant NodeValue;
```

Description

Use NodeValue to get or set the value of the node. The meaning of a node's value depends on the type of the node, as indicated in the following table:

NodeType (see page 7640)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, trying to read or set NodeValue causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.

Trying to read or set the value of any other node type causes an exception.

See Also

ChildValues (see page 7636)

Text (see page 7643)

84.2.2.3.14 IXMLNode.OwnerDocument Property

Identifies the document in which this node appears.

Pascal

```
property OwnerDocument: IXMLDocument;
```

C++

```
__property IXMLDocument OwnerDocument;
```

Description

Read (see page 3727) OwnerDocument to obtain the IXMLDocument (see page 7586) interface of the document to which this node belongs.

See Also

ParentNode (see page 7641)

84.2.2.3.15 IXMLNode.ParentNode Property

Identifies the interface of the node's parent node.

Pascal

```
property ParentNode: IXMLNode;
```

C++

```
__property IXMLNode ParentNode;
```

Description

Read (see page 3727) ParentNode to access the node's parent node.

See Also

ChildNodes (see page 7635)

84.2.2.3.16 IXMLNode.Prefix Property

Identifies the namespace prefix of the node's name.

Pascal

```
property Prefix: DOMString;
```

C++

```
__property DOMString Prefix;
```

Description

Prefix is the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML (see page 7643) tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

See Also

LocalName (see page 7638)

NodeType (see page 7640)

NamespaceURI (see page 7639)

84.2.2.3.17 IXMLNode.ReadOnly Property

Specifies whether the node can be modified.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

ReadOnly specifies whether the value, child nodes, or attributes of the node can be changed. When ReadOnly is true, the node can't be modified. Read (see page 3727)-only nodes raise an exception when an attempt is made to modify them.

When ReadOnly is false, the node can be modified.

See Also

AttributeNodes (see page 7634)

ChildNodes (see page 7635)

NodeValue (see page 7641)

84.2.2.3.18 IXMLNode.Text Property

Represents the Text of the node.

Pascal

```
property Text: DOMString;
```

C++

```
__property DOMString Text;
```

Description

Text represents the text of the node. For example, if the node corresponds to

```
<Title> Understanding XML </Title>
```

The text of the node is the string 'Understanding XML (see page 7643)'.

Text is intended for use nodes where the IsTextElement (see page 7638) property is true. If IsTextElement (see page 7638) is false, then

If the node has no children, the value of Text is an empty string. Setting Text is this case result in a node where IsTextElement (see page 7638) istrue.

If the node has children (other than a single DOM text node), reading or setting Text causes an exception.

See Also

IsTextElement (see page 7638)

NodeValue (see page 7641)

Working with XML Nodes

84.2.2.3.19 IXMLNode.XML Property

Indicates the XML that corresponds to the subtree rooted at this node.

Pascal

```
property XML: DOMString;
```

C++

```
__property DOMString XML;
```

Description

XML is the XML that corresponds to the document fragment that consists of this node and any child nodes it contains.

Note: XML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

Text (see page 7643)

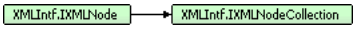
ChildValues (see page 7636)

DOMVendor (see page 7337)

84.2.3 IXMLNodeCollection Interface

IXMLNodeCollection is the interface for a set of child nodes that are all the same type.

Class Hierarchy



Pascal

```
IXMLNodeCollection = interface(IXMLNode);
```

C++

```
__interface IXMLNodeCollection : public IXMLNode;
```

File

XMLIntf

Description

Use IXMLNodeCollection to work with repeating elements in an XML (see page 7643) document. The XML (see page 7643) Data Binding wizard generates descendants of IXMLNodeCollection interface to represent sets of child nodes that are all of a particular type.

IXMLNodeCollection is a descendant of IXMLNode (see page 7615). As such, it provides generic support for working with a node in an XML (see page 7643) document. In addition, IXMLNodeCollection introduces properties and methods that manage a set of child nodes that are all the same type of node (that all have the same tag names or that are defined as the same type in an XML (see page 7643) schema).

Note: C++ method declarations that use IXMLNodeCollection use the _di_IXMLNodeCollection type instead. This type is a DelphiInterface (see page 3263) wrapper around the IXMLNodeCollection interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeCollection >
_di_IXMLNodeCollection;
```

See Also

TXMLDocument (see page 7299)

IXMLNode (see page 7615)

TXMLNodeCollection (see page 7396)

84.2.3.1 IXMLNodeCollection Members

The following tables list the members exposed by IXMLNodeCollection.

Public Methods

	Name	Description
	AddChild (see page 7618)	This is the overview for the AddChild method overload.
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.

	CloneNode (see page 7619)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7620)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7620)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7620)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7621)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7621)	Returns the value of the AttributeNodes (see page 7634) property.
	GetAttributeNS (see page 7622)	Returns the value of a specified attribute.
	GetChildNodes (see page 7622)	Returns the value of the ChildNodes (see page 7635) property.
	GetChildValue (see page 7622)	Returns the value of a specified child node.
	GetCollection (see page 7623)	Returns the value of the Collection (see page 7636) property.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetDOMNode (see page 7623)	Returns the value of the DOMNode (see page 7637) property.
	GetHasChildNodes (see page 7624)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7624)	Indicates whether the node has a single text value.
	GetLocalName (see page 7624)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7625)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7625)	Returns the name of the node.
	GetNodeType (see page 7625)	Returns the type of the node.
	GetNodeValue (see page 7626)	Returns the value of the node.
	GetOwnerDocument (see page 7626)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7626)	Returns the interface for the node's parent node.
	GetPrefix (see page 7626)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7627)	Returns the value of the ReadOnly (see page 7642) property.
	GetText (see page 7627)	Returns the text value of the node.
	GetXML (see page 7627)	Returns the value of the XML (see page 7643) property.
	HasAttribute (see page 7628)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7628)	Returns the next child of this node's parent.
	Normalize (see page 7629)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7629)	Returns the previous child of this node's parent.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.
	Resync (see page 7630)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7630)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7630)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7631)	Sets the value of a specified child node.
	SetNodeValue (see page 7631)	Sets the value of this node.
	SetReadOnly (see page 7632)	Specifies whether the node can be modified.
	SetText (see page 7632)	Sets the text value of the node.
	TransformNode (see page 7632)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7634)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7634)	Represents the values of this node's attributes.
R	ChildNodes (see page 7635)	Represents the child nodes of the node.
	ChildValues (see page 7636)	Represents the values of this node's child nodes.
R	Collection (see page 7636)	Indicates the collection in which this node appears.
R	Count (see page 7648)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7637)	Provides access to the IDOMNode (see page 7509) interface for this node.
R	HasChildNodes (see page 7637)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7638)	Indicates whether the node has a single text value.

	LocalName (see page 7638)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7639)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7639)	Indicates the node name.
	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7640)	Indicates the type of the node.
	NodeValue (see page 7641)	Specifies the value of this node.
	OwnerDocument (see page 7641)	Identifies the document in which this node appears.
	ParentNode (see page 7641)	Identifies the interface of the node's parent node.
	Prefix (see page 7642)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7642)	Specifies whether the node can be modified.
	Text (see page 7643)	Represents the Text of the node.
	XML (see page 7643)	Indicates the XML that corresponds to the subtree rooted at this node.

84.2.3.2 IXMLNodeCollection Methods

The methods of the IXMLNodeCollection class are listed here.

Public Methods

	Name	Description
	Clear (see page 7646)	Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.
	Delete (see page 7646)	Removes one of the repeating child nodes from this node.
	GetCount (see page 7647)	Returns the value of the Count (see page 7648) property.
	GetNode (see page 7647)	Returns the interface for one of the repeating child nodes.
	Remove (see page 7648)	Removes one of the repeating child nodes from this node.

84.2.3.2.1 IXMLNodeCollection.Clear Method

Removes all child nodes in the Nodes (see page 7649) property from the ChildList and Nodes (see page 7649) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear deletes every child node from the collection (all the nodes in the Nodes (see page 7649) property), along with the corresponding nodes in the ChildNodes (see page 7635) property.

See Also

Nodes (see page 7649)

Delete (see page 7646)

Remove (see page 7648)

ChildNodes (see page 7635)

84.2.3.2.2 IXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the Nodes (see page 7649) property list of repeating child nodes. The index 0 represents the first repetition, 1 represents the second repetition, and so on.

See Also

Remove (see page 7648)

AddChild (see page 7618)

Nodes (see page 7649)

Clear (see page 7646)

84.2.3.2.3 IXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 7648) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7648) property.

See Also

Count (see page 7648)

84.2.3.2.4 IXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

GetNode is the read implementation of the Nodes (see page 7649) property.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

See Also

Nodes (see page 7649)

84.2.3.2.5 IXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes a specified child node from the collection of repeating child nodes. It removes the node from both the collection, and from the ChildNodes (see page 7635) property, which lists all children of this node.

Node is the interface of the child to remove. It must represent on of the repeating child nodes.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 7635) property instead.

See Also

Delete (see page 7646)

ChildNodes (see page 7635)

Nodes (see page 7649)

84.2.3.3 IXMLNodeCollection Properties

The properties of the IXMLNodeCollection class are listed here.

Public Properties

	Name	Description
	Count (see page 7648)	Indicates the number of repeating child nodes.
	Nodes (see page 7649)	Lists the interfaces of the repeating child nodes.

84.2.3.3.1 IXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to obtain the number of repeating children of this node. Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in the Nodes (see page 7649) property list).

Count is changed when you add child nodes using the AddChild (see page 7618) method or delete them using the Delete (see page 7646) or Remove (see page 7648) method. You can access the individual repeating child nodes using the Nodes (see page 7649) property.

See Also

Nodes (see page 7649)

AddChild (see page 7618)

Remove (see page 7648)

Delete (see page 7646)

84.2.3.3.2 IXMLNodeCollection.Nodes Property

Lists the interfaces of the repeating child nodes.

Pascal

```
property Nodes [Index: Integer]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[int Index];
```

Description

Use Nodes to access nodes from the set of repeating child nodes. The value of Nodes is the IXMLNode (see page 7615) interface that represents a specified child node.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

Note: The XML (see page 7643) Data Binding wizard generates a nearly identical property in IXMLNodeCollection (see page 7644) descendants called Items. Items differs from Nodes in that the value is an IXMLNode (see page 7615) descendant specific to the type of the repeating element, rather than the generic IXMLNode (see page 7615) interface.

Note: In Delphi, ChildNodes (see page 7635) is the default property for IXMLNodeCollection (see page 7644). That means that you can omit the property name when using this property. In IXMLNodeCollection (see page 7644) descendants that are generated by the XML (see page 7643) Data Binding wizard, the Items property is the default property instead.

Note: As a descendant of IXMLNode (see page 7615), IXMLNodeCollection (see page 7644) also inherits the ChildNodes (see page 7635) property, which lists all the child nodes for this node. Nodes differs from ChildNodes (see page 7635) in that it contains only those child nodes that are in the repeating collection.

See Also

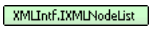
Count (see page 7648)

ChildNodes (see page 7635)

84.2.4 IXMLNodeList Interface

IXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy



Pascal

```
IXMLNodeList = interface;
```

C++

```
__interface IXMLNodeList;
```

File

XMLIntf

Description

Use IXMLNodeList to work with a set of related nodes in an XML document. IXMLNodeList can be used to add or delete nodes from the set, or to access specific nodes.

Note: C++ method declarations that use IXMLNodeList use the _di_IXMLNodeList type instead. This type is a DelphiInterface (see page 3263) wrapper around the IXMLNodeList interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeList >  _di_IXMLNodeList;
```

See Also

TXMLNodeList (see page 7411)

IXMLNode (see page 7615)

IXMLNode (see page 7615)

TXMLDocument (see page 7299)

84.2.4.1 IXMLNodeList Members

The following tables list the members exposed by IXMLNodeList.

Public Methods

	Name	Description
	Add (see page 7651)	Adds a new node to the end of the list.
	BeginUpdate (see page 7651)	Increments the value of UpdateCount (see page 7661).
	Clear (see page 7652)	Removes all nodes from the list.
	Delete (see page 7652)	This is the overview for the Delete method overload.
	EndUpdate (see page 7653)	Decrements the value of UpdateCount (see page 7661).
	FindNode (see page 7654)	This is the overview for the FindNode method overload.
	FindSibling (see page 7655)	Returns a node that appears a specified amount before or after another node.
	First (see page 7655)	Returns the first node in the list.
	Get (see page 7656)	Returns a specified node in the list.
	GetCount (see page 7656)	Returns the number of nodes in the list.
	GetNode (see page 7656)	Returns a specified node from the list.
	GetUpdateCount (see page 7657)	Returns the value of the UpdateCount (see page 7661) property.
	IndexOf (see page 7657)	This is the overview for the IndexOf method overload.
	Insert (see page 7658)	Inserts a new node into a specified position in the list.
	Last (see page 7658)	Returns the last node in the list.
	Remove (see page 7659)	Removes a specified node from the list.
	ReplaceNode (see page 7659)	Replaces a specified node in the list with another node.

Public Properties

	Name	Description
R	Count (see page 7660)	Indicates the number of nodes in the list.
R	Nodes (see page 7660)	Provides access to the nodes in the list.
R	UpdateCount (see page 7661)	Indicates the number of times an application has called BeginUpdate (see page 7651) without a matching call to EndUpdate (see page 7653).

84.2.4.2 IXMLNodeList Methods

The methods of the IXMLNodeList class are listed here.

Public Methods

	Name	Description
	Add (see page 7651)	Adds a new node to the end of the list.
	BeginUpdate (see page 7651)	Increments the value of UpdateCount (see page 7661).
	Clear (see page 7652)	Removes all nodes from the list.
	Delete (see page 7652)	This is the overview for the Delete method overload.
	EndUpdate (see page 7653)	Decrements the value of UpdateCount (see page 7661).
	FindNode (see page 7654)	This is the overview for the FindNode method overload.
	FindSibling (see page 7655)	Returns a node that appears a specified amount before or after another node.
	First (see page 7655)	Returns the first node in the list.
	Get (see page 7656)	Returns a specified node in the list.
	GetCount (see page 7656)	Returns the number of nodes in the list.
	GetNode (see page 7656)	Returns a specified node from the list.
	GetUpdateCount (see page 7657)	Returns the value of the UpdateCount (see page 7661) property.
	IndexOf (see page 7657)	This is the overview for the IndexOf method overload.
	Insert (see page 7658)	Inserts a new node into a specified position in the list.
	Last (see page 7658)	Returns the last node in the list.
	Remove (see page 7659)	Removes a specified node from the list.
	ReplaceNode (see page 7659)	Replaces a specified node in the list with another node.

84.2.4.2.1 IXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 7653)

Remove (see page 7659)

IndexOf (see page 7657)

Insert (see page 7658)

Nodes (see page 7660)

84.2.4.2.2 IXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 7661).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 7661) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 7653) to decrement the UpdateCount (see page 7661) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 7653), so that the UpdateCount (see page 7661) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7661)

EndUpdate (see page 7653)

84.2.4.2.3 IXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 7651) and EndUpdate (see page 7653) methods, even though it may result in the deletion of more than one node.

See Also

Delete (see page 7653)

Remove (see page 7659)

Nodes (see page 7660)

84.2.4.2.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	IXMLNodeList.Delete (DOMString) (see page 7653)	Removes a specified node from the list.
	IXMLNodeList.Delete (DOMString, DOMString) (see page 7653)	Removes a specified node from the list.

	IXMLNodeList.Delete (Integer) (see page 7653)	Removes a specified node from the list.

84.2.4.2.4.1 IXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name (if necessary).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 7660) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 7659)

Clear (see page 7652)

IndexOf (see page 7657)

LocalName (see page 7638)

NamespaceURI (see page 7639)

Working with XML Nodes

84.2.4.2.5 IXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 7661).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 7661) property, which was incremented by a previous call to BeginUpdate (see page 7651).

Every call to BeginUpdate (see page 7651) should be matched by a call to EndUpdate, so that the UpdateCount (see page

7661) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7661)

BeginUpdate (see page 7651)

84.2.4.2.6 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	IXMLNodeList.FindNode (DOMString) (see page 7654)	Returns a specified node from the list.
	IXMLNodeList.FindNode (DOMString, DOMString) (see page 7654)	Returns a specified node from the list.
	IXMLNodeList.FindNode (TGuid) (see page 7654)	Returns a specified node from the list.

84.2.4.2.6.1 IXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 7655)

First (see page 7655)

Last (see page 7658)

IndexOf (see page 7657)

LocalName (see page 7638)

Nodes (see page 7660)

NamespaceURI (see page 7639)

84.2.4.2.7 IXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 7654)

First (see page 7655)

Last (see page 7658)

IndexOf (see page 7657)

84.2.4.2.8 IXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 7654)

FindSibling (see page 7655)

Last (see page 7658)

Nodes (see page 7660)

84.2.4.2.9 IXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 7660) property.

See Also

Nodes (see page 7660)

GetNode (see page 7656)

Count (see page 7660)

84.2.4.2.10 IXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7660) property.

See Also

Count (see page 7660)

84.2.4.2.11 IXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode is the read implementation of the Nodes (see page 7660) property.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 7654)

Nodes (see page 7660)

LocalName (see page 7638)

Get (see page 7656)

84.2.4.2.12 IXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 7661) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 7661) property.

See Also

UpdateCount (see page 7661)

84.2.4.2.13 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	IXMLNodeList.IndexOf (DOMString) (see page 7657)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (DOMString, DOMString) (see page 7657)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (IXMLNode) (see page 7657)	Returns the index of a specified node.

84.2.4.2.13.1 IXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
```

```
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName (if necessary).

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 7654)

LocalName (see page 7638)

Nodes (see page 7660)

NamespaceURI (see page 7639)

84.2.4.2.14 IXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 7653)

Remove (see page 7659)

IndexOf (see page 7657)

Add (see page 7651)

Nodes (see page 7660)

84.2.4.2.15 IXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 7654)

FindSibling (see page 7655)

First (see page 7655)

Nodes (see page 7660)

84.2.4.2.16 IXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 7653)

Clear (see page 7652)

84.2.4.2.17 IXMLNodeList.ReplaceNode Method

Replaces a specified node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

Remove (see page 7659)

Insert (see page 7658)

84.2.4.3 IXMLNodeList Properties

The properties of the IXMLNodeList class are listed here.

Public Properties

	Name	Description
R	Count (see page 7660)	Indicates the number of nodes in the list.
R	Nodes (see page 7660)	Provides access to the nodes in the list.
R	UpdateCount (see page 7661)	Indicates the number of times an application has called BeginUpdate (see page 7651) without a matching call to EndUpdate (see page 7653).

84.2.4.3.1 IXMLNodeList.Count Property

Indicates the number of nodes in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3727) Count to determine the number of nodes in the list. Count provides an upper limit to the indexes that can be used with the Nodes (see page 7660) property.

To change the value of Count, add nodes using the Add (see page 7651) or Insert (see page 7658) method, or remove them using the Delete (see page 7653) or Remove (see page 7659) method.

See Also

Nodes (see page 7660)

Add (see page 7651)

Delete (see page 7653)

Remove (see page 7659)

Insert (see page 7658)

84.2.4.3.2 IXMLNodeList.Nodes Property

Provides access to the nodes in the list.

Pascal

```
property Nodes [const IndexOrName: OleVariant]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[OleVariant const IndexOrName];
```

Description

Read (see page 3727) Nodes to access a specified node in the list.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on. The Count (see page 7660) property provides an upper bound on the indexes you can specify.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, and if the document that contains this node list's parent includes doNodeAutoCreate in its Options property, then the node list tries to create a new node with the name specified by IndexOrName. If the node list can't create the new node, it raises an exception.

See Also

Count (see page 7660)

FindNode (see page 7654)

LocalName (see page 7638)

Options (see page 7341)

84.2.4.3.3 IXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 7651) without a matching call to EndUpdate (see page 7653).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 7651), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 7653), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 7651) and EndUpdate (see page 7653) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0.

See Also

BeginUpdate (see page 7651)

EndUpdate (see page 7653)

84.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeType (see page 7662)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 7664)	TParseOption and TParseOptions (see page 7667) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 7664)	TXMLDocOption and TXMLDocOptions (see page 7667) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 7665)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

84.3.1 XMLIntf.TNodeType Enumeration

TNodeType identifies the type of a node in an XML document.

Pascal

```
TNodeType = (
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
);
```

C++

```
enum TNodeType {
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
};
```

File

XMLIntf

Description

TNodeType identifies the type of a node in an XML document. The following table lists the possible values:

Value	Meaning
ntReserved	Not used
ntElement	The node represents an element. Element nodes represent simple tags that have child nodes. (Note that sometimes these child nodes do not appear when using IXMLNode (see page 7615). For example child nodes of type ntText are typically hidden by IXMLNode (see page 7615) and appear only as the value of the Text property). The child nodes of an element node can have the following node types: ntElement, ntText, ntCData, ntEntityRef, ntProcessingInstr, and ntComment. Element nodes can also have attributes (ntAttribute). Element nodes can be the child of a node of type ntDocument, ntDocFragment, ntEntityRef, and ntElement.
ntAttribute	The node represents an attribute of an element. It is not the child of another node, but its value can be accessed using the Attribute property of the element node's IXMLNode (see page 7615) interface. An attribute node can have child nodes of type ntText and ntEntityRef.
ntText	The node represents the text content of a tag. A text node cannot have any child nodes, but can appear as the child node of a node of type ntAttribute, ntDocFragment, ntElement, or ntEntityRef.
ntCData	The node represents a CDATA section in the XML source. CDATA sections identify blocks of text that would otherwise be interpreted as markup. An ntCData node can't have any child nodes. They can appear as the child of an ntDocFragment, ntEntityRef, or ntElement node.
ntEntityRef	The node represents a reference to an entity in the XML document. This can be any type of entity, including character entity references. The children of an entity reference node can be of the following types: ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. The entity reference node can appear as the child of an ntAttribute, ntDocFragment, ntElement, or ntEntityRef node.
ntEntity	The node represents an expanded entity. Entity nodes can have child nodes that represent the expanded entity (for example, ntText and ntEntityRef nodes). Entity nodes only appear as the child of an ntDocType node.
ntProcessingInstr	The node represents a processing instruction (PI) from the XML document. A PI node cannot have any child nodes, but can appear as the child node of a node of type ntDocument, ntDocFragment, ntElement, or ntEntityRef.
ntComment	The node represents a comment in the XML document. Comment nodes do not have child nodes. They appear as the child of an ntDocument, ntDocFragment, ntElement, or ntEntityRef node.
ntDocument	The node represents a document object, which is the root of the entire XML document. Document nodes have a single ntElement node as a child (the DocumentElement). In addition, they can have child nodes of type ntProcessingInstr, ntComment, and ntDocType. Because the document is the root of the XML document, it never appears as a child node.
ntDocType	The node represents the document type declaration, indicated by the <!DOCTYPE > tag. The document type node can child nodes of type ntNotation and ntEntity. It always appears as the child of the document node.
ntDocFragment	The node represents a document fragment. A document fragment node associates a node or subtree with a document without actually being contained in the document. Document fragment nodes can have child nodes of type ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. It never appears as the child of another node.
ntNotation	A node represents a notation in the document type declaration. It always appears as the child of an ntDocType node and never has any child nodes.

84.3.2 XMLIntf.TParseOption Enumeration

TParseOption and TParseOptions (see page 7667) represent implementation-specific options that are followed when parsing the XML document.

Pascal

```
TParseOption = (
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
);
```

C++

```
enum TParseOption {
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
};
```

File

XMLIntf

Description

TParseOptions (see page 7667) is a set of TParseOption values. Each TParseOption value provides information about how a DOM parser should parse an XML document. Different DOM implementations support different options.

The following table lists the possible values:

OptionDOM VendorMeaning

Option	DOM Vendor	Meaning
poResolveExternals	MSXML	External definitions (resolvable namespaces, DTD external subsets, and external entity references) are resolved at parse time.
poValidateOnParse	MSXML	The DOM parser validates the XML document against its schema information as well as verifying that it is well-formed XML.
poPreserveWhiteSpace	MSXML	White space in the text of the XML document is not stripped off.
poAsyncLoad	MSXML	The DOM parser works asynchronously. This can speed up performance, especially when parsing a large XML document, but can also lead to exceptions if the application tries to access nodes in the document before it is completely parsed.

84.3.3 XMLIntf.TXMLDocOption Enumeration

TXMLDocOption and TXMLDocOptions (see page 7667) indicate types of information that an XML document automatically supplies.

Pascal

```
TXMLDocOption = (
```

```
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  );
```

C++

```
  enum TXMLDocOption {
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  };
```

File

XMLIntf

Description

TXMLDocOptions (see page 7667) is a set of TXMLDocOption values. The following table lists the TXMLDocOption values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7649) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (see page 3430) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild creates the new namespace but does not add an attribute to declare it.
doAutoSave	When the XML document closes, any changes are automatically saved back to the XML document file or to the XML property.

84.3.4 XMLIntf.TXMLEncodingType Enumeration

TXMLEncodingType indicates the character set used to encode characters in an XML document.

Pascal

```
  TXMLEncodingType = (
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
```

```
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  );
```

C++

```
  enum TXMLEncodingType {
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  };
```

File

XMLIntf

Description

The following table lists the possible TXMLEncodingType values:

Value	Character encoding
xetUnknown	The character set is unknown.
xetUCS_4BE	UCS-4, big-endian (1234).
xetUCS_4LE	UCS-4, little-endian (4321).
xetUCS_4Order2134	UCS-4, unusual octet order (2134)
xetUCS_4Order3412	UCS-4, unusual octet order (3412)
xetUTF_16BE	UTF-16, big-endian order
xetUTF_16LE	UTF-16, little-endian order
xetUTF_8	UTF-8
xetUCS_4Like	Any UCS-4 encoding
xetUTF_16BELike	Any UTF-16, big-endian encoding, may be without byte order mark
xetUTF_16LELike	Any UTF-16, little-endian encoding, my be without byte order mark
xetUTF_8Like	UTF-8, ISO 646, ASCII, or other 7- or 8-bit encoding, including multi-byte character sets such as Shift-JIS.
xetEBCDICLike	EBCDIC

84.4 Types

The following table lists types in this documentation.

Types

Name	Description
TParseOptions (see page 7667)	This is type XMLIntf.TParseOptions.
TXMLDocOptions (see page 7667)	This is type XMLIntf.TXMLDocOptions.

84.4.1 XMLIntf.TParseOptions Type

Pascal

```
TParseOptions = set of TParseOption;
```

C++

```
TParseOption TParseOptions;
```

File

XMLIntf

Description

This is type XMLIntf.TParseOptions.

84.4.2 XMLIntf.TXMLDocOptions Type

Pascal

```
TXMLDocOptions = set of TXMLDocOption;
```

C++

```
TXMLDocOption TXMLDocOptions;
```

File

XMLIntf

Description

This is type XMLIntf.TXMLDocOptions.

84.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NodeTypeNames (see page 7668)	This is constant XMLIntf.NodeTypeNames.

84.5.1 XMLIntf.NodeTypeNames Constant

Pascal

```
NodeTypeNames: array[TNodeType] of string = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

C++

```
array[TNodeType] of string NodeTypeNames = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

File

XMLIntf

Description

This is constant XMLIntf.NodeTypeNames.

85 Xmlxform Namespace

This is namespace Xmlxform.

Topics

Name	Description
Classes (see page 7670)	The following table lists classes in this documentation.
Types (see page 7705)	The following table lists types in this documentation.

Classes

	Name	Description
	TranslateException (see page 7670)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 7671)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 7683)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 7691)	TXMLTransformProvider provides data from and resolves updates to an XML document.

Types

Name	Description
TRowEvent (see page 7705)	This is type Xmlxform.TRowEvent.
TTranslateEvent (see page 7705)	This is type Xmlxform.TTranslateEvent.

85.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TranslateException (see page 7670)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 7671)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 7683)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 7691)	TXMLTransformProvider provides data from and resolves updates to an XML document.

85.1.1 TranslateException Class

TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.

Class Hierarchy

Pascal

```
TranslateException = class(Exception);
```

C++

```
class TranslateException : public Exception;
```

File

Xmlxform

Description

TranslateException is raised by the transform components (TXMLTransform (see page 7671), TXMLTransformClient (see page 7683), and TXMLTransformProvider (see page 7691)) when they can't find a file that is needed to perform a transformation. The missing file can be the XML document that is to be translated into a data packet, or a transformation file that contains instructions on how to map between a data packet and an XML document.

See Also

TXMLTransform (see page 7671)

TXMLTransformClient (see page 7683)

TXMLTransformProvider (see page 7691)

85.1.2 TXMLTransform Class

TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.

Class Hierarchy

Pascal

```
TXMLTransform = class(TComponent);
```

C++

```
class TXMLTransform : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransform to convert an XML document into a data packet or an XML data packet into an XML document that is not a data packet. You can use a TXMLTransform component by itself, or specify the transformation to be performed on a TXMLTransform component that is part of another component such as TXMLTransformClient (see page 7683) or TXMLTransformProvider (see page 7691).

TXMLTransform uses a special XML document called a transformation, which indicates how to perform the conversion. This transformation is specified by the TransformationFile (see page 7680) or TransformationDocument (see page 7679) property. You can generate a transformation file using the xmlmapper.exe utility.

There are three ways to specify the source document or data packet that TXMLTransform converts:

Use the SourceXmlFile (see page 7679) property if the source document is a file on disk.

Use the SourceXml (see page 7678) property if the source document is an in-memory string of XML.

Use the SourceXmlDocument (see page 7678) property if the source document is available through a DOM interface (IDOMDocument (see page 7482)).

TXMLTransform performs its transformation when you read the Data (see page 7675) property or call the TransformXML (see page 7674) method. Once the source XML has been transformed, you can access the DOM interface for the resulting XML document or data packet using the ResultDocument (see page 7677) property, or read the Data (see page 7675) property to directly access the string of XML.

See Also

TXMLTransformClient (see page 7683)

TXMLTransformProvider (see page 7691)

Converting XML Documents into Data Packets

85.1.2.1 TXMLTransform Members

The following tables list the members exposed by TXMLTransform.

Protected Methods

	Name	Description
	DoTransform (see page 7673)	This is DoTransform, a member of class TXMLTransform.
	GetData (see page 7673)	This is GetData, a member of class TXMLTransform.
	GetResultString (see page 7673)	This is GetResultString, a member of class TXMLTransform.
	Transform (see page 7673)	This is Transform, a member of class TXMLTransform.
	Translate (see page 7674)	This is Translate, a member of class TXMLTransform.

Public Methods

	Name	Description
	TransformXML (see page 7674)	Converts an XML document using the specifications in a transformation file.

Public Properties

	Name	Description
R	Data (see page 7675)	Represents the translated string of XML.
	EmptyDestinationDocument (see page 7676)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
R	ResultDocument (see page 7677)	Specifies the DOM interface for the results of the transformation.
R	ResultString (see page 7677)	Specifies the results of the transformation.
	SourceXml (see page 7678)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 7678)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 7679)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 7679)	Specifies the transformation that indicates how to convert the source XML document.

Published Events

	Name	Description
	AfterEachRow (see page 7681)	Occurs after TXMLTransform (see page 7671) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 7681)	Occurs after TXMLTransform (see page 7671) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 7682)	Occurs immediately before TXMLTransform (see page 7671) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 7682)	Occurs immediately before TXMLTransform (see page 7671) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 7683)	Occurs when the TXMLTransform (see page 7671) component needs to translate a user-defined node.

Published Properties

	Name	Description
	TransformationFile (see page 7680)	Specifies the transformation file that indicates how to convert the source XML document.

85.1.2.2 TXMLTransform Methods

The methods of the TXMLTransform class are listed here.

Protected Methods

	Name	Description
	DoTransform (see page 7673)	This is DoTransform, a member of class TXMLTransform.
	GetData (see page 7673)	This is GetData, a member of class TXMLTransform.
	GetResultString (see page 7673)	This is GetResultString, a member of class TXMLTransform.
	Transform (see page 7673)	This is Transform, a member of class TXMLTransform.
	Translate (see page 7674)	This is Translate, a member of class TXMLTransform.

Public Methods

	Name	Description
	TransformXML (see page 7674)	Converts an XML document using the specifications in a transformation file.

85.1.2.2.1 TXMLTransform.DoTransform Method

Pascal

```
function DoTransform(const XMLSrc: IDOMDocument; const XMLExtract: IDOMDocument; const XMLOut:
IDOMDocument): string;
```

C++

```
__fastcall UnicodeString DoTransform(const IDOMDocument XMLSrc, const IDOMDocument XMLExtract,
const IDOMDocument XMLOut);
```

Description

This is DoTransform, a member of class TXMLTransform.

85.1.2.2.2 TXMLTransform.GetData Method

Pascal

```
function GetData: string;
```

C++

```
__fastcall UnicodeString GetData();
```

Description

This is GetData, a member of class TXMLTransform.

85.1.2.2.3 TXMLTransform.GetResultString Method

Pascal

```
function GetResultString: string;
```

C++

```
__fastcall UnicodeString GetResultString();
```

Description

This is GetResultString, a member of class TXMLTransform.

85.1.2.2.4 TXMLTransform.Transform Method

Pascal

```
procedure Transform(TransNode: IDOMNode; SrcNode: IDOMNode; DestNode: IDOMNode; Count:
Integer; InFromList: TStringList; InDestList: TStringList; InIdStrList: TStringList;
InValueList: TStringList; InOptionalList: TStringList; InDateFormatList: TStringList;
InDateFormatTypeList: TStringList; InMapValuesList: TStringList);
```

C++

```
__fastcall Transform(IDOMNode TransNode, IDOMNode SrcNode, IDOMNode DestNode, int Count,
TStringList InFromList, TStringList InDestList, TStringList InIdStrList, TStringList
InValueList, TStringList InOptionalList, TStringList InDateFormatList, TStringList
InDateFormatTypeList, TStringList InMapValuesList);
```

Description

This is Transform, a member of class TXMLTransform.

85.1.2.2.5 TXMLTransform.TransformXML Method

Converts an XML document using the specifications in a transformation file.

Pascal

```
function TransformXML(const SourceXml: string; const ATransformationFile: string = ''): string;
```

C++

```
__fastcall UnicodeString TransformXML(const UnicodeString SourceXml, const UnicodeString
ATransformationFile = '');
```

Description

Call TransformXML to convert an XML document when the source XML is an in-memory string.

SourceXml (see page 7678) is a string of XML to be transformed. It can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation document used to perform the transformation. If the transformation converts a specific type of XML document into a data packet, SourceXml (see page 7678) must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml (see page 7678) must be an XML data packet with the expected metadata.

ATransformationFile is the transformation file to use when performing the translation. A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the ATransformationFile parameter is omitted, TransformXML uses the file specified by the TransformationFile (see page 7680) property, or the document specified by the TransformationDocument (see page 7679) property.

TransformXML returns the results of the conversion as a string of XML.

See Also

Data (see page 7675)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

85.1.2.2.6 TXMLTransform.Translate Method

Pascal

```
procedure Translate(const Id: string; const SrcNode: IDOMNode; var SrcValue: string; const
DestNode: IDOMNode); dynamic;
```

C++

```
__fastcall Translate(const UnicodeString Id, const IDOMNode SrcNode, UnicodeString SrcValue,
const IDOMNode DestNode);
```

Description

This is Translate, a member of class TXMLTransform.

85.1.2.3 TXMLTransform Properties

The properties of the TXMLTransform class are listed here.

Public Properties

	Name	Description
	Data (see page 7675)	Represents the translated string of XML.
	EmptyDestinationDocument (see page 7676)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
	ResultDocument (see page 7677)	Specifies the DOM interface for the results of the transformation.
	ResultString (see page 7677)	Specifies the results of the transformation.
	SourceXml (see page 7678)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 7678)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 7679)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 7679)	Specifies the transformation that indicates how to convert the source XML document.

Published Properties

	Name	Description
	TransformationFile (see page 7680)	Specifies the transformation file that indicates how to convert the source XML document.

85.1.2.3.1 TXMLTransform.Data Property

Represents the translated string of XML.

Pascal

```
property Data: string;
```

C++

```
__property UnicodeString Data;
```

Description

Read (see page 3727) Data to obtain the results of the transformation. Reading the Data property causes TXMLTransform (see page 7671) to translate its source XML document or data packet into the destination format, which becomes the value of the Data property. The resulting XML string is not cached in memory: reading Data a second time causes TXMLTransform (see page 7671) to perform its transformation a second time.

If the source XML is an XML document and the transformation indicates how to convert it into a data packet, Data is the contents of a data packet in XML format.

If the source XML is an XML data packet and the transformation indicates how to convert it into another type of XML document, Data is the contents of the resulting document.

There are three ways to specify the source document or data packet that is converted to produce Data:

- Use the SourceXmlFile (see page 7679) property if the source document is a file on disk.
- Use the SourceXml (see page 7678) property if the source document is an in-memory string of XML.
- Use the SourceXmlDocument (see page 7678) property if the source document is available through a DOM interface (IDOMDocument (see page 7482)).
- If SourceXMLFile is set to an invalid file name or SourceXMLFile is an empty string and SourceXml (see page 7678) is not a valid string of XML, or SourceXMLFile and SourceXML are both empty strings and SourceXmlDocument (see page 7678) is not set, TXMLTransform (see page 7671) raises a TranslateException (see page 7670) when you try to read Data.
- There are two ways to specify the transformation that converts the source XML into Data:

- Use the TransformationFile (see page 7680) property if the transformation is stored in a file on disk.
- Use the TransformationDocument (see page 7679) method if the transformation is available through a DOM interface (IDOMDocument (see page 7482)).
- If TransformationFile (see page 7680) is set to an invalid file name or TransformationFile (see page 7680) is an empty string and TransformationDocument (see page 7679) is not set, TXMLTransform (see page 7671) raises a TranslateException (see page 7670) when you try to read Data.
- You can augment the transformation by using the EmptyDestinationDocument (see page 7676) property to supply a skeleton document that gives the form for the resulting XML document.
- When you read Data, for every user-defined node in the source XML document, TXMLTransform (see page 7671) generates an OnTranslate (see page 7683) event to convert the node.

Tip: When converting an XML document into a data packet, the value of Data can be assigned to a client dataset's XMLData property.

See Also

SourceXmlFile (see page 7679)

TransformationFile (see page 7680)

TranslateException (see page 7670)

OnTranslate (see page 7683)

XMLData

ResultDocument (see page 7677)

EmptyDestinationDocument (see page 7676)

TransformationDocument (see page 7679)

SourceXml (see page 7678)

Converting XML Documents into Data Packets

85.1.2.3.2 TXMLTransform.EmptyDestinationDocument Property

Specifies the DOM interface for an XML document that defines the structure of the transformation result.

Pascal

```
property EmptyDestinationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument EmptyDestinationDocument;
```

Description

The transformation defined by TransformationFile (see page 7680) or TransformationDocument (see page 7679) contains a default structure for the results of transforming the source document in addition to the mappings between nodes in the source document and nodes in the destination. You can override this default structure, however, by setting EmptyDestinationDocument. This is the DOM interface for an XML document that has the desired structure, but with no values in the nodes that should be filled in using values in the source document. This is useful, for example, when you want to add additional information that does not come from the source document.

EmptyDestinationDocument is not changed when TXMLTransform (see page 7671) performs its transformation. To obtain a DOM interface for the resulting document, use the ResultDocument (see page 7677) property.

See Also

ResultDocument (see page 7677)

Data (see page 7675)

TransformXML (see page 7674)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

85.1.2.3.3 TXMLTransform.ResultDocument Property

Specifies the DOM interface for the results of the transformation.

Pascal

```
property ResultDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ResultDocument;
```

Description

Read (see page 3727) ResultDocument to work with the results of the transformation using a DOM interface. ResultDocument is created when you call the TransformXML (see page 7674) method to perform the transformation.

See Also

SourceXmlDocument (see page 7678)

Data (see page 7675)

TransformXML (see page 7674)

EmptyDestinationDocument (see page 7676)

ResultString (see page 7677)

85.1.2.3.4 TXMLTransform.ResultString Property

Specifies the results of the transformation.

Pascal

```
property ResultString: string;
```

C++

```
__property UnicodeString ResultString;
```

Description

Read (see page 3727) ResultString to obtain the results of the transformation as a string of XML. ResultString is created when you call the TransformXML (see page 7674) method to perform the transformation.

See Also

SourceXmlDocument (see page 7678)

Data (see page 7675)

TransformXML (see page 7674)

EmptyDestinationDocument (see page 7676)

ResultString

85.1.2.3.5 TXMLTransform.SourceXml Property

Specifies a string of XML to be translated.

Pascal

```
property SourceXml: string;
```

C++

```
__property UnicodeString SourceXml;
```

Description

Specify the XML to be translated as the value of SourceXml. SourceXml can be the text of any XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7680) or TransformationDocument (see page 7679). If the transformation converts a specific type of XML document into a data packet, SourceXml must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml is an XML data packet with the expected metadata.

SourceXmlDocument (see page 7678) is used only if the SourceXMLFile property has not been set. If both SourceXMLFile and SourceXml are empty strings, TXMLTransform (see page 7671) uses the SourceXmlDocument (see page 7678) property instead. SourceXml takes precedence over SourceXmlDocument (see page 7678): If SourceXmlDocument (see page 7678) is set, TXMLTransform (see page 7671) still uses SourceXml (even if its value is not a valid string of XML).

See Also

SourceXmlDocument (see page 7678)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

SourceXmlFile (see page 7679)

Converting XML Documents into Data Packets

85.1.2.3.6 TXMLTransform.SourceXmlDocument Property

Specifies the DOM interface for the XML document to be converted.

Pascal

```
property SourceXmlDocument: IDOMDocument;
```

C++

```
__property IDOMDocument SourceXmlDocument;
```

Description

Specify the document to be translated as the value of SourceXmlDocument. SourceXmlDocument can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7680) or TransformationDocument (see page 7679). If the transformation converts a specific type of XML document into a data packet, then SourceXmlDocument is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlDocument is the interface to an XML data packet with the expected metadata.

SourceXmlDocument is used only if the SourceXMLFile and SourceXml (see page 7678) properties have not been set.

See Also

SourceXmlFile (see page 7679)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

SourceXml (see page 7678)

Converting XML Documents into Data Packets

85.1.2.3.7 TXMLTransform.SourceXmlFile Property

Specifies an XML document or XML data packet file to be translated.

Pascal

```
property SourceXmlFile: string;
```

C++

```
__property UnicodeString SourceXmlFile;
```

Description

Specify the file to be translated as the value of SourceXmlFile. SourceXmlFile can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7680) or TransformationDocument (see page 7679). If the transformation converts a specific type of XML document into a data packet, SourceXmlFile is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlFile is an XML data packet with the expected metadata.

If SourceXmlFile is an empty string, TXMLTransform (see page 7671) uses the SourceXml (see page 7678) or SourceXmlDocument (see page 7678) property instead. SourceXmlFile takes precedence over both of these other properties. If SourceXml (see page 7678) or SourceXmlDocument (see page 7678) is set, TXMLTransform (see page 7671) still uses SourceXmlFile (even if it does not specify a valid file name).

See Also

SourceXmlDocument (see page 7678)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

SourceXml (see page 7678)

Converting XML Documents into Data Packets

85.1.2.3.8 TXMLTransform.TransformationDocument Property

Specifies the transformation that indicates how to convert the source XML document.

Pascal

```
property TransformationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument TransformationDocument;
```

Description

TransformationDocument is the DOM interface for the transformation that contains instructions for converting the source XML document into the XML that is the value of Data (see page 7675) and ResultDocument (see page 7677).

A transformation is a special type of XML document that defines the mapping between the nodes of a source document and a destination document and the structure of the destination document. Each transformation is specific to a particular XML schema

and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformations.

If the EmptyDestinationDocument (see page 7676) property is set, TXMLTransform (see page 7671) uses that to override the default structure for the destination document that is defined in TransformationDocument.

TransformationDocument is used only if the value of TransformationFile (see page 7680) is an empty string.

See Also

TransformXML (see page 7674)

TransformationFile (see page 7680)

Data (see page 7675)

EmptyDestinationDocument (see page 7676)

ResultDocument (see page 7677)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

85.1.2.3.9 TXMLTransform.TransformationFile Property

Specifies the transformation file that indicates how to convert the source XML document.

Pascal

```
property TransformationFile: string;
```

C++

```
__property UnicodeString TransformationFile;
```

Description

TransformationFile is a file that contains instructions for converting the source XML document into the XML that is the value of Data (see page 7675) and ResultDocument (see page 7677). If the source document is a general XML document, TransformationFile represents a transformation from that document's schema into a data packet. If the source document is an XML data packet, TransformationFile represents a transformation from that data packet into another type of XML document.

A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the EmptyDestinationDocument (see page 7676) property is set, TXMLTransform (see page 7671) uses that to override the default structure for the destination document that is defined in the transformation file.

TransformationFile takes precedence over the TransformationDocument (see page 7679) property. That is, TXMLTransform (see page 7671) only uses TransformationDocument (see page 7679) if TransformationFile is an empty string.

See Also

TransformXML (see page 7674)

Data (see page 7675)

TransformationDocument (see page 7679)

ResultDocument (see page 7677)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

85.1.2.4 TXMLTransform Events

The events of the TXMLTransform class are listed here.

Published Events

	Name	Description
	AfterEachRow (see page 7681)	Occurs after TXMLTransform (see page 7671) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 7681)	Occurs after TXMLTransform (see page 7671) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 7682)	Occurs immediately before TXMLTransform (see page 7671) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 7682)	Occurs immediately before TXMLTransform (see page 7671) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 7683)	Occurs when the TXMLTransform (see page 7671) component needs to translate a user-defined node.

85.1.2.4.1 TXMLTransform.AfterEachRow Event

Occurs after TXMLTransform (see page 7671) has finished transforming a record in the XML data packet.

Pascal

```
property AfterEachRow: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRow;
```

Description

Write (see page 3789) an AfterEachRow event handler to take specific action after the XML transform component has translated each row in the data packet. For example, you can use AfterEachRow to read the data from the generated row and use it to compute the value of an internally calculated field.

See Also

TransformXML (see page 7674)

AfterEachRowSet (see page 7681)

BeforeEachRow (see page 7682)

Data (see page 7675)

85.1.2.4.2 TXMLTransform.AfterEachRowSet Event

Occurs after TXMLTransform (see page 7671) has finished transforming the entire XML data packet or a nested detail set.

Pascal

```
property AfterEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRowSet;
```

Description

Write (see page 3789) an AfterEachRowSet event handler to take specific action after the XML transform component has translated an entire dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use AfterEachRowSet with the BeforeEachRowSet (see page 7682) and OnTranslate (see page 7683) methods to compute summary values: initialize a variable in the BeforeEachRowSet (see page 7682) method, update it for every OnTranslate (see page 7683) event, and use the result in the AfterEachRowSet event.

See Also

TransformXML (see page 7674)

AfterEachRow (see page 7681)

BeforeEachRow (see page 7682)

Data (see page 7675)

85.1.2.4.3 TXMLTransform.BeforeEachRow Event

Occurs immediately before TXMLTransform (see page 7671) starts transforming a record in the XML data packet.

Pascal

```
property BeforeEachRow: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRow;
```

Description

Write (see page 3789) a BeforeEachRow event handler to take specific action before the XML transform component starts translating each row in the data packet.

See Also

TransformXML (see page 7674)

BeforeEachRowSet (see page 7682)

AfterEachRow (see page 7681)

Data (see page 7675)

85.1.2.4.4 TXMLTransform.BeforeEachRowSet Event

Occurs immediately before TXMLTransform (see page 7671) starts transforming the entire XML data packet or a nested detail set.

Pascal

```
property BeforeEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRowSet;
```

Description

Write (see page 3789) an BeforeEachRowSet event handler to take specific action before the XML transform component starts translating a dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use BeforeEachRowSet with the AfterEachRowSet (see page 7681) and OnTranslate (see page

7683) methods to compute summary values: initialize a variable in the BeforeEachRowSet method, update it for every OnTranslate (see page 7683) event, and use the result in the AfterEachRowSet (see page 7681) event.

See Also

TransformXML (see page 7674)

AfterEachRowSet (see page 7681)

BeforeEachRow (see page 7682)

Data (see page 7675)

85.1.2.4.5 TXMLTransform.OnTranslate Event

Occurs when the TXMLTransform (see page 7671) component needs to translate a user-defined node.

Pascal

```
property OnTranslate: TTranslateEvent;
```

C++

```
__property TTranslateEvent OnTranslate;
```

Description

Write (see page 3789) an OnTranslate event handler to provide the translation for user-defined nodes.

TXMLTransform (see page 7671) performs a translation when an application reads the Data (see page 7675) property or calls the TransformXML (see page 7674) method. If the transformation document that governs how to perform that translation indicates that a node in the XML document is user defined, TXMLTransform (see page 7671) generates an OnTranslate event when it needs to translate to or from that node. OnTranslate only occurs for nodes that are marked as user defined.

If you do not translate a user-defined node in an OnTranslate event handler, the corresponding node in the translated XML is assigned the source XML node's value, if available. Otherwise, it is left blank.

See Also

TransformXML (see page 7674)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

Data (see page 7675)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

85.1.3 TXMLTransformClient Class

TXMLTransformClient acts as an adapter between an XML document and a provider component.

Class Hierarchy

Pascal

```
TXMLTransformClient = class(TComponent);
```

C++

```
class TXMLTransformClient : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransformClient to convert the data from a provider into an XML document or to apply updates in the form of an XML document to a provider component. TXMLTransformClient can act as the client of any provider component, local or remote. It uses a transformation file that is generated using the xmlmapper utility to convert between XML documents and the data packets and delta packets used by the provider.

See Also

TXMLTransform (see page 7671)

TXMLTransformProvider (see page 7691)

Using an XML Document as the Client of a Provider

85.1.3.1 TXMLTransformClient Members

The following tables list the members exposed by TXMLTransformClient.

Protected Methods

	Name	Description
	SetupAppServer (see page 7688)	This is SetupAppServer, a member of class TXMLTransformClient.

Public Methods

	Name	Description
V	ApplyUpdates (see page 7686)	Inserts or deletes the records corresponding to the data in an XML document.
V	Create (see page 7685)	Creates and initializes a TXMLTransformClient (see page 7683) object.
V	Destroy (see page 7685)	Destroys the instance of a TXMLTransformClient (see page 7683) component.
V	GetDataAsXml (see page 7686)	Fetches a data packet from the provider and converts it to an XML document.
	SetParams (see page 7687)	Sets input parameters to values supplied in an XML document.

Published Properties

	Name	Description
	ProviderName (see page 7689)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 7689)	Specifies the connection component that this XML transform client uses to connect to an application server.
R	TransformApplyUpdates (see page 7690)	Specifies the TXMLTransform (see page 7671) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 7690)	Specifies the TXMLTransform (see page 7671) component that converts data packets from the provider into XML.
R	TransformSetParams (see page 7691)	Specifies the TXMLTransform (see page 7671) component that converts the XML supplied to the SetParams (see page 7687) method into a data packet format.

85.1.3.2 TXMLTransformClient.Create Constructor

Creates and initializes a TXMLTransformClient (see page 7683) object.

Pascal

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

C++

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

Description

Call Create to instantiate an XML transform client at runtime. XML transform clients placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the TXMLTransformClient (see page 7683) object.

See Also

Owner

85.1.3.3 TXMLTransformClient.Destroy Destructor

Destroys the instance of a TXMLTransformClient (see page 7683) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformClient();
```

Description

Do not call Destroy directly in an application. Usually destruction of TXMLTransformClient (see page 7683) objects is handled automatically. If an application creates its own instances of TXMLTransformClient (see page 7683) without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

Owner

Create (see page 7685)

System::TObject::Free (see page 3382)

85.1.3.4 TXMLTransformClient Methods

The methods of the TXMLTransformClient class are listed here.

Protected Methods

	Name	Description
	SetupAppServer (see page 7688)	This is SetupAppServer, a member of class TXMLTransformClient.

Public Methods

	Name	Description
	ApplyUpdates (see page 7686)	Inserts or deletes the records corresponding to the data in an XML document.
	GetDataAsXml (see page 7686)	Fetches a data packet from the provider and converts it to an XML document.
	SetParams (see page 7687)	Sets input parameters to values supplied in an XML document.

85.1.3.4.1 TXMLTransformClient.ApplyUpdates Method

Inserts or deletes the records corresponding to the data in an XML document.

Pascal

```
function ApplyUpdates(const UpdateXML: string; const UpdateTransformFile: string; MaxErrors:
Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ApplyUpdates(const UnicodeString UpdateXML, const UnicodeString
UpdateTransformFile, int MaxErrors);
```

Description

Call ApplyUpdates to transform an XML document into a delta packet that inserts or deletes all of the resulting records and send it to the associated provider so that it can be applied. Before you can call ApplyUpdates, you must define the transformation to use by setting properties on the XML transform component specified by TransformApplyUpdates (see page 7690).

UpdateXML is the XML document that can be transformed into a delta packet. Typically, it contains the contents of an XML document file.

UpdateTransformFile is the name of a transformation file that converts UpdateXML into an insert or delete delta packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML schema, data packet format, and update operation (insert or delete). Use the xmlmapper utility to create the transformation file.

Note: If UpdateTransformFile includes any user-defined conversions, ApplyUpdates generates an OnTranslate event on TransformApplyUpdates (see page 7690) when it needs to transform user-defined nodes in the UpdateXML string into corresponding field values.

MaxErrors indicates the maximum number of errors that the provider should allow before prematurely stopping the update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

ApplyUpdates returns the number of errors it encountered.

See Also

ProviderName (see page 7689)

TransformApplyUpdates (see page 7690)

ApplyUpdates

Using an XML Document as the Client of a Provider

85.1.3.4.2 TXMLTransformClient.GetDataAsXml Method

Fetches a data packet from the provider and converts it to an XML document.

Pascal

```
function GetDataAsXml(const PublishTransformFile: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDataAsXml(const UnicodeString PublishTransformFile);
```

Description

Call GetDataAsXml to obtain an XML document that is the translation of a data packet from the associated provider. GetDataAsXml

1. Passes any parameters supplied to the SetParams (see page 7687) method to the provider specified by ProviderName (see page 7689).
2. Obtains a data packet that the provider returns.
3. Uses the transform component specified by TransformGetData (see page 7690) to convert the data packet to an XML document. If the PublishTransformFile parameter is not an empty string, GetDataAsXml uses the file specified by this parameter as the transformation file. Otherwise, it uses the transformation file specified on the TransformGetData (see page 7690) property.
4. Generates an OnTranslate event for every user-defined conversion in its transformation. The application can use this event to perform the user-defined conversion that converts field values into XML nodes.
5. Returns the resulting XML document as a string.

Once you have obtained the resulting, translated XML document, you can assign it to a TXMLDocument component or save it to a file.

Note: Before you can call GetDataAsXml, you must supply any input parameters using the SetParams (see page 7687) method and set properties on TransformGetData (see page 7690) to indicate what transformation to perform.

See Also

ProviderName (see page 7689)

TransformGetData (see page 7690)

XML (see page 7344)

OnTranslate (see page 7683)

SetParams (see page 7687)

Using an XML Document as the Client of a Provider

85.1.3.4.3 TXMLTransformClient.SetParams Method

Sets input parameters to values supplied in an XML document.

Pascal

```
procedure SetParams(const ParamsXml: string; const ParamsTransformFile: string);
```

C++

```
__fastcall SetParams(const UnicodeString ParamsXml, const UnicodeString ParamsTransformFile);
```

Description

Call SetParams to supply input parameter values before fetching data from the provider. Parameter values can represent the input parameters to a query or stored procedure, or they can represent field values that filter the data when the provider represents the data in a TTable or TSQLTable component.

ParamsXML is the XML document that represents the parameter values to set. Typically, it contains the contents of an XML document file.

ParamsTransformFile is the name of a transformation file that converts ParamsXml into a data packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML schema and data packet format. Use the xmlmapper.exe utility to create the transformation file.

Note: If ParamsTransformFile includes any user-defined conversions, the component specified by TransformSetParams (see page 7691) generates an OnTranslate event when it needs to transform user-defined nodes in the ParamsXml string into corresponding parameter values.

The TXMLTransformClient (see page 7683) component stores parameter values internally. Each time you call SetParams, you update those internally stored values. Parameter values from previous calls are not cleared, so be sure to set every parameter value you want to change.

The next the GetDataAsXml (see page 7686) method is called, TXMLTransformClient (see page 7683) sends the currently stored parameter values to the provider.

See Also

GetDataAsXml (see page 7686)

OnTranslate (see page 7683)

ProviderName (see page 7689)

TTable

TSQLTable (see page 1604)

TransformSetParams (see page 7691)

Using an XML Document as the Client of a Provider

85.1.3.4.4 TXMLTransformClient.SetupAppServer Method

Pascal

```
procedure SetupAppServer;
```

C++

```
__fastcall SetupAppServer();
```

Description

This is SetupAppServer, a member of class TXMLTransformClient.

85.1.3.5 TXMLTransformClient Properties

The properties of the TXMLTransformClient class are listed here.

Published Properties

	Name	Description
	ProviderName (see page 7689)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 7689)	Specifies the connection component that this XML transform client uses to connect to an application server.
R	TransformApplyUpdates (see page 7690)	Specifies the TXMLTransform (see page 7671) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 7690)	Specifies the TXMLTransform (see page 7671) component that converts data packets from the provider into XML.

	TransformSetParams (see page 7691)	Specifies the TXMLTransform (see page 7671) component that converts the XML supplied to the SetParams (see page 7687) method into a data packet format.

85.1.3.5.1 TXMLTransformClient.ProviderName Property

Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.

Pascal

```
property ProviderName: string;
```

C++

```
__property UnicodeString ProviderName;
```

Description

Use ProviderName to specify the provider object that supplies the XML transform client with data packets and applies updates on its behalf. This property is passed as a parameter to IAppServer interface calls when the client fetches data from or applies updates to the source dataset, its database server, or the source XML document.

The provider specified by ProviderName can reside in the same application as the XML transform client or in an application server running on another system.

If the provider is in the same application as the XML transform client, it must have the same Owner as the client.

If the provider is on a remote application server, it must be exported by that application server. If the application server contains multiple provider objects, you can set ProviderName to any of them once the XML transform client establishes an initial connection to the application server through its RemoteServer (see page 7689) property.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 7689) property is set if

The connection component is a SOAP connection.

The connection component is a Web connection or socket connection (Windows only).

The application server is COM-based and registered on the client system using TRegsvr.exe (Windows only).

See Also

RemoteServer (see page 7689)

GetDataAsXml (see page 7686)

IAppServer

ApplyUpdates (see page 7686)

Using an XML Document as the Client of a Provider

85.1.3.5.2 TXMLTransformClient.RemoteServer Property

Specifies the connection component that this XML transform client uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

If the XML transform client is in the client portion of a multi-tiered application, use RemoteServer to specify the connection

component that connects to the application server. The connection component locates the application server and obtains its IAppServer interface, which the XML transform client can use for communicating with a remote provider.

If a client application uses more than connection component, the application can change which connection an XML transform client uses by changing the value for this property.

See Also

ProviderName (see page 7689)

TSocketConnection (see page 176)

TDCOMConnection

TWebConnection (see page 209)

TSOAPConnection

AppServer

IAppServer

Using an XML Document as the Client of a Provider

85.1.3.5.3 TXMLTransformClient.TransformApplyUpdates Property

Specifies the TXMLTransform (see page 7671) component that converts XML into delta packets that are then sent to the provider to apply updates.

Pascal

```
property TransformApplyUpdates: TXMLTransform;
```

C++

```
__property TXMLTransform TransformApplyUpdates;
```

Description

Use TransformApplyUpdates to access the subcomponent that transforms the XML passed as the UpdateXML parameter of the ApplyUpdates (see page 7686) method into a delta packet, which is then forwarded to the provider. If the transformation passed to the ApplyUpdates (see page 7686) method includes any user-defined nodes, you must supply TransformApplyUpdates with an OnTranslate event handler.

See Also

ApplyUpdates (see page 7686)

ProviderName (see page 7689)

OnTranslate (see page 7683)

85.1.3.5.4 TXMLTransformClient.TransformGetData Property

Specifies the TXMLTransform (see page 7671) component that converts data packets from the provider into XML.

Pascal

```
property TransformGetData: TXMLTransform;
```

C++

```
__property TXMLTransform TransformGetData;
```

Description

Use TransformGetData to access the subcomponent that performs the transformation of data packets from the provider into another XML format. When an application calls the GetDataAsXml (see page 7686) method, this subcomponent converts the data packet from the provider into XML.

Before you can call GetDataAsXml (see page 7686), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformGetData with an OnTranslate event handler.

See Also

GetDataAsXml (see page 7686)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

OnTranslate (see page 7683)

Using an XML Document as the Client of a Provider

85.1.3.5.5 TXMLTransformClient.TransformSetParams Property

Specifies the TXMLTransform (see page 7671) component that converts the XML supplied to the SetParams (see page 7687) method into a data packet format.

Pascal

```
property TransformSetParams: TXMLTransform;
```

C++

```
__property TXMLTransform TransformSetParams;
```

Description

Use TransformSetParams to directly access the subcomponent that performs the transformation of an XML document into a set of parameter values. When an application calls the SetParams (see page 7687) method, this subcomponent converts the XML supplied as the ParamsXml parameter into a data packet, which it then uses to read and store parameter values. These parameter values are later passed to the provider when you call GetDataAsXml (see page 7686).

If the transformation passed to the SetParams (see page 7687) method includes any user-defined nodes, you must supply TransformSetParams with an OnTranslate event handler.

See Also

SetParams (see page 7687)

ProviderName (see page 7689)

OnTranslate (see page 7683)

GetDataAsXml (see page 7686)

Using an XML Document as the Client of a Provider

85.1.4 TXMLTransformProvider Class

TXMLTransformProvider provides data from and resolves updates to an XML document.

Class Hierarchy

Pascal

```
TXMLTransformProvider = class(TCustomProvider);
```

C++

```
class TXMLTransformProvider : public TCustomProvider;
```

File

Xmlxform

Description

Use TXMLTransformProvider to provide data from an XML document to a client dataset or XML broker and to resolve updates from that client dataset or XML broker back to the source XML document. TXMLTransformProvider can be a part of the same application as the client dataset or XML broker, or it can be placed in the application server of a multi-tiered application. It serves as a translator between an XML document and a client dataset.

TDataSetProvider packages data from the source XML document and passes it in one or more transportable data packets to the client dataset or XML broker.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the source XML document.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the source XML document.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet

TDataSetProvider

TXMLTransform (see page 7671)

TRemoteDataModule

TCRemoteDataModule

TSoapDataModule (see page 1146)

TXMLTransformClient (see page 7683)

TCustomProvider

Determining the Source of Data

Using Provider Components

Using an XML Document as the Source for a Provider

85.1.4.1 TXMLTransformProvider Members

The following tables list the members exposed by TXMLTransformProvider.

Protected Methods

	Name	Description
	InternalApplyUpdates (see page 7694)	Provides the underlying implementation of the ApplyUpdates method.
	InternalGetRecords (see page 7695)	Provides the underlying implementation of the GetRecords method.

Public Methods

	Name	Description
	Create (see page 7693)	Creates and initializes a TXMLTransformProvider (see page 7691) object.
	Destroy (see page 7694)	Frees an instance of a provider object.

Published Events

	Name	Description
	AfterApplyUpdates (see page 7698)	Occurs after the provider applies updates.
	AfterGetRecords (see page 7699)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 7700)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 7701)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 7701)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 7703)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 7703)	Occurs when a provider calls its DataRequest method.

Published Properties

	Name	Description
	CacheData (see page 7696)	Specifies whether the provider caches the data packet internally in memory.
R	TransformRead (see page 7696)	Specifies the TXMLTransform (see page 7671) component that converts the source XML document into data packets.
R	TransformWrite (see page 7697)	Specifies the TXMLTransform (see page 7671) component that generates the new value of the source XML document when updates are applied.
	XMLDataFile (see page 7697)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

85.1.4.2 TXMLTransformProvider.Create Constructor

Creates and initializes a TXMLTransformProvider (see page 7691) object.

Pascal

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

C++

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

Description

Call Create to instantiate an XML transform provider at runtime. Providers placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the provider.

See Also

Owner

85.1.4.3 TXMLTransformProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformProvider();
```

Description

Do not call Destroy directly. Instead, use the delete keyword, which indirectly calls Destroy.

See Also

delete

85.1.4.4 TXMLTransformProvider Methods

The methods of the TXMLTransformProvider class are listed here.

Protected Methods

	Name	Description
	InternalApplyUpdates (see page 7694)	Provides the underlying implementation of the ApplyUpdates method.
	InternalGetRecords (see page 7695)	Provides the underlying implementation of the GetRecords method.

85.1.4.4.1 TXMLTransformProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates event and before generating the AfterApplyUpdates event. The InternalApplyUpdates method of TCustomProvider is abstract (Delphi) or pure virtual (C++), meaning that it has no implementation. Descendant classes override this method to apply updates, resolve update errors, and return any updates that could not be applied.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

applyupdates

85.1.4.4.2 TXMLTransformProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords to apply the CommandText and Params and generate a BeforeGetRecords event. As implemented in TCustomProvider, InternalGetRecords always returns NULL. Descendant classes override this method to fetch the requested records and assemble them into a data packet.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable object, Params specifies field values that filter the result set. On exit from GetRecords, Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

GetRecords

DoBeforeGetRecords

85.1.4.5 TXMLTransformProvider Properties

The properties of the TXMLTransformProvider class are listed here.

Published Properties

	Name	Description
	CacheData (see page 7696)	Specifies whether the provider caches the data packet internally in memory.
R	TransformRead (see page 7696)	Specifies the TXMLTransform (see page 7671) component that converts the source XML document into data packets.
R	TransformWrite (see page 7697)	Specifies the TXMLTransform (see page 7671) component that generates the new value of the source XML document when updates are applied.
	XMLDataFile (see page 7697)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

85.1.4.5.1 TXMLTransformProvider.CacheData Property

Specifies whether the provider caches the data packet internally in memory.

Pascal

```
property CacheData: Boolean;
```

C++

```
__property Boolean CacheData;
```

Description

Set CacheData to true to specify that the provider should cache its data packet in memory after the first time it is read from the source XML document. When CacheData is false, every time the Data property is read, the GetRecords method is called, or updates are applied, the provider must reread the source document specified by XMLDataFile (see page 7697).

Caching the data improves performance on subsequent data fetches or when applying updates, but consumes additional resources.

See Also

Data

TransformRead (see page 7696)

GetRecords

TransformRead (see page 7696)

XMLDataFile (see page 7697)

85.1.4.5.2 TXMLTransformProvider.TransformRead Property

Specifies the TXMLTransform (see page 7671) component that converts the source XML document into data packets.

Pascal

```
property TransformRead: TXMLTransform;
```

C++

```
__property TXMLTransform TransformRead;
```

Description

Use TransformRead to access the subcomponent that transforms the source XML document into the data packet that is the value of the Data property. Before you can use TXMLTransformProvider (see page 7691), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformRead with an OnTranslate event handler.

See Also

XMLDataFile (see page 7697)

TransformWrite (see page 7697)

Data

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

OnTranslate (see page 7683)

Determining the Source of Data

Using an XML Document as the Source for a Provider

85.1.4.5.3 TXMLTransformProvider.TransformWrite Property

Specifies the TXMLTransform (see page 7671) component that generates the new value of the source XML document when updates are applied.

Pascal

```
property TransformWrite: TXMLTransform;
```

C++

```
__property TXMLTransform TransformWrite;
```

Description

Use TransformWrite to access the subcomponent that transforms a data packet back into the format required by the source XML document. When a client calls the ApplyUpdates method, the delta packet is merged into the transformed data from the XML document, and then TransformWrite converts the resulting dataset into the format of the source XML document.

Before you can use TXMLTransformProvider (see page 7691) to apply updates, you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformWrite with an OnTranslate event handler.

See Also

applyupdates

TransformRead (see page 7696)

TransformationFile (see page 7680)

TransformationDocument (see page 7679)

OnTranslate (see page 7683)

Using an XML Document as the Source for a Provider

85.1.4.5.4 TXMLTransformProvider.XMLDataFile Property

Specifies the XML document from which the provider creates data packets and to which it applies updates.

Pascal

```
property XMLDataFile: string;
```

C++

```
__property UnicodeString XMLDataFile;
```

Description

Use XMLDataFile to specify the XML document whose data the provider represents. When the Data property is read or the GetRecords method is called, TXMLTransformProvider (see page 7691) reads the contents of XMLDataFile and converts it into a data packet using the transform component specified by TransformRead (see page 7696). When the ApplyUpdates method is called, TXMLTransformProvider (see page 7691) uses the transform component specified by TransformWrite (see page 7697) to convert the data into the proper format for XMLDataFile and updates this XML document accordingly.

XMLDataFile is the same as the SourceXmlFile property of TransformRead (see page 7696).

See Also

Data

GetRecords

TransformRead (see page 7696)

TransformWrite (see page 7697)

applyupdates

SourceXmlFile (see page 7679)

Determining the Source of Data

Using an XML Document as the Source for a Provider

85.1.4.6 TXMLTransformProvider Events

The events of the TXMLTransformProvider class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 7698)	Occurs after the provider applies updates.
	AfterGetRecords (see page 7699)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 7700)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 7701)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 7701)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 7703)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 7703)	Occurs when a provider calls its DataRequest method.

85.1.4.6.1 TXMLTransformProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates;
```

C++

```
__property AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, on entry to the event handler the OwnerData parameter contains the value set in the BeforeApplyUpdates event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates

AS_ApplyUpdates

ApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

OnUpdateData

Responding to Client Update Requests

Communicating with the Client Dataset

85.1.4.6.2 TXMLTransformProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords;
```

C++

```
__property AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords

OnGetData

GetRecords

AfterGetRecords

BeforeGetRecords

Communicating with the Client Dataset

Responding to Client Data Requests

85.1.4.6.3 TXMLTransformProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest;
```

C++

```
__property AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

85.1.4.6.4 TXMLTransformProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates;
```

C++

```
__property BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, the OwnerData parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates event handler but not returned to the caller.

See Also

applyupdates

AS_ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

Responding to Client Update Requests

Communicating with the Client Dataset

85.1.4.6.5 TXMLTransformProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords;
```

C++

```
__property BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords

GetRecords

BeforeGetRecords

AfterGetRecords

Communicating with the Client Dataset

Responding to Client Data Requests

85.1.4.6.6 TXMLTransformProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest;
```

C++

```
__property BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

85.1.4.6.7 TXMLTransformProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest;
```

C++

```
__property OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest

IAppServer

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

85.2 Types

The following table lists types in this documentation.

Types

Name	Description
TRowEvent (see page 7705)	This is type Xmlxform.TRowEvent.
TTranslateEvent (see page 7705)	This is type Xmlxform.TTranslateEvent.

85.2.1 Xmlxform.TRowEvent Type

Pascal

```
TRowEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) of object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) ( TRowEvent)();
```

File

Xmlxform

Description

This is type Xmlxform.TRowEvent.

85.2.2 Xmlxform.TTranslateEvent Type

Pascal

```
TTranslateEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; var Value: string; DestNode: IDOMNode) of object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; var Value: string; DestNode: IDOMNode) ( TTranslateEvent)();
```

File

Xmlxform

Description

This is type Xmlxform.TTranslateEvent.

86 XPActnCtrls Namespace

This is namespace XPActnCtrls.

Topics

Name	Description
Classes (see page 7708)	The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleAddRemoveItem (see page 7708)	This is class XPActnCtrls.TXPStyleAddRemoveItem.
	TXPStyleButton (see page 7711)	This is class XPActnCtrls.TXPStyleButton.
	TXPStyleCustomizePopup (see page 7713)	This is class XPActnCtrls.TXPStyleCustomizePopup.
	TXPStyleDropDownBtn (see page 7714)	This is class XPActnCtrls.TXPStyleDropDownBtn.
	TXPStyleExpandBtn (see page 7716)	This is class XPActnCtrls.TXPStyleExpandBtn.
	TXPStyleMenuButton (see page 7718)	This is class XPActnCtrls.TXPStyleMenuButton.
	TXPStyleMenuItem (see page 7719)	This is class XPActnCtrls.TXPStyleMenuItem.
	TXPStylePopupMenu (see page 7722)	This is class XPActnCtrls.TXPStylePopupMenu.
	TXPStyleToolScrollBtn (see page 7726)	This is class XPActnCtrls.TXPStyleToolScrollBtn.

86.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleAddRemoveItem (see page 7708)	This is class XPActnCtrls.TXPStyleAddRemoveItem.
	TXPStyleButton (see page 7711)	This is class XPActnCtrls.TXPStyleButton.
	TXPStyleCustomizePopup (see page 7713)	This is class XPActnCtrls.TXPStyleCustomizePopup.
	TXPStyleDropDownBtn (see page 7714)	This is class XPActnCtrls.TXPStyleDropDownBtn.
	TXPStyleExpandBtn (see page 7716)	This is class XPActnCtrls.TXPStyleExpandBtn.
	TXPStyleMenuButton (see page 7718)	This is class XPActnCtrls.TXPStyleMenuButton.
	TXPStyleMenuItem (see page 7719)	This is class XPActnCtrls.TXPStyleMenuItem.
	TXPStylePopupMenu (see page 7722)	This is class XPActnCtrls.TXPStylePopupMenu.
	TXPStyleToolScrollBtn (see page 7726)	This is class XPActnCtrls.TXPStyleToolScrollBtn.

86.1.1 TXPStyleAddRemoveItem Class

Class Hierarchy

Pascal

```
TXPStyleAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TXPStyleAddRemoveItem : public TCustomAddRemoveItem;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleAddRemoveItem.

86.1.1.1 TXPStyleAddRemoveItem Members

The following tables list the members exposed by TXPStyleAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 7709)	Draws a background image on the control.
	DrawGlyph (see page 7710)	Draws a symbol on the control.
	DrawSeparator (see page 7710)	This is DrawSeparator, a member of class TXPStyleAddRemoveItem.
	DrawText (see page 7710)	This is DrawText, a member of class TXPStyleAddRemoveItem.
	DrawUnusedEdges (see page 7710)	This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 7709)	Calculates the size of the add/remove menu item.

86.1.1.2 TXPStyleAddRemoveItem Methods

The methods of the TXPStyleAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7709)	Draws a background image on the control.
	DrawGlyph (see page 7710)	Draws a symbol on the control.
	DrawSeparator (see page 7710)	This is DrawSeparator, a member of class TXPStyleAddRemoveItem.
	DrawText (see page 7710)	This is DrawText, a member of class TXPStyleAddRemoveItem.
	DrawUnusedEdges (see page 7710)	This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 7709)	Calculates the size of the add/remove menu item.

86.1.1.2.1 TXPStyleAddRemoveItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height

Width

TextBounds

86.1.1.2.2 TXPStyleAddRemoveItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.1.2.3 TXPStyleAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

86.1.1.2.4 TXPStyleAddRemoveItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TXPStyleAddRemoveItem.

86.1.1.2.5 TXPStyleAddRemoveItem.DrawText Method

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleAddRemoveItem.

86.1.1.2.6 TXPStyleAddRemoveItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

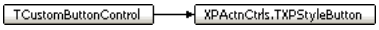
```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

86.1.2 TXPStyleButton Class

Class Hierarchy



Pascal

```
TXPStyleButton = class(TCustomButtonControl);
```

C++

```
class TXPStyleButton : public TCustomButtonControl;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleButton.

86.1.2.1 TXPStyleButton Members

The following tables list the members exposed by TXPStyleButton.

Protected Methods

	Name	Description
	DrawBackground (see page 7711)	Draws a background image on the control.
	DrawFrame (see page 7712)	This is DrawFrame, a member of class TXPStyleButton.
	DrawGlyph (see page 7712)	Draws a symbol on the control.
	DrawText (see page 7712)	This is DrawText, a member of class TXPStyleButton.

86.1.2.2 TXPStyleButton Methods

The methods of the TXPStyleButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7711)	Draws a background image on the control.
	DrawFrame (see page 7712)	This is DrawFrame, a member of class TXPStyleButton.
	DrawGlyph (see page 7712)	Draws a symbol on the control.
	DrawText (see page 7712)	This is DrawText, a member of class TXPStyleButton.

86.1.2.2.1 TXPStyleButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.2.2.2 TXPStyleButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleButton.

86.1.2.2.3 TXPStyleButton.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

86.1.2.2.4 TXPStyleButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleButton.

86.1.3 TXPStyleCustomizePopup Class

Class Hierarchy

Pascal

```
TXPStyleCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TXPStyleCustomizePopup : public TCustomizeActionToolBar;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleCustomizePopup.

86.1.3.1 TXPStyleCustomizePopup Members

The following tables list the members exposed by TXPStyleCustomizePopup.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 7714)	This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.
	GetDefaultColorMapClass (see page 7714)	This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.
	NCPaint (see page 7714)	This is NCPaint, a member of class TXPStyleCustomizePopup.

Public Methods

	Name	Description
	Create (see page 7713)	Creates and initializes a TCustomActionPopupMenu instance.

86.1.3.2 TXPStyleCustomizePopup.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

86.1.3.3 TXPStyleCustomizePopup Methods

The methods of the TXPStyleCustomizePopup class are listed here.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 7714)	This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.
	GetDefaultColorMapClass (see page 7714)	This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.
	NCPaint (see page 7714)	This is NCPaint, a member of class TXPStyleCustomizePopup.

86.1.3.3.1 TXPStyleCustomizePopup.GetAddRemoveItemClass Method

Pascal

```
function GetAddRemoveItemClass: TCustomAddRemoveItemClass; override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass();
```

Description

This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.

86.1.3.3.2 TXPStyleCustomizePopup.GetDefaultColorMapClass Method

Pascal

```
function GetDefaultColorMapClass: TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetDefaultColorMapClass();
```

Description

This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.

86.1.3.3.3 TXPStyleCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TXPStyleCustomizePopup.

86.1.4 TXPStyleDropDownBtn Class

Class Hierarchy

Pascal

```
TXPStyleDropDownBtn = class(TCustomDropDownButton);
```

C++

```
class TXPStyleDropDownBtn : public TCustomDropDownButton;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleDropDownBtn.

86.1.4.1 TXPStyleDropDownBtn Members

The following tables list the members exposed by TXPStyleDropDownBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7715)	Draws a background image on the control.
	DrawFrame (see page 7716)	This is DrawFrame, a member of class TXPStyleDropDownBtn.
	DrawGlyph (see page 7716)	Draws a symbol on the control.
	GetPopupClass (see page 7716)	This is GetPopupClass, a member of class TXPStyleDropDownBtn.

86.1.4.2 TXPStyleDropDownBtn Methods

The methods of the TXPStyleDropDownBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7715)	Draws a background image on the control.
	DrawFrame (see page 7716)	This is DrawFrame, a member of class TXPStyleDropDownBtn.
	DrawGlyph (see page 7716)	Draws a symbol on the control.
	GetPopupClass (see page 7716)	This is GetPopupClass, a member of class TXPStyleDropDownBtn.

86.1.4.2.1 TXPStyleDropDownBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.4.2.2 TXPStyleDropDownBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleDropDownBtn.

86.1.4.2.3 TXPStyleDropDownBtn.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

86.1.4.2.4 TXPStyleDropDownBtn.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TXPStyleDropDownBtn.

86.1.5 TXPStyleExpandBtn Class

Class Hierarchy

Pascal

```
TXPStyleExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TXPStyleExpandBtn : public TCustomMenuExpandBtn;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleExpandBtn.

86.1.5.1 TXPStyleExpandBtn Members

The following tables list the members exposed by TXPStyleExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7717)	Draws a background image on the control.
	DrawFrame (see page 7717)	This is DrawFrame, a member of class TXPStyleExpandBtn.

86.1.5.2 TXPStyleExpandBtn Methods

The methods of the TXPStyleExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7717)	Draws a background image on the control.
	DrawFrame (see page 7717)	This is DrawFrame, a member of class TXPStyleExpandBtn.

86.1.5.2.1 TXPStyleExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.5.2.2 TXPStyleExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleExpandBtn.

86.1.6 TXPStyleMenuButton Class

Class Hierarchy

Pascal

```
TXPStyleMenuButton = class(TCustomMenuButton);
```

C++

```
class TXPStyleMenuButton : public TCustomMenuButton;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleMenuButton.

86.1.6.1 TXPStyleMenuButton Members

The following tables list the members exposed by TXPStyleMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 7718)	Draws a background image on the control.
	DrawText (see page 7719)	This is DrawText, a member of class TXPStyleMenuButton.

86.1.6.2 TXPStyleMenuButton Methods

The methods of the TXPStyleMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7718)	Draws a background image on the control.
	DrawText (see page 7719)	This is DrawText, a member of class TXPStyleMenuButton.

86.1.6.2.1 TXPStyleMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.6.2.2 TXPStyleMenuButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleMenuButton.

86.1.7 TXPStyleMenuItem Class

Class Hierarchy

Pascal

```
TXPStyleMenuItem = class(TCustomMenuItem);
```

C++

```
class TXPStyleMenuItem : public TCustomMenuItem;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleMenuItem.

86.1.7.1 TXPStyleMenuItem Members

The following tables list the members exposed by TXPStyleMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7720)	This is CalcLayout, a member of class TXPStyleMenuItem.
	DrawBackground (see page 7721)	Draws a background image on the control.
	DrawGlyph (see page 7721)	Draws a symbol on the control.
	DrawSeparator (see page 7721)	This is DrawSeparator, a member of class TXPStyleMenuItem.
	DrawShadowedText (see page 7721)	This is DrawShadowedText, a member of class TXPStyleMenuItem.
	DrawSubMenuGlyph (see page 7722)	This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.
	DrawText (see page 7722)	This is DrawText, a member of class TXPStyleMenuItem.

	DrawUnusedEdges (see page 7722)	This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7720)	Calculates the size of the menu item.

86.1.7.2 TXPStyleMenuItem Methods

The methods of the TXPStyleMenuItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7720)	This is CalcLayout, a member of class TXPStyleMenuItem.
	DrawBackground (see page 7721)	Draws a background image on the control.
	DrawGlyph (see page 7721)	Draws a symbol on the control.
	DrawSeparator (see page 7721)	This is DrawSeparator, a member of class TXPStyleMenuItem.
	DrawShadowedText (see page 7721)	This is DrawShadowedText, a member of class TXPStyleMenuItem.
	DrawSubMenuGlyph (see page 7722)	This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.
	DrawText (see page 7722)	This is DrawText, a member of class TXPStyleMenuItem.
	DrawUnusedEdges (see page 7722)	This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7720)	Calculates the size of the menu item.

86.1.7.2.1 TXPStyleMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

86.1.7.2.2 TXPStyleMenuItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TXPStyleMenuItem.

86.1.7.2.3 TXPStyleMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

86.1.7.2.4 TXPStyleMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

86.1.7.2.5 TXPStyleMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TXPStyleMenuItem.

86.1.7.2.6 TXPStyleMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: String; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TXPStyleMenuItem.

86.1.7.2.7 TXPStyleMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; override;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.

86.1.7.2.8 TXPStyleMenuItem.DrawText Method

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleMenuItem.

86.1.7.2.9 TXPStyleMenuItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

86.1.8 TXPStylePopupMenu Class

Class Hierarchy

Pascal

```
TXPStylePopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TXPStylePopupMenu : public TCustomActionPopupMenu;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStylePopupMenu.

86.1.8.1 TXPStylePopupMenu Members

The following tables list the members exposed by TXPStylePopupMenu.

Protected Methods

	Name	Description
	DisplayShadow (see page 7724)	This is DisplayShadow, a member of class TXPStylePopupMenu.
	GetExpandBtnClass (see page 7724)	This is GetExpandBtnClass, a member of class TXPStylePopupMenu.
	HideShadow (see page 7725)	This is HideShadow, a member of class TXPStylePopupMenu.
	NCPaint (see page 7725)	This is NCPaint, a member of class TXPStylePopupMenu.
	PositionPopup (see page 7725)	This is PositionPopup, a member of class TXPStylePopupMenu.
	VisibleChanging (see page 7725)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
	Create (see page 7723)	Creates and initializes a TCustomActionPopupMenu instance.
	Destroy (see page 7723)	Destroys an instance of TCustomActionMenuBar.

86.1.8.2 TXPStylePopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

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

C++

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

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

86.1.8.3 TXPStylePopupMenu.Destroy Destructor

Destroys an instance of TCustomActionMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXPStylePopupMenu();
```

Description

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

See Also

System::TObject::Free (see page 3382)

Destroy

86.1.8.4 TXPStylePopupMenu Methods

The methods of the TXPStylePopupMenu class are listed here.

Protected Methods

	Name	Description
	DisplayShadow (see page 7724)	This is DisplayShadow, a member of class TXPStylePopupMenu.
	GetExpandBtnClass (see page 7724)	This is GetExpandBtnClass, a member of class TXPStylePopupMenu.
	HideShadow (see page 7725)	This is HideShadow, a member of class TXPStylePopupMenu.
	NCPaint (see page 7725)	This is NCPaint, a member of class TXPStylePopupMenu.
	PositionPopup (see page 7725)	This is PositionPopup, a member of class TXPStylePopupMenu.
	VisibleChanging (see page 7725)	Provides the interface for a method that responds to changes in the control's visibility.

86.1.8.4.1 TXPStylePopupMenu.DisplayShadow Method

Pascal

```
procedure DisplayShadow; override;
```

C++

```
virtual __fastcall DisplayShadow();
```

Description

This is DisplayShadow, a member of class TXPStylePopupMenu.

86.1.8.4.2 TXPStylePopupMenu.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TXPStylePopupMenu.

86.1.8.4.3 TXPStylePopupMenu.HideShadow Method

Pascal

```
procedure HideShadow; override;
```

C++

```
virtual __fastcall HideShadow();
```

Description

This is HideShadow, a member of class TXPStylePopupMenu.

86.1.8.4.4 TXPStylePopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TXPStylePopupMenu.

86.1.8.4.5 TXPStylePopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TXPStylePopupMenu.

86.1.8.4.6 TXPStylePopupMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible

86.1.9 TXPStyleToolScrollBtn Class

Class Hierarchy

Pascal

```
TXPStyleToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TXPStyleToolScrollBtn : public TCustomToolScrollBtn;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleToolScrollBtn.

86.1.9.1 TXPStyleToolScrollBtn Members

The following tables list the members exposed by TXPStyleToolScrollBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7726)	Draws a background image on the control.

86.1.9.2 TXPStyleToolScrollBtn Methods

The methods of the TXPStyleToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7726)	Draws a background image on the control.

86.1.9.2.1 TXPStyleToolScrollBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

87 XPMan Namespace

This is namespace XPMan.

Topics

Name	Description
Classes (see page 7730)	The following table lists classes in this documentation.

Classes

	Name	Description
	TXPManifest (see page 7730)	This is class XPMan.TXPManifest.

87.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPManifest (see page 7730)	This is class XPMan.TXPManifest.

87.1.1 TXPManifest Class

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TXPManifest = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TXPManifest : public TComponent;
```

File

XPMan

Description

This is class XPMan.TXPManifest.

88 XPStyleActnCtrls Namespace

This is namespace XPStyleActnCtrls.

Topics

Name	Description
Classes (see page 7732)	The following table lists classes in this documentation.
Variables (see page 7735)	The following table lists variables in this documentation.

Classes

	Name	Description
	TXPStyleActionBars (see page 7732)	This is class XPStyleActnCtrls.TXPStyleActionBars.

Variables

Name	Description
XPStyle (see page 7735)	This is variable XPStyleActnCtrls.XPStyle.

88.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleActionBars (see page 7732)	This is class XPStyleActnCtrls.TXPStyleActionBars.

88.1.1 TXPStyleActionBars Class

Class Hierarchy

Pascal

```
TXPStyleActionBars = class(TActionBarStyleEx);
```

C++

```
class TXPStyleActionBars : public TActionBarStyleEx;
```

File

XPStyleActnCtrls

Description

This is class XPStyleActnCtrls.TXPStyleActionBars.

88.1.1.1 TXPStyleActionBars Members

The following tables list the members exposed by TXPStyleActionBars.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 7733)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 7733)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7733)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7734)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7734)	This is GetScrollBtnClass, a member of class TXPStyleActionBars.
	GetStyleName (see page 7734)	Returns the style associated with action manager.

88.1.1.2 TXPStyleActionBars Methods

The methods of the TXPStyleActionBars class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 7733)	Returns the base class for ActionBand menus for customizing action bands.

	GetColorMapClass (see page 7733)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7733)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7734)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7734)	This is GetScrollBtnClass, a member of class TXPStyleActionBars.
	GetStyleName (see page 7734)	Returns the style associated with action manager.

88.1.1.2.1 TXPStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

88.1.1.2.2 TXPStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

88.1.1.2.3 TXPStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

88.1.1.2.4 TXPStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

88.1.1.2.5 TXPStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TXPStyleActionBars.

88.1.1.2.6 TXPStyleActionBars.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetStyleName();
```

Description

Returns the value of the Style property.

88.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
XPStyle (see page 7735)	This is variable XPStyleActnCtrls.XPStyle.

88.2.1 XPStyleActnCtrls.XPStyle Variable

Pascal

```
XPStyle: TXPStyleActionBars;
```

C++

```
TXPStyleActionBars XPStyle;
```

File

XPStyleActnCtrls

Description

This is variable XPStyleActnCtrls.XPStyle.

89 XSBuiltIns Namespace

This is namespace XSBuiltIns.

Topics

Name	Description
Classes (see page 7739)	The following table lists classes in this documentation.
Functions (see page 7805)	The following table lists functions in this documentation.
Types (see page 7809)	The following table lists types in this documentation.
Constants (see page 7810)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 7739)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 7740)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 7740)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXMLData (see page 7741)	This is class XSBuiltIns.TXMLData.
	TXSBoolean (see page 7745)	This is class XSBuiltIns.TXSBoolean.
	TXSCustomDateTime (see page 7747)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 7766)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 7772)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 7776)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 7780)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 7784)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSInteger (see page 7787)	This is class XSBuiltIns.TXSInteger.
	TXSLong (see page 7789)	This is class XSBuiltIns.TXSLong.
	TXSString (see page 7791)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 7792)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Constants

Name	Description
IncrementAmount (see page 7810)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 7811)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 7811)	This is constant XSBuiltIns.MaxMonth.
MinDay (see page 7811)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 7812)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 7812)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 7812)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 7813)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 7813)	This is constant XSBuiltIns.SNAN.
SNegative (see page 7813)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 7814)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 7814)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 7814)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 7815)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 7815)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 7815)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 7816)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 7816)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 7816)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 7817)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 7817)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 7817)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 7818)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 7818)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 7818)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 7819)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 7819)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 7819)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 7820)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 7820)	This is constant XSBuiltIns.XMLYearPos.

Functions

	Name	Description
	DateTimeToXMLTime (see page 7805)	Returns the XML string that represents a TDateTime (see page 3335) value.
	DateTimeToXSDateTime (see page 7805)	Returns a TXSDateTime (see page 7772) object that encodes a TDateTime (see page 3335) value.
	GetDataFromFile (see page 7806)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 7806)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 7807)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 7807)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 7807)	Returns a TDateTime (see page 3335) value to represent an XML datetime value.

Types

Name	Description
TXSTimeInstant (see page 7809)	This is type XSBuiltIns.TXSTimeInstant.

89.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 7739)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 7740)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 7740)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXMLData (see page 7741)	This is class XSBuiltIns.TXMLData.
	TXSBoolean (see page 7745)	This is class XSBuiltIns.TXSBoolean.
	TXSCustomDateTime (see page 7747)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 7766)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 7772)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 7776)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 7780)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 7784)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSInteger (see page 7787)	This is class XSBuiltIns.TXSInteger.
	TXSLong (see page 7789)	This is class XSBuiltIns.TXSLong.
	TXSString (see page 7791)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 7792)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

89.1.1 EXSDateTimeException Class

EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.

Class Hierarchy

Pascal

```
EXSDateTimeException = class(Exception);
```

C++

```
class EXSDateTimeException : public Exception;
```

File

XSBuiltIns

Description

EXSDateTimeException is raised by the TXSDate (see page 7766), TXSTime (see page 7792), TXSDateTime (see page

7772), and TXSDuration (see page 7780) objects when

An application assigns a value that is out of range to one of their properties. Such values include Month values greater than 12, Hour values greater than 23, Minute values greater than 59, and so on.

The string that represents the object's value does not have the proper format and can't be parsed.

See Also

TXSTime (see page 7792)

TXSDate (see page 7766)

TXSDateTime (see page 7772)

TXSDuration (see page 7780)

89.1.2 EXSDecimalException Class

EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.

Class Hierarchy

Pascal

```
EXSDecimalException = class(Exception);
```

C++

```
class EXSDecimalException : public Exception;
```

File

XSBuiltIns

Description

EXSDecimalException is raised by TXSDecimal (see page 7776) when it is unable to convert between its string value and a TBcd value.

See Also

TXSDecimal (see page 7776)

TBcd

89.1.3 EXSHexBinaryException Class

EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.

Class Hierarchy

Pascal

```
EXSHexBinaryException = class(Exception);
```

C++

```
class EXSHexBinaryException : public Exception;
```

File

XSBuiltIns

Description

EXSHexBinaryException is raised by TXSHexBinary (see page 7784) when it encounters a string that contains invalid characters for a hexadecimal value.

See Also

TXSDecimal (see page 7776)

TBcd

89.1.4 TXMLData Class

Class Hierarchy

Pascal

```
TXMLData = class(TRemotable);
```

C++

```
class TXMLData : public TRemotable;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXMLData.

89.1.4.1 TXMLData Members

The following tables list the members exposed by TXMLData.

Public Methods

	Name	Description
	Create (see page 7742)	Creates and initializes a TRemotable object.
	Destroy (see page 7742)	Frees the memory associated with the TRemotable object.
	LoadFromXML (see page 7743)	This is the overview for the LoadFromXML method overload.
	ObjectToSOAP (see page 7743)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 7744)	Initializes this TRemotable instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	XMLNode (see page 7745)	This is XMLNode, a member of class TXMLData.

89.1.4.2 TXMLData.Create Constructor

Creates and initializes a TRemotable object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXMLData();
```

Description

The TRemotable constructor does nothing special. It exists primarily because

The remotable type registry knows how to call it.

It is virtual, so that the remotable type registry can also instantiate descendant classes.

See Also

RemTypeRegistry

89.1.4.3 TXMLData.Destroy Destructor

Frees the memory associated with the TRemotable object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLData();
```

Description

Use the Free method to free a TRemotable instance. Free checks that the object is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TRemotable instance (for example, one created as a return value or output parameter). By default, when a TRemotable descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TRemotable instances that are passed in as a parameter belong to the data context.

On client applications, TRemotable instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

System::TObject::Free (see page 3382)

DataContext

89.1.4.4 TXMLData Methods

The methods of the TXMLData class are listed here.

Public Methods

	Name	Description
	LoadFromXML (see page 7743)	This is the overview for the LoadFromXML method overload.
V	ObjectToSOAP (see page 7743)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 7744)	Initializes this TRemotable instance to represent the value of its SOAP representation.

89.1.4.4.1 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	TXMLData.LoadFromXML (string) (see page 7743)	This is LoadFromXML, a member of class TXMLData.
	TXMLData.LoadFromXML (WideString) (see page 7743)	This is LoadFromXML, a member of class TXMLData.

89.1.4.4.1.1 TXMLData.LoadFromXML Method (WideString)

Pascal

```
procedure LoadFromXML(const XML: WideString); overload;
```

C++

```
__fastcall LoadFromXML(const BSTR XML);
```

Description

This is LoadFromXML, a member of class TXMLData.

89.1.4.4.1.2 TXMLData.LoadFromXML Method (string)

Pascal

```
procedure LoadFromXML(const XML: string); overload;
```

C++

```
__fastcall LoadFromXML(const UnicodeString XML);
```

Description

This is LoadFromXML, a member of class TXMLData.

89.1.4.4.2 TXMLData.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
```

```
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotable descendant to its SOAP representation. In TRemotable, ObjectToSOAP simply calls the ObjInstanceToSOAP method of the converter that is supplied by the ObjConverter parameter.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP

SOAPToObject

89.1.4.4.3 TXMLData.SOAPToObject Method

Initializes this TRemotable instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific remotable type from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotable, SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP

InitObjectFromSOAP

89.1.4.5 TXMLData Properties

The properties of the TXMLData class are listed here.

Public Properties

	Name	Description
	XMLNode (see page 7745)	This is XMLNode, a member of class TXMLData.

89.1.4.5.1 TXMLData.XMLNode Property

Pascal

```
property XMLNode: IXMLNode;
```

C++

```
__property IXMLNode XMLNode;
```

Description

This is XMLNode, a member of class TXMLData.

89.1.5 TXSBoolean Class

Class Hierarchy

Pascal

```
TXSBoolean = class(TRemotableXS);
```

C++

```
class TXSBoolean : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSBoolean.

89.1.5.1 TXSBoolean Members

The following tables list the members exposed by TXSBoolean.

Public Methods

	Name	Description
	NativeToXS (see page 7746)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7746)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsBoolean (see page 7747)	This is AsBoolean, a member of class TXSBoolean.

89.1.5.2 TXSBoolean Methods

The methods of the TXSBoolean class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7746)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7746)	Assigns the value of a string to the TRemotableXS instance.

89.1.5.2.1 TXSBoolean.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

89.1.5.2.2 TXSBoolean.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

89.1.5.3 TXSBoolean Properties

The properties of the TXSBoolean class are listed here.

Public Properties

	Name	Description
	AsBoolean (see page 7747)	This is AsBoolean, a member of class TXSBoolean.

89.1.5.3.1 TXSBoolean.AsBoolean Property

Pascal

```
property AsBoolean: Boolean;
```

C++

```
__property Boolean AsBoolean;
```

Description

This is AsBoolean, a member of class TXSBoolean.

89.1.6 TXSCustomDateTime Class

TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.

Class Hierarchy

Pascal

```
TXSCustomDateTime = class(TRemotableXS);
```

C++

```
class TXSCustomDateTime : public TRemotableXS;
```

File

XSBuiltIns

Description

Use TXSCustomDateTime as a base class when defining a class to represent a type that includes both date and time portions and that represents a scalar value in a WSDL document. When creating a TXSCustomDateTime descendant, override the XSToNative and NativeToXS methods to indicate how the value of the class can be converted to and from the string representation that is used to encode the value for transport.

On server applications, instances of TXSCustomDateTime descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSCustomDateTime descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSCustomDateTime instances that it passes as input parameters, and for freeing any TXSCustomDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDateTime (see page 7772)

TXSDuration (see page 7780)

89.1.6.1 TXSCustomDateTime Members

The following tables list the members exposed by TXSCustomDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 7750)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7751)	Returns the value of the AsDateTime (see page 7759) property.
	GetAsUTCDateTime (see page 7751)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7751)	Returns the value of the Day (see page 7760) property.
	GetFractionalSeconds (see page 7752)	Returns the value of the FractionalSeconds (see page 7761) property.
	GetHour (see page 7752)	Returns the value of the Hour (see page 7761) property.
	GetHourOffset (see page 7752)	Returns the value of the HourOffset (see page 7762) property.
	GetMillisecond (see page 7752)	Returns the value of the Millisecond (see page 7762) property.
	GetMinute (see page 7753)	Returns the value of the Minute (see page 7763) property.
	GetMinuteOffset (see page 7753)	Returns the value of the MinuteOffset (see page 7763) property.
	GetMonth (see page 7753)	Returns the value of the Month (see page 7764) property.
	GetSecond (see page 7754)	Returns the value of the Second (see page 7764) property.
	GetUseZeroMilliseconds (see page 7754)	Returns the value of the UseZeroMilliseconds (see page 7765) property.
	GetYear (see page 7754)	Returns the value of the Year (see page 7765) property.
	SetAsDateTime (see page 7755)	Sets the value of the AsDateTime (see page 7759) property.
	SetAsUTCDateTime (see page 7755)	Sets the value of the AsUTCDateTime (see page 7760) property.
	SetDay (see page 7755)	Sets the value of the Day (see page 7760) property.
	SetFractionalSeconds (see page 7755)	Sets the value of the FractionalSeconds (see page 7761) property.
	SetHour (see page 7756)	Sets the value of the Hour (see page 7761) property.
	SetHourOffset (see page 7756)	Sets the value of the HourOffset (see page 7762) property.
	SetMillisecond (see page 7756)	Sets the value of the Millisecond (see page 7762) property.
	SetMinute (see page 7757)	Sets the value of the Minute (see page 7763) property.
	SetMinuteOffset (see page 7757)	Sets the value of the MinuteOffset (see page 7763) property.
	SetMonth (see page 7757)	Sets the value of the Month (see page 7764) property.
	SetSecond (see page 7758)	Sets the value of the Second (see page 7764) property.
	SetUseZeroMilliseconds (see page 7758)	Sets the value of the UseZeroMilliseconds (see page 7765) property.
	SetYear (see page 7758)	Sets the value of the Year (see page 7765) property.

Protected Properties

	Name	Description
	Millisecond (see page 7762)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
	Create (see page 7749)	Creates and initializes a TXSCustomDateTime (see page 7747) object.
	Destroy (see page 7749)	Frees the memory associated with the TXSCustomDateTime (see page 7747) object.

Public Properties

	Name	Description
	AsDateTime (see page 7759)	Represents the object's value as a TDateTime (see page 3335) value.
	AsUTCDateTime (see page 7760)	Represents the object's value as a TDateTime (see page 3335) value using UTC time.
	Day (see page 7760)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7761)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7764) property.
	Hour (see page 7761)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7762)	Indicates the difference between the Hour (see page 7761) property and the hour in the local time zone.
	Minute (see page 7763)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7763)	Indicates the difference between the Minute (see page 7763) property and the minute of the hour in local time.
	Month (see page 7764)	Indicates the month of the year that the object represents.
	Second (see page 7764)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7765)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7762) is zero.
	Year (see page 7765)	Indicates the year that the object represents.

89.1.6.2 TXSCustomDateTime.Create Constructor

Creates and initializes a TXSCustomDateTime (see page 7747) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSCustomDateTime();
```

Description

Call Create to create a TXSCustomDateTime (see page 7747) object at runtime.

See Also

Destroy (see page 7749)

89.1.6.3 TXSCustomDateTime.Destroy Destructor

Frees the memory associated with the TXSCustomDateTime (see page 7747) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXSCustomDateTime();
```

Description

Do not call Destroy to free a TXSCustomDateTime (see page 7747) object. Instead, use the Free method, which checks that the TXSCustomDatetime reference is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TXSCustomDateTime (see page 7747) instance (for example, one created as a return value or output parameter). By default, when a TXSCustomDateTime (see page 7747) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext property). As long as the remotable object belongs to the data context, the data context handles freeing the

object. Similarly, TCustomXSDateTime objects that are passed in as a parameter belong to the data context.

On client applications, TXSCustomDateTime (see page 7747) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

DataContext

89.1.6.4 TXSCustomDateTime Fields

The fields of the TXSCustomDateTime class are listed here.

Protected Fields

	Name	Description
	FDateTime (see page 7750)	This is FDateTime, a member of class TXSCustomDateTime.

89.1.6.4.1 TXSCustomDateTime.FDateTime Field

Pascal

```
FDateTime: WideString;
```

C++

```
BSTR FDateTime;
```

Description

This is FDateTime, a member of class TXSCustomDateTime.

89.1.6.5 TXSCustomDateTime Methods

The methods of the TXSCustomDateTime class are listed here.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7751)	Returns the value of the AsDateTime (see page 7759) property.
	GetAsUTCDateTime (see page 7751)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7751)	Returns the value of the Day (see page 7760) property.
	GetFractionalSeconds (see page 7752)	Returns the value of the FractionalSeconds (see page 7761) property.
	GetHour (see page 7752)	Returns the value of the Hour (see page 7761) property.
	GetHourOffset (see page 7752)	Returns the value of the HourOffset (see page 7762) property.
	GetMillisecond (see page 7752)	Returns the value of the Millisecond (see page 7762) property.
	GetMinute (see page 7753)	Returns the value of the Minute (see page 7763) property.
	GetMinuteOffset (see page 7753)	Returns the value of the MinuteOffset (see page 7763) property.
	GetMonth (see page 7753)	Returns the value of the Month (see page 7764) property.
	GetSecond (see page 7754)	Returns the value of the Second (see page 7764) property.
	GetUseZeroMilliseconds (see page 7754)	Returns the value of the UseZeroMilliseconds (see page 7765) property.
	GetYear (see page 7754)	Returns the value of the Year (see page 7765) property.
	SetAsDateTime (see page 7755)	Sets the value of the AsDateTime (see page 7759) property.
	SetAsUTCDateTime (see page 7755)	Sets the value of the AsUTCDateTime (see page 7760) property.
	SetDay (see page 7755)	Sets the value of the Day (see page 7760) property.
	SetFractionalSeconds (see page 7755)	Sets the value of the FractionalSeconds (see page 7761) property.
	SetHour (see page 7756)	Sets the value of the Hour (see page 7761) property.
	SetHourOffset (see page 7756)	Sets the value of the HourOffset (see page 7762) property.

	SetMillisecond (see page 7756)	Sets the value of the Millisecond (see page 7762) property.
	SetMinute (see page 7757)	Sets the value of the Minute (see page 7763) property.
	SetMinuteOffset (see page 7757)	Sets the value of the MinuteOffset (see page 7763) property.
	SetMonth (see page 7757)	Sets the value of the Month (see page 7764) property.
	SetSecond (see page 7758)	Sets the value of the Second (see page 7764) property.
	SetUseZeroMilliseconds (see page 7758)	Sets the value of the UseZeroMilliseconds (see page 7765) property.
	SetYear (see page 7758)	Sets the value of the Year (see page 7765) property.

89.1.6.5.1 TXSCustomDateTime.GetAsDateTime Method

Returns the value of the AsDateTime (see page 7759) property.

Pascal

```
function GetAsDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDateTime();
```

Description

GetAsDateTime is the protected read implementation of the AsDateTime (see page 7759) property.

See Also

AsDateTime (see page 7759)

89.1.6.5.2 TXSCustomDateTime.GetAsUTCDateTime Method

Pascal

```
function GetAsUTCDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsUTCDateTime();
```

Description

This is GetAsUTCDateTime, a member of class TXSCustomDateTime.

89.1.6.5.3 TXSCustomDateTime.GetDay Method

Returns the value of the Day (see page 7760) property.

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

GetDay is the protected read implementation of the Day (see page 7760) property.

See Also

Day (see page 7760)

89.1.6.5.4 TXSCustomDateTime.GetFractionalSeconds Method

Returns the value of the FractionalSeconds (see page 7761) property.

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

GetFractionalSeconds is the protected read implementation of the FractionalSeconds (see page 7761) property.

See Also

FractionalSeconds (see page 7761)

89.1.6.5.5 TXSCustomDateTime.GetHour Method

Returns the value of the Hour (see page 7761) property.

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

GetHour is the protected read implementation of the Hour (see page 7761) property.

See Also

Hour (see page 7761)

89.1.6.5.6 TXSCustomDateTime.GetHourOffset Method

Returns the value of the HourOffset (see page 7762) property.

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

GetHourOffset is the protected read implementation of the HourOffset (see page 7762) property.

See Also

HourOffset (see page 7762)

89.1.6.5.7 TXSCustomDateTime.GetMillisecond Method

Returns the value of the Millisecond (see page 7762) property.

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

GetMillisecond is the protected read implementation of the Millisecond (see page 7762) property.

See Also

Millisecond (see page 7762)

89.1.6.5.8 TXSCustomDateTime.GetMinute Method

Returns the value of the Minute (see page 7763) property.

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

GetMinute is the protected read implementation of the Minute (see page 7763) property.

See Also

Minute (see page 7763)

89.1.6.5.9 TXSCustomDateTime.GetMinuteOffset Method

Returns the value of the MinuteOffset (see page 7763) property.

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

GetMinuteOffset is the protected read implementation of the MinuteOffset (see page 7763) property.

See Also

MinuteOffset (see page 7763)

89.1.6.5.10 TXSCustomDateTime.GetMonth Method

Returns the value of the Month (see page 7764) property.

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

GetMonth is the protected read implementation of the Month (see page 7764) property.

See Also

Month (see page 7764)

89.1.6.5.11 TXSCustomDateTime.GetSecond Method

Returns the value of the Second (see page 7764) property.

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

GetSecond is the protected read implementation of the Second (see page 7764) property.

See Also

Second (see page 7764)

89.1.6.5.12 TXSCustomDateTime.GetUseZeroMilliseconds Method

Returns the value of the UseZeroMilliseconds (see page 7765) property.

Pascal

```
function GetUseZeroMilliseconds: Boolean;
```

C++

```
__fastcall Boolean GetUseZeroMilliseconds();
```

Description

GetUseZeroMilliseconds is the protected read implementation of the UseZeroMilliseconds (see page 7765) property.

See Also

UseZeroMilliseconds (see page 7765)

89.1.6.5.13 TXSCustomDateTime.GetYear Method

Returns the value of the Year (see page 7765) property.

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

GetYear is the protected read implementation of the Year (see page 7765) property.

See Also

Year (see page 7765)

89.1.6.5.14 TXSCustomDateTime.SetAsDateTime Method

Sets the value of the AsDateTime (see page 7759) property.

Pascal

```
procedure SetAsDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsDateTime(const TDateTime Value);
```

Description

SetAsDateTime is the protected write implementation of the AsDateTime (see page 7759) property.

See Also

AsDateTime (see page 7759)

89.1.6.5.15 TXSCustomDateTime.SetAsUTCDateTime Method

Sets the value of the AsUTCDateTime (see page 7760) property.

Pascal

```
procedure SetAsUTCDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsUTCDateTime(const TDateTime Value);
```

Description

SetAsUTCDateTime is the protected write implementation of the AsUTCDateTime (see page 7760) property.

See Also

AsUTCDateTime (see page 7760)

89.1.6.5.16 TXSCustomDateTime.SetDay Method

Sets the value of the Day (see page 7760) property.

Pascal

```
procedure SetDay(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetDay(const Word Value);
```

Description

SetDay is the protected write implementation of the Day (see page 7760) property.

See Also

Day (see page 7760)

89.1.6.5.17 TXSCustomDateTime.SetFractionalSeconds Method

Sets the value of the FractionalSeconds (see page 7761) property.

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

SetDay (see page 7755) is the protected write implementation of the FractionalSeconds (see page 7761) property.

Value is the fraction of a second. It is a value between 0 and 1.

See Also

FractionalSeconds (see page 7761)

89.1.6.5.18 TXSCustomDateTime.SetHour Method

Sets the value of the Hour (see page 7761) property.

Pascal

```
procedure SetHour(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetHour(const Word Value);
```

Description

SetHour is the protected write implementation of the Hour (see page 7761) property.

See Also

Hour (see page 7761)

89.1.6.5.19 TXSCustomDateTime.SetHourOffset Method

Sets the value of the HourOffset (see page 7762) property.

Pascal

```
procedure SetHourOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetHourOffset(const SmallInt Value);
```

Description

SetHourOffset is the protected write implementation of the HourOffset (see page 7762) property.

See Also

HourOffset (see page 7762)

89.1.6.5.20 TXSCustomDateTime.SetMillisecond Method

Sets the value of the Millisecond (see page 7762) property.

Pascal

```
procedure SetMillisecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMillisecond(const Word Value);
```

Description

SetMillisecond is the protected write implementation for the Millisecond (see page 7762) property.

See Also

Millisecond (see page 7762)

89.1.6.5.21 TXSCustomDateTime.SetMinute Method

Sets the value of the Minute (see page 7763) property.

Pascal

```
procedure SetMinute(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMinute(const Word Value);
```

Description

SetMinute is the protected write implementation of the Minute (see page 7763) property.

See Also

Minute (see page 7763)

89.1.6.5.22 TXSCustomDateTime.SetMinuteOffset Method

Sets the value of the MinuteOffset (see page 7763) property.

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetMinuteOffset(const SmallInt Value);
```

Description

SetMinuteOffest is the protected write implementation of the MinuteOffset (see page 7763) property.

See Also

MinuteOffset (see page 7763)

89.1.6.5.23 TXSCustomDateTime.SetMonth Method

Sets the value of the Month (see page 7764) property.

Pascal

```
procedure SetMonth(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMonth(const Word Value);
```

Description

SetMonth is the protected write implementation of the Month (see page 7764) property.

See Also

Month (see page 7764)

89.1.6.5.24 TXSCustomDateTime.SetSecond Method

Sets the value of the Second (see page 7764) property.

Pascal

```
procedure SetSecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetSecond(const Word Value);
```

Description

SetSecond is the protected write implementation of the Second (see page 7764) property.

See Also

Second (see page 7764)

89.1.6.5.25 TXSCustomDateTime.SetUseZeroMilliseconds Method

Sets the value of the UseZeroMilliseconds (see page 7765) property.

Pascal

```
procedure SetUseZeroMilliseconds(const Value: Boolean);
```

C++

```
__fastcall SetUseZeroMilliseconds(const Boolean Value);
```

Description

SetUseZeroMilliseconds is the protected write implementation of the UseZeroMilliseconds (see page 7765) property.

See Also

UseZeroMilliseconds (see page 7765)

89.1.6.5.26 TXSCustomDateTime.SetYear Method

Sets the value of the Year (see page 7765) property.

Pascal

```
procedure SetYear(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetYear(const int Value);
```

Description

SetYear is the protected write implementation of the Year (see page 7765) property.

See Also

Year (see page 7765)

89.1.6.6 TXSCustomDateTime Properties

The properties of the TXSCustomDateTime class are listed here.

Protected Properties

	Name	Description
	Millisecond (see page 7762)	Indicates the Millisecond of the second that the object represents.

Public Properties

	Name	Description
	AsDateTime (see page 7759)	Represents the object's value as a TDateTime (see page 3335) value.
	AsUTCDateTime (see page 7760)	Represents the object's value as a TDateTime (see page 3335) value using UTC time.
	Day (see page 7760)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7761)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7764) property.
	Hour (see page 7761)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7762)	Indicates the difference between the Hour (see page 7761) property and the hour in the local time zone.
	Minute (see page 7763)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7763)	Indicates the difference between the Minute (see page 7763) property and the minute of the hour in local time.
	Month (see page 7764)	Indicates the month of the year that the object represents.
	Second (see page 7764)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7765)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7762) is zero.
	Year (see page 7765)	Indicates the year that the object represents.

89.1.6.6.1 TXSCustomDateTime.AsDateTime Property

Represents the object's value as a TDateTime (see page 3335) value.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Use AsDateTime to read the value of the TXSCustomDateTime (see page 7747) object into a variable of type TDateTime (see page 3335) or set its value from a TDateTime (see page 3335) value. AsDateTime handles the mapping between a TDateTime (see page 3335) value and the Year (see page 7765), Month (see page 7764), Day (see page 7760), Hour (see page 7761), Minute (see page 7763), and Second (see page 7764) values.

Note: AsDateTime represents the local date-and-time, without using Coordinated Universal Time (see page 4338) (UTC). That is, AsDateTime ignores the HourOffset (see page 7762) and MinuteOffset (see page 7763) properties. To get or set the value taking the HourOffset (see page 7762) and MinuteOffset (see page 7763) properties into account, use the AsUTCDateTime (see page 7760) property instead.

See Also

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Month (see page 7764)

Second (see page 7764)

Year (see page 7765)

HourOffset (see page 7762)

MinuteOffset (see page 7763)

Millisecond (see page 7762)

AsUTCDateTime (see page 7760)

89.1.6.6.2 TXSCustomDateTime.AsUTCDateTime Property

Represents the object's value as a TDateTime (see page 3335) value using UTC time.

Pascal

```
property AsUTCDateTime: TDateTime;
```

C++

```
__property TDateTime AsUTCDateTime;
```

Description

Use AsUTCDateTime to read the value of the TXSCustomDateTime (see page 7747) object into a variable of type TDateTime (see page 3335) or set its value from a TDateTime (see page 3335) value. AsUTCDateTime handles the mapping between a TDateTime (see page 3335) value and the Year (see page 7765), Month (see page 7764), Day (see page 7760), Hour (see page 7761), Minute (see page 7763), and Second (see page 7764) values. It also makes an adjustment for the HourOffset (see page 7762) and MinuteOffset (see page 7763) properties, so that the value of AsUTCDateTime is a Coordinated Universal Time (see page 4338) (UTC) value rather than a local time value.

Note: To get or set the value as a local time value ignoring the HourOffset (see page 7762) and MinuteOffset (see page 7763) properties, use the AsDateTime (see page 7759) property instead.

See Also

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Month (see page 7764)

Second (see page 7764)

Year (see page 7765)

HourOffset (see page 7762)

MinuteOffset (see page 7763)

Millisecond (see page 7762)

AsDateTime (see page 7759)

89.1.6.6.3 TXSCustomDateTime.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 7764) value.

See Also

AsDateTime (see page 7759)

Hour (see page 7761)

Minute (see page 7763)

Month (see page 7764)

Second (see page 7764)

Year (see page 7765)

Millisecond (see page 7762)

89.1.6.6.4 TXSCustomDateTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7764) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 7762) property, but can provide greater precision.

See Also

AsDateTime (see page 7759)

Minute (see page 7763)

Second (see page 7764)

Millisecond (see page 7762)

UseZeroMilliseconds (see page 7765)

89.1.6.6.5 TXSCustomDateTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the date-and-time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (see page 4338) (UTC). To convert the hour to local time, you must add the hour offset.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Minute (see page 7763)

Month (see page 7764)

Second (see page 7764)

Year (see page 7765)

HourOffset (see page 7762)

Millisecond (see page 7762)

89.1.6.6.6 TXSCustomDateTime.HourOffset Property

Indicates the difference between the Hour (see page 7761) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 7761) property represents the hour of the day using Coordinated Universal Time (see page 4338) (UTC). HourOffset is the correction factor you can add to Hour (see page 7761) to obtain the hour of the day in local time.

See Also

Hour (see page 7761)

89.1.6.6.7 TXSCustomDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration (see page 7780).

Note: You can also use the FractionalSeconds (see page 7761) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 7761) returns a fraction of a second.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Second (see page 7764)

Year (see page 7765)

Month (see page 7764)

UseZeroMilliseconds (see page 7765)

FractionalSeconds (see page 7761)

89.1.6.6.8 TXSCustomDateTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the date-and-time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (see page 4338) (UTC). To convert the minutes to local time, you must add the minute offset.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Month (see page 7764)

Second (see page 7764)

Year (see page 7765)

MinuteOffset (see page 7763)

Millisecond (see page 7762)

89.1.6.6.9 TXSCustomDateTime.MinuteOffset Property

Indicates the difference between the Minute (see page 7763) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 7763) property represents the minute of the hour using Coordinated Universal Time (see page 4338) (UTC). MinuteOffset is the correction factor you can add to Minute (see page 7763) to obtain the minute of the hour in local time.

See Also

Minute (see page 7763)

89.1.6.6.10 TXSCustomDateTime.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Second (see page 7764)

Year (see page 7765)

Millisecond (see page 7762)

89.1.6.6.11 TXSCustomDateTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the date-and-time value. Valid Second values are 0 through 59.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Second

Year (see page 7765)

Millisecond (see page 7762)

89.1.6.6.12 TXSCustomDateTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7762) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS method does not include the milliseconds in the string encoding of this object's value when the value of the Millisecond (see page 7762) property is zero. If you set UseZeroMilliseconds, NativeToXS changes its encoding to include the milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full schema defined by W3C.

See Also

Millisecond (see page 7762)

NativeToXS

89.1.6.6.13 TXSCustomDateTime.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Second (see page 7764)

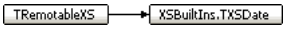
Month (see page 7764)

Millisecond (see page 7762)

89.1.7 TXSDate Class

TXSDate implements the XML schema date type so that it can be used on invokable interfaces.

Class Hierarchy



Pascal

```
TXSDate = class(TRemotableXS);
```

C++

```
class TXSDate : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDate acts as a wrapper for the simple scalar type date that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDate enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDate value to a TDateTime (see page 3335) value, use the AsDate (see page 7771) property. Other properties give the Year (see page 7772), Month (see page 7771), and Day (see page 7771) that make up that value.

On server applications, instances of TXSDate that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDate instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDate instances that it passes as input parameters, and for freeing any TXSDate instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDate

TXSDuration (see page 7780)

TXSTime (see page 7792)

89.1.7.1 TXSDate Members

The following tables list the members exposed by TXSDate.

Protected Methods

	Name	Description
	GetAsDate (see page 7768)	This is GetAsDate, a member of class TXSDate.
	GetDay (see page 7768)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 7768)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 7769)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 7769)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 7769)	This is SetDay, a member of class TXSDate.
	SetMonth (see page 7770)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 7770)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 7768)	Returns a new TXSDate (see page 7766) object that represents the same value as this one.
V	Create (see page 7767)	Creates and initializes a Create object.
V	NativeToXS (see page 7769)	Returns a string that represents the value of a TXSDate (see page 7766) instance.
V	XSToNative (see page 7770)	Assigns the properties of the TXSDate (see page 7766) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDate (see page 7771)	Represents the object's value as a TDateTime (see page 3335) value.
	Day (see page 7771)	Indicates the Day of the month that the object represents.
	Month (see page 7771)	Indicates the month of the year that the object represents.
	Year (see page 7772)	Indicates the year that the object represents.

89.1.7.2 TXSDate.Create Constructor

Creates and initializes a Create object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDate();
```

Description

Call Create indirectly, using the new keyword, to create a Create object at runtime.

89.1.7.3 TXSDate Methods

The methods of the TXSDate class are listed here.

Protected Methods

	Name	Description
	GetAsDate (see page 7768)	This is GetAsDate, a member of class TXSDate.
	GetDay (see page 7768)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 7768)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 7769)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 7769)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 7769)	This is SetDay, a member of class TXSDate.

	SetMonth (see page 7770)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 7770)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 7768)	Returns a new TXSDate (see page 7766) object that represents the same value as this one.
	NativeToXS (see page 7769)	Returns a string that represents the value of a TXSDate (see page 7766) instance.
	XSToNative (see page 7770)	Assigns the properties of the TXSDate (see page 7766) instance from the string representation of its value.

89.1.7.3.1 TXSDate.Clone Method

Returns a new TXSDate (see page 7766) object that represents the same value as this one.

Pascal

```
function Clone: TXSDate;
```

C++

```
__fastcall TXSDate Clone();
```

Description

Call Clone to obtain a new TXSDate (see page 7766) instance that represents the same time as this one. Clone returns the new TXSDate (see page 7766) object.

89.1.7.3.2 TXSDate.GetAsDate Method

Pascal

```
function GetAsDate: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDate();
```

Description

This is GetAsDate, a member of class TXSDate.

89.1.7.3.3 TXSDate.GetDay Method

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

This is GetDay, a member of class TXSDate.

89.1.7.3.4 TXSDate.GetMonth Method

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

This is GetMonth, a member of class TXSDate.

89.1.7.3.5 TXSDate.GetYear Method

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

This is GetYear, a member of class TXSDate.

89.1.7.3.6 TXSDate.NativeToXS Method

Returns a string that represents the value of a TXSDate (see page 7766) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDate (see page 7766) in a SOAP packet.

See Also

XSToNative (see page 7770)

89.1.7.3.7 TXSDate.SetAsDate Method

Pascal

```
procedure SetAsDate(const Value: TDateTime);
```

C++

```
__fastcall SetAsDate(const TDateTime Value);
```

Description

This is SetAsDate, a member of class TXSDate.

89.1.7.3.8 TXSDate.SetDay Method

Pascal

```
procedure SetDay(const Value: Word);
```

C++

```
__fastcall SetDay(const Word Value);
```

Description

This is SetDay, a member of class TXSDate.

89.1.7.3.9 TXSDate.SetMonth Method

Pascal

```
procedure SetMonth(const Value: Word);
```

C++

```
__fastcall SetMonth(const Word Value);
```

Description

This is SetMonth, a member of class TXSDate.

89.1.7.3.10 TXSDate.SetYear Method

Pascal

```
procedure SetYear(const Value: Integer);
```

C++

```
__fastcall SetYear(const int Value);
```

Description

This is SetYear, a member of class TXSDate.

89.1.7.3.11 TXSDate.XSToNative Method

Assigns the properties of the TXSDate (see page 7766) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a date value. It uses this information to set the properties of the TXSDate (see page 7766) object.

See Also

NativeToXS (see page 7769)

89.1.7.4 TXSDate Properties

The properties of the TXSDate class are listed here.

Public Properties

	Name	Description
	AsDate (see page 7771)	Represents the object's value as a TDateTime (see page 3335) value.
	Day (see page 7771)	Indicates the Day of the month that the object represents.
	Month (see page 7771)	Indicates the month of the year that the object represents.

	Year (see page 7772)	Indicates the year that the object represents.

89.1.7.4.1 TXSDate.AsDate Property

Represents the object's value as a TDateTime (see page 3335) value.

Pascal

```
property AsDate: TDateTime;
```

C++

```
__property TDateTime AsDate;
```

Description

Use AsDate to read the value of the TXSDate (see page 7766) object into a variable of type TDateTime (see page 3335) or set this objects value from a TDateTime (see page 3335) value. AsDate handles the mapping between a TDateTime (see page 3335) value and the Year (see page 7772), Month (see page 7771), and Day (see page 7771) properties. When reading AsDate, the resulting TDateTime (see page 3335) value has a time portion of 0. When setting AsDate, the time portion is ignored.

See Also

Day (see page 7771)

Month (see page 7771)

Year (see page 7772)

89.1.7.4.2 TXSDate.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 7771) value.

See Also

AsDate (see page 7771)

Month (see page 7771)

Year (see page 7772)

89.1.7.4.3 TXSDate.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDate (see page 7771)

Day (see page 7771)

Year (see page 7772)

89.1.7.4.4 TXSDate.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDate (see page 7771)

Day (see page 7771)

Month (see page 7771)

89.1.8 TXSDateTime Class

TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDateTime = class(TXSCustomDateTime);
```

C++

```
class TXSDateTime : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDateTime acts as a wrapper for the simple scalar type datetime that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDateTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDateTime value to a TDateTime (see page 3335) value, use the AsDateTime (see page 7759) property. The resulting value represents the time portion using Coordinated Universal Time (see page 4338) (UTC). To convert that to a local time zone, adjust the hour and minute portions using the HourOffset (see page 7762) and MinuteOffset (see page 7763) properties.

On server applications, instances of TXSDateTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDateTime instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDateTime instances that it passes as input parameters, and for freeing any TXSDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDate (see page 7766)

TXSDuration (see page 7780)

TXSTime (see page 7792)

89.1.8.1 TXSDateTime Members

The following tables list the members exposed by TXSDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 7750)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7751)	Returns the value of the AsDateTime (see page 7759) property.
	GetAsUTCDateTime (see page 7751)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7751)	Returns the value of the Day (see page 7760) property.
	GetFractionalSeconds (see page 7752)	Returns the value of the FractionalSeconds (see page 7761) property.
	GetHour (see page 7752)	Returns the value of the Hour (see page 7761) property.
	GetHourOffset (see page 7752)	Returns the value of the HourOffset (see page 7762) property.
	GetMillisecond (see page 7752)	Returns the value of the Millisecond (see page 7762) property.
	GetMinute (see page 7753)	Returns the value of the Minute (see page 7763) property.
	GetMinuteOffset (see page 7753)	Returns the value of the MinuteOffset (see page 7763) property.
	GetMonth (see page 7753)	Returns the value of the Month (see page 7764) property.
	GetSecond (see page 7754)	Returns the value of the Second (see page 7764) property.
	GetUseZeroMilliseconds (see page 7754)	Returns the value of the UseZeroMilliseconds (see page 7765) property.
	GetYear (see page 7754)	Returns the value of the Year (see page 7765) property.
	SetAsDateTime (see page 7755)	Sets the value of the AsDateTime (see page 7759) property.
	SetAsUTCDateTime (see page 7755)	Sets the value of the AsUTCDateTime (see page 7760) property.
	SetDay (see page 7755)	Sets the value of the Day (see page 7760) property.
	SetFractionalSeconds (see page 7755)	Sets the value of the FractionalSeconds (see page 7761) property.
	SetHour (see page 7756)	Sets the value of the Hour (see page 7761) property.
	SetHourOffset (see page 7756)	Sets the value of the HourOffset (see page 7762) property.
	SetMillisecond (see page 7756)	Sets the value of the Millisecond (see page 7762) property.
	SetMinute (see page 7757)	Sets the value of the Minute (see page 7763) property.
	SetMinuteOffset (see page 7757)	Sets the value of the MinuteOffset (see page 7763) property.

	SetMonth (see page 7757)	Sets the value of the Month (see page 7764) property.
	SetSecond (see page 7758)	Sets the value of the Second (see page 7764) property.
	SetUseZeroMilliseconds (see page 7758)	Sets the value of the UseZeroMilliseconds (see page 7765) property.
	SetYear (see page 7758)	Sets the value of the Year (see page 7765) property.

Public Methods

	Name	Description
	Clone (see page 7774)	Returns a new TXSDateTime (see page 7772) object that represents the same value as this one.
	CompareDateTimeParam (see page 7775)	This is CompareDateTimeParam, a member of class TXSDateTime.
	Create (see page 7749)	Creates and initializes a TXSCustomDateTime (see page 7747) object.
	Destroy (see page 7749)	Frees the memory associated with the TXSCustomDateTime (see page 7747) object.
	NativeToXS (see page 7775)	Returns a string that represents the value of a TXSDateTime (see page 7772) instance.
	XSToNative (see page 7775)	Assigns the properties of the TXSDateTime (see page 7772) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 7759)	Represents the object's value as a TDateTime (see page 3335) value.
	AsUTCDateTime (see page 7760)	Represents the object's value as a TDateTime (see page 3335) value using UTC time.
	Day (see page 7760)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7761)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7764) property.
	Hour (see page 7761)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7762)	Indicates the difference between the Hour (see page 7761) property and the hour in the local time zone.
	Millisecond (see page 7776)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7763)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7763)	Indicates the difference between the Minute (see page 7763) property and the minute of the hour in local time.
	Month (see page 7764)	Indicates the month of the year that the object represents.
	Second (see page 7764)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7765)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7762) is zero.
	Year (see page 7765)	Indicates the year that the object represents.

89.1.8.2 TXSDateTime Methods

The methods of the TXSDateTime class are listed here.

Public Methods

	Name	Description
	Clone (see page 7774)	Returns a new TXSDateTime (see page 7772) object that represents the same value as this one.
	CompareDateTimeParam (see page 7775)	This is CompareDateTimeParam, a member of class TXSDateTime.
	NativeToXS (see page 7775)	Returns a string that represents the value of a TXSDateTime (see page 7772) instance.
	XSToNative (see page 7775)	Assigns the properties of the TXSDateTime (see page 7772) instance from the string representation of its value.

89.1.8.2.1 TXSDateTime.Clone Method

Returns a new TXSDateTime (see page 7772) object that represents the same value as this one.

Pascal

```
function Clone: TXSDateTime;
```

C++

```
__fastcall TXSDateTime Clone();
```

Description

Call Clone to obtain a new TXSDateTime (see page 7772) instance that represents the same date and time as this one. Clone returns the new TXSDateTime (see page 7772) object.

89.1.8.2.2 TXSDateTime.CompareDateTimeParam Method

Pascal

```
function CompareDateTimeParam(const Value1: TXSDateTime; const Value2: TXSDateTime):
TXSDuration;
```

C++

```
__fastcall TXSDuration CompareDateTimeParam(const TXSDateTime Value1, const TXSDateTime
Value2);
```

Description

This is CompareDateTimeParam, a member of class TXSDateTime.

89.1.8.2.3 TXSDateTime.NativeToXS Method

Returns a string that represents the value of a TXSDateTime (see page 7772) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDateTime (see page 7772) in a SOAP packet.

See Also

XSToNative (see page 7775)

UseZeroMilliseconds (see page 7765)

89.1.8.2.4 TXSDateTime.XSToNative Method

Assigns the properties of the TXSDateTime (see page 7772) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a datetime value. It uses this information to set the properties of the TXSDateTime (see page 7772) object.

See Also

NativeToXS (see page 7775)

89.1.8.3 TXSDateTime Properties

The properties of the TXSDateTime class are listed here.

Public Properties

	Name	Description
	Millisecond (see page 7776)	Indicates the Millisecond of the second that the object represents.

89.1.8.3.1 TXSDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration.

Note: You can also use the FractionalSeconds property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds returns a fraction of a second.

See Also

AsDateTime (see page 7759)

Day (see page 7760)

Hour (see page 7761)

Minute (see page 7763)

Second (see page 7764)

Year (see page 7765)

Month (see page 7764)

UseZeroMilliseconds (see page 7765)

FractionalSeconds (see page 7761)

89.1.9 TXSDecimal Class

TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDecimal = class(TRemotableXS);
```

C++

```
class TXSDecimal : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDecimal acts as a wrapper for the simple scalar decimal type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDecimal enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDecimal value to a TBcd value, use the AsBcd (see page 7779) property.

On server applications, instances of TXSDecimal that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDecimal instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDecimal instances that it passes as input parameters, and for freeing any TXSDecimal instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSHexBinary (see page 7784)

EXSDecimalException (see page 7740)

89.1.9.1 TXSDecimal Members

The following tables list the members exposed by TXSDecimal.

Public Methods

	Name	Description
	NativeToXS (see page 7778)	Returns a string that represents the value of a TXSDecimal (see page 7776) instance.
	XSToNative (see page 7778)	Assigns a string as the value of the TXSDecimal (see page 7776) instance.

Public Properties

	Name	Description
	AsBcd (see page 7779)	Represents the object's value as a TBcd value.
	DecimalString (see page 7779)	Represents the object's value as a string.

89.1.9.2 TXSDecimal Methods

The methods of the TXSDecimal class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7778)	Returns a string that represents the value of a TXSDecimal (see page 7776) instance.
	XSToNative (see page 7778)	Assigns a string as the value of the TXSDecimal (see page 7776) instance.

89.1.9.2.1 TXSDecimal.NativeToXS Method

Returns a string that represents the value of a TXSDecimal (see page 7776) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDecimal (see page 7776) object in a SOAP packet. That is, NativeToXS converts the DecimalString (see page 7779) property to a WideString (see page 3476) and returns the result.

See Also

XSToNative (see page 7778)

DecimalString (see page 7779)

89.1.9.2.2 TXSDecimal.XSToNative Method

Assigns a string as the value of the TXSDecimal (see page 7776) instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative sets the value of this TXSDecimal (see page 7776) object to a string that represents a formatted decimal value.

Data is the SOAP encoding of a decimal value. It becomes the value of the DecimalString (see page 7779) property.

See Also

NativeToXS (see page 7778)

DecimalString (see page 7779)

89.1.9.3 TXSDecimal Properties

The properties of the TXSDecimal class are listed here.

Public Properties

	Name	Description
	AsBcd (see page 7779)	Represents the object's value as a TBcd value.
	DecimalString (see page 7779)	Represents the object's value as a string.

89.1.9.3.1 TXSDecimal.AsBcd Property

Represents the object's value as a TBcd value.

Pascal

```
property AsBcd: TBcd;
```

C++

```
__property TBcd AsBcd;
```

Description

Use AsBcd to get or set this object's value as a TBcd type. TXSDecimal (see page 7776) converts between the internal string representation that is the value of the DecimalString (see page 7779) property and a TBcd value when the application gets or sets this property. Thus, if the string representation is ill-formed, reading AsBcd can cause an EXSDecimalException (see page 7740) exception. Similarly, setting AsBcd can cause an EXSDecimalException (see page 7740) exception if the TBcd value has an invalid scale or precision.

See Also

BcdToStr

TryStrToBCD

DecimalString (see page 7779)

EXSDecimalException (see page 7740)

89.1.9.3.2 TXSDecimal.DecimalString Property

Represents the object's value as a string.

Pascal

```
property DecimalString: string;
```

C++

```
__property UnicodeString DecimalString;
```

Description

Use DecimalString to obtain the value of the TXSDecimal (see page 7776) object as a string. This value is the same as the string that appears in XML documents to represent the decimal value. To work with this object's value as a TBcd value, use the AsBcd (see page 7779) property instead.

Note: If you do not need the precision of binary-coded decimal values, you can use global functions such as StrToFloat (see page 4324) and FloatToStr (see page 4224) or StrToCurr (see page 4319) and CurrToStr (see page 4150) to convert this string to a floating-point representation.

See Also

StrToCurr (see page 4319)

CurrToStr (see page 4150)

StrToFloat (see page 4324)

FloatToStr (see page 4224)

AsBcd (see page 7779)

89.1.10 TXSDuration Class

TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDuration = class(TXSCustomDateTime);
```

C++

```
class TXSDuration : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDuration acts as a wrapper for the simple scalar type duration that W3C defines for XML schemas. That is, TXSDuration represents the interval between two points in time. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDuration enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDuration value to a TDateTime (see page 3335) value, use the AsDateTime (see page 7759) property.

On server applications, instances of TXSDuration that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDuration instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDuration instances that it passes as input parameters, and for freeing any TXSDuration instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDate (see page 7766)

TXSDuration

TXSTime (see page 7792)

89.1.10.1 TXSDuration Members

The following tables list the members exposed by TXSDuration.

Protected Fields

	Name	Description
	FDateTime (see page 7750)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7751)	Returns the value of the AsDateTime (see page 7759) property.
	GetAsUTCDateTime (see page 7751)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7751)	Returns the value of the Day (see page 7760) property.
	GetDecimalSecond (see page 7782)	This is GetDecimalSecond, a member of class TXSDuration.
	GetFractionalSeconds (see page 7752)	Returns the value of the FractionalSeconds (see page 7761) property.
	GetHour (see page 7752)	Returns the value of the Hour (see page 7761) property.
	GetHourOffset (see page 7752)	Returns the value of the HourOffset (see page 7762) property.
	GetMillisecond (see page 7752)	Returns the value of the Millisecond (see page 7762) property.
	GetMinute (see page 7753)	Returns the value of the Minute (see page 7763) property.
	GetMinuteOffset (see page 7753)	Returns the value of the MinuteOffset (see page 7763) property.
	GetMonth (see page 7753)	Returns the value of the Month (see page 7764) property.
	GetSecond (see page 7754)	Returns the value of the Second (see page 7764) property.
	GetUseZeroMilliseconds (see page 7754)	Returns the value of the UseZeroMilliseconds (see page 7765) property.
	GetYear (see page 7754)	Returns the value of the Year (see page 7765) property.
	SetAsDateTime (see page 7755)	Sets the value of the AsDateTime (see page 7759) property.
	SetAsUTCDateTime (see page 7755)	Sets the value of the AsUTCDateTime (see page 7760) property.
	SetDay (see page 7755)	Sets the value of the Day (see page 7760) property.
	SetDecimalSecond (see page 7783)	This is SetDecimalSecond, a member of class TXSDuration.
	SetFractionalSeconds (see page 7755)	Sets the value of the FractionalSeconds (see page 7761) property.
	SetHour (see page 7756)	Sets the value of the Hour (see page 7761) property.
	SetHourOffset (see page 7756)	Sets the value of the HourOffset (see page 7762) property.
	SetMillisecond (see page 7756)	Sets the value of the Millisecond (see page 7762) property.
	SetMinute (see page 7757)	Sets the value of the Minute (see page 7763) property.
	SetMinuteOffset (see page 7757)	Sets the value of the MinuteOffset (see page 7763) property.
	SetMonth (see page 7757)	Sets the value of the Month (see page 7764) property.
	SetSecond (see page 7758)	Sets the value of the Second (see page 7764) property.
	SetUseZeroMilliseconds (see page 7758)	Sets the value of the UseZeroMilliseconds (see page 7765) property.
	SetYear (see page 7758)	Sets the value of the Year (see page 7765) property.

Protected Properties

	Name	Description
	Millisecond (see page 7762)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
	Create (see page 7782)	Creates and initializes a TXSDuration (see page 7780) object.
	Destroy (see page 7749)	Frees the memory associated with the TXSCustomDateTime (see page 7747) object.
	NativeToXS (see page 7783)	Returns a string that represents the value of a TXSDuration (see page 7780) instance.
	XSToNative (see page 7783)	Assigns the properties of the TXSDuration (see page 7780) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 7759)	Represents the object's value as a TDateTime (see page 3335) value.
	AsUTCDateTime (see page 7760)	Represents the object's value as a TDateTime (see page 3335) value using UTC time.
	Day (see page 7760)	Indicates the Day of the month that the object represents.
	DecimalSecond (see page 7784)	Indicates the number of seconds, including fractional seconds, in the duration.
	FractionalSeconds (see page 7761)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7764) property.
	Hour (see page 7761)	Indicates the hour of the day that the object represents.

	HourOffset (see page 7762)	Indicates the difference between the Hour (see page 7761) property and the hour in the local time zone.
	Minute (see page 7763)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7763)	Indicates the difference between the Minute (see page 7763) property and the minute of the hour in local time.
	Month (see page 7764)	Indicates the month of the year that the object represents.
	Second (see page 7764)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7765)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7762) is zero.
	Year (see page 7765)	Indicates the year that the object represents.

89.1.10.2 TXSDuration.Create Constructor

Creates and initializes a TXSDuration (see page 7780) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDuration();
```

Description

Call Create to create a TXSDuration (see page 7780) object at runtime.

89.1.10.3 TXSDuration Methods

The methods of the TXSDuration class are listed here.

Protected Methods

	Name	Description
	GetDecimalSecond (see page 7782)	This is GetDecimalSecond, a member of class TXSDuration.
	SetDecimalSecond (see page 7783)	This is SetDecimalSecond, a member of class TXSDuration.

Public Methods

	Name	Description
	NativeToXS (see page 7783)	Returns a string that represents the value of a TXSDuration (see page 7780) instance.
	XSToNative (see page 7783)	Assigns the properties of the TXSDuration (see page 7780) instance from the string representation of its value.

89.1.10.3.1 TXSDuration.GetDecimalSecond Method

Pascal

```
function GetDecimalSecond: Double;
```

C++

```
__fastcall Double GetDecimalSecond();
```

Description

This is GetDecimalSecond, a member of class TXSDuration.

89.1.10.3.2 TXSDuration.NativeToXS Method

Returns a string that represents the value of a TXSDuration (see page 7780) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDuration (see page 7780) in a SOAP packet.

See Also

XSToNative (see page 7783)

89.1.10.3.3 TXSDuration.SetDecimalSecond Method

Pascal

```
procedure SetDecimalSecond(const Value: Double);
```

C++

```
__fastcall SetDecimalSecond(const Double Value);
```

Description

This is SetDecimalSecond, a member of class TXSDuration.

89.1.10.3.4 TXSDuration.XSToNative Method

Assigns the properties of the TXSDuration (see page 7780) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a duration value. It uses this information to set the properties of the TXSDuration (see page 7780) object.

See Also

NativeToXS (see page 7783)

89.1.10.4 TXSDuration Properties

The properties of the TXSDuration class are listed here.

Public Properties

	Name	Description
	DecimalSecond (see page 7784)	Indicates the number of seconds, including fractional seconds, in the duration.

89.1.10.4.1 TXSDuration.DecimalSecond Property

Indicates the number of seconds, including fractional seconds, in the duration.

Pascal

```
property DecimalSecond: Double;
```

C++

```
__property Double DecimalSecond;
```

Description

Use the DecimalSecond property to determine the size of the duration with a granularity finer than the Seconds property provides. DecimalSecond represents the number of seconds in the duration (above and beyond the years, months, days, hours and minutes represented by the other TXSDuration (see page 7780) properties), including any fractional seconds.

See Also

Second (see page 7764)

FractionalSeconds (see page 7761)

89.1.11 TXSHexBinary Class

TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSHexBinary = class(TRemotableXS);
```

C++

```
class TXSHexBinary : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSHexBinary acts as a wrapper for the simple scalar hexBinary type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSHexBinary enables this marshaling because it is a descendant of TRemotable.

Use the AsByteArray (see page 7786) property to get or set the value of TXSHexBinary as an array of bytes. You can also set the value of TXSHexBinary from an integer using the global IntToHex (see page 4259) function to convert an integer into a hexadecimal string and setting the HexBinaryString (see page 7787) property.

On server applications, instances of TXSHexBinary that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSHexBinary instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSHexBinary instances that it passes as input

parameters, and for freeing any TXSHexBinary instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDecimal (see page 7776)

HexToBin

BinToHex

IntToHex

EXSHexBinaryException (see page 7740)

89.1.11.1 TXSHexBinary Members

The following tables list the members exposed by TXSHexBinary.

Public Methods

	Name	Description
	NativeToXS (see page 7785)	Returns a string that represents the value of a TXSHexBinary (see page 7784) instance.
	XSToNative (see page 7786)	Assigns a string as the value of the TXSHexBinary (see page 7784) instance.

Public Properties

	Name	Description
	AsByteArray (see page 7786)	Represents the object's value as an array of bytes.
	HexBinaryString (see page 7787)	This is HexBinaryString, a member of class TXSHexBinary.

89.1.11.2 TXSHexBinary Methods

The methods of the TXSHexBinary class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7785)	Returns a string that represents the value of a TXSHexBinary (see page 7784) instance.
	XSToNative (see page 7786)	Assigns a string as the value of the TXSHexBinary (see page 7784) instance.

89.1.11.2.1 TXSHexBinary.NativeToXS Method

Returns a string that represents the value of a TXSHexBinary (see page 7784) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSHexBinary (see page 7784) object in a SOAP packet. That is, NativeToXS converts the HexBinaryString (see page 7787) property to a WideString (see page 3476) and returns the result.

See Also

XSToNative (see page 7786)

hexbinarystring (see page 7787)

89.1.11.2.2 TXSHexBinary.XSToNative Method

Assigns a string as the value of the TXSHexBinary (see page 7784) instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative sets the value of this TXSHexBinary (see page 7784) object to a string that represents a formatted hexadecimal value.

Data is the SOAP encoding of a hexadecimal value. It becomes the value of the HexBinaryString (see page 7787) property.

See Also

NativeToXS (see page 7785)

hexbinarystring (see page 7787)

89.1.11.3 TXSHexBinary Properties

The properties of the TXSHexBinary class are listed here.

Public Properties

	Name	Description
	AsByteArray (see page 7786)	Represents the object's value as an array of bytes.
	HexBinaryString (see page 7787)	This is HexBinaryString, a member of class TXSHexBinary.

89.1.11.3.1 TXSHexBinary.AsByteArray Property

Represents the object's value as an array of bytes.

Pascal

```
property AsByteArray: TByteDynArray;
```

C++

```
__property TByteDynArray AsByteArray;
```

Description

Use AsByteArray to obtain the value of the TXSHexBinary (see page 7784) object as an array of bytes. This value is obtained from the HexBinaryString (see page 7787) property using the global HexToBin function. When you set AsByteArray, TXSHexBinary (see page 7784) uses the global BinToHex function to convert the binary value to a string and sets the HexBinaryString (see page 7787) property.

See Also

BinToHex

HexToBin

hexbinarystring (see page 7787)

89.1.11.3.2 TXSHexBinary.HexBinaryString Property

Pascal

```
property HexBinaryString: string;
```

C++

```
__property UnicodeString HexBinaryString;
```

Description

This is HexBinaryString, a member of class TXSHexBinary.

89.1.12 TXSInteger Class

Class Hierarchy

Pascal

```
TXSInteger = class(TRemotableXS);
```

C++

```
class TXSInteger : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSInteger.

89.1.12.1 TXSInteger Members

The following tables list the members exposed by TXSInteger.

Public Methods

	Name	Description
	NativeToXS (see page 7788)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7788)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsInteger (see page 7789)	This is AsInteger, a member of class TXSInteger.

89.1.12.2 TXSInteger Methods

The methods of the TXSInteger class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7788)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7788)	Assigns the value of a string to the TRemotableXS instance.

89.1.12.2.1 TXSInteger.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

89.1.12.2.2 TXSInteger.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

89.1.12.3 TXSInteger Properties

The properties of the TXSInteger class are listed here.

Public Properties

	Name	Description
	AsInteger (see page 7789)	This is AsInteger, a member of class TXSInteger.

89.1.12.3.1 TXSInteger.AsInteger Property

Pascal

```
property AsInteger: Integer;
```

C++

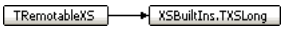
```
__property int AsInteger;
```

Description

This is AsInteger, a member of class TXSInteger.

89.1.13 TXSLong Class

Class Hierarchy



Pascal

```
TXSLong = class(TRemotableXS);
```

C++

```
class TXSLong : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSLong.

89.1.13.1 TXSLong Members

The following tables list the members exposed by TXSLong.

Public Methods

	Name	Description
	NativeToXS (see page 7790)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7790)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsLong (see page 7791)	This is AsLong, a member of class TXSLong.

89.1.13.2 TXSLong Methods

The methods of the TXSLong class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7790)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7790)	Assigns the value of a string to the TRemotableXS instance.

89.1.13.2.1 TXSLong.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

89.1.13.2.2 TXSLong.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

89.1.13.3 TXSLong Properties

The properties of the TXSLong class are listed here.

Public Properties

	Name	Description
	AsLong (see page 7791)	This is AsLong, a member of class TXSLong.

89.1.13.3.1 TXSLong.AsLong Property

Pascal

```
property AsLong: Int64;
```

C++

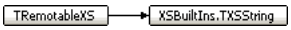
```
__property Int64 AsLong;
```

Description

This is AsLong, a member of class TXSLong.

89.1.14 TXSString Class

Class Hierarchy



Pascal

```
TXSString = class(TRemotableXS);
```

C++

```
class TXSString : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSString.

89.1.14.1 TXSString Members

The following tables list the members exposed by TXSString.

Public Methods

	Name	Description
	NativeToXS (see page 7792)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7792)	Assigns the value of a string to the TRemotableXS instance.

89.1.14.2 TXSString Methods

The methods of the TXSString class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7792)	Returns a string that represents the value of a TRemotableXS instance.

	XSToNative (see page 7792)	Assigns the value of a string to the TRemotableXS instance.

89.1.14.2.1 TXSString.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

89.1.14.2.2 TXSString.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

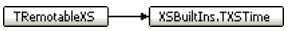
NativeToXS

Using Remotable Objects

89.1.15 TXSTime Class

TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Class Hierarchy



Pascal

```
TXSTime = class(TRemotableXS);
```

C++

```
class TXSTime : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSTime acts as a wrapper for the simple scalar type time that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSTime value to a TDateTime (see page 3335) value, use the AsTime (see page 7800) property. Other properties give the Hour (see page 7801), Minute (see page 7802), Second (see page 7803), and Millisecond (see page 7802) that make up that value. These properties represent the time using Coordinated Universal Time (see page 4338) (UTC). To convert that to a local time zone, adjust the hour and minute portions using the HourOffset (see page 7802) and MinuteOffset (see page 7803) properties.

On server applications, instances of TXSTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSTime instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSTime instances that it passes as input parameters, and for freeing any TXSTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3505)

TXSDate (see page 7766)

TXSDuration (see page 7780)

TXSDateTime (see page 7772)

89.1.15.1 TXSTime Members

The following tables list the members exposed by TXSTime.

Protected Methods

	Name	Description
	GetAsTime (see page 7795)	This is GetAsTime, a member of class TXSTime.
	GetFractionalSeconds (see page 7795)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 7796)	This is GetFractionalSecondString, a member of class TXSTime.
	GetHour (see page 7796)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 7796)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 7796)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 7796)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 7797)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 7797)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 7797)	This is SetAsTime, a member of class TXSTime.

	SetFractionalSeconds (see page 7798)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 7798)	This is SetHour, a member of class TXSTime.
	SetHourOffset (see page 7798)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 7798)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 7799)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 7799)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 7799)	This is SetSecond, a member of class TXSTime.

Protected Properties

	Name	Description
R	FractionalSecondString (see page 7801)	This is FractionalSecondString, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 7795)	Returns a new TXSTime (see page 7792) object that represents the same value as this one.
V	NativeToXS (see page 7797)	Returns a string that represents the value of a TXSTime (see page 7792) instance.
V	XSToNative (see page 7799)	Assigns the properties of the TXSTime (see page 7792) instance from the string representation of its value.

Public Properties

	Name	Description
	AsTime (see page 7800)	Represents the object's value as a TDateTime (see page 3335) value.
	FractionalSeconds (see page 7801)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7803) property.
	Hour (see page 7801)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7802)	Indicates the difference between the Hour (see page 7801) property and the hour in the local time zone.
	Millisecond (see page 7802)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7802)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7803)	Indicates the difference between the Minute (see page 7802) property and the minute of the hour in local time.
	Second (see page 7803)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7804)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7802) and FractionalSeconds (see page 7801) is zero.

89.1.15.2 TXSTime Methods

The methods of the TXSTime class are listed here.

Protected Methods

	Name	Description
	GetAsTime (see page 7795)	This is GetAsTime, a member of class TXSTime.
	GetFractionalSeconds (see page 7795)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 7796)	This is GetFractionalSecondString, a member of class TXSTime.
	GetHour (see page 7796)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 7796)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 7796)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 7796)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 7797)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 7797)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 7797)	This is SetAsTime, a member of class TXSTime.
	SetFractionalSeconds (see page 7798)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 7798)	This is SetHour, a member of class TXSTime.

	SetHourOffset (see page 7798)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 7798)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 7799)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 7799)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 7799)	This is SetSecond, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 7795)	Returns a new TXSTime (see page 7792) object that represents the same value as this one.
V	NativeToXS (see page 7797)	Returns a string that represents the value of a TXSTime (see page 7792) instance.
V	XSToNative (see page 7799)	Assigns the properties of the TXSTime (see page 7792) instance from the string representation of its value.

89.1.15.2.1 TXSTime.Clone Method

Returns a new TXSTime (see page 7792) object that represents the same value as this one.

Pascal

```
function Clone: TXSTime;
```

C++

```
__fastcall TXSTime Clone();
```

Description

Call Clone to obtain a new TXSTime (see page 7792) instance that represents the same time as this one. Clone returns the new TXSTime (see page 7792) object.

89.1.15.2.2 TXSTime.GetAsTime Method

Pascal

```
function GetAsTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsTime();
```

Description

This is GetAsTime, a member of class TXSTime.

89.1.15.2.3 TXSTime.GetFractionalSeconds Method

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

This is GetFractionalSeconds, a member of class TXSTime.

89.1.15.2.4 TXSTime.GetFractionalSecondString Method

Pascal

```
function GetFractionalSecondString: string;
```

C++

```
__fastcall UnicodeString GetFractionalSecondString();
```

Description

This is GetFractionalSecondString, a member of class TXSTime.

89.1.15.2.5 TXSTime.GetHour Method

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

This is GetHour, a member of class TXSTime.

89.1.15.2.6 TXSTime.GetHourOffset Method

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

This is GetHourOffset, a member of class TXSTime.

89.1.15.2.7 TXSTime.GetMillisecond Method

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

This is GetMillisecond, a member of class TXSTime.

89.1.15.2.8 TXSTime.GetMinute Method

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

This is GetMinute, a member of class TXSTime.

89.1.15.2.9 TXSTime.GetMinuteOffset Method

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

This is GetMinuteOffset, a member of class TXSTime.

89.1.15.2.10 TXSTime.GetSecond Method

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

This is GetSecond, a member of class TXSTime.

89.1.15.2.11 TXSTime.NativeToXS Method

Returns a string that represents the value of a TXSTime (see page 7792) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSTime (see page 7792) object in a SOAP packet.

See Also

XSToNative (see page 7799)

UseZeroMilliseconds (see page 7804)

89.1.15.2.12 TXSTime.SetAsTime Method

Pascal

```
procedure SetAsTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsTime(const TDateTime Value);
```

Description

This is SetAsTime, a member of class TXSTime.

89.1.15.2.13 TXSTime.SetFractionalSeconds Method

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

This is SetFractionalSeconds, a member of class TXSTime.

89.1.15.2.14 TXSTime.SetHour Method

Pascal

```
procedure SetHour(const Value: Word);
```

C++

```
__fastcall SetHour(const Word Value);
```

Description

This is SetHour, a member of class TXSTime.

89.1.15.2.15 TXSTime.SetHourOffset Method

Pascal

```
procedure SetHourOffset(const Value: SmallInt);
```

C++

```
__fastcall SetHourOffset(const SmallInt Value);
```

Description

This is SetHourOffset, a member of class TXSTime.

89.1.15.2.16 TXSTime.SetMillisecond Method

Pascal

```
procedure SetMillisecond(const Value: Word);
```

C++

```
__fastcall SetMillisecond(const Word Value);
```

Description

This is SetMillisecond, a member of class TXSTime.

89.1.15.2.17 TXSTime.SetMinute Method

Pascal

```
procedure SetMinute(const Value: Word);
```

C++

```
__fastcall SetMinute(const Word Value);
```

Description

This is SetMinute, a member of class TXSTime.

89.1.15.2.18 TXSTime.SetMinuteOffset Method

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt);
```

C++

```
__fastcall SetMinuteOffset(const SmallInt Value);
```

Description

This is SetMinuteOffset, a member of class TXSTime.

89.1.15.2.19 TXSTime.SetSecond Method

Pascal

```
procedure SetSecond(const Value: Word);
```

C++

```
__fastcall SetSecond(const Word Value);
```

Description

This is SetSecond, a member of class TXSTime.

89.1.15.2.20 TXSTime.XSToNative Method

Assigns the properties of the TXSTime (see page 7792) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a time value. It uses this information to set the properties of the TXSTime (see page 7792) object.

See Also

NativeToXS (see page 7797)

89.1.15.3 TXSTime Properties

The properties of the TXSTime class are listed here.

Protected Properties

	Name	Description
	FractionalSecondString (see page 7801)	This is FractionalSecondString, a member of class TXSTime.

Public Properties

	Name	Description
	AsTime (see page 7800)	Represents the object's value as a TDateTime (see page 3335) value.
	FractionalSeconds (see page 7801)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7803) property.
	Hour (see page 7801)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7802)	Indicates the difference between the Hour (see page 7801) property and the hour in the local time zone.
	Millisecond (see page 7802)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7802)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7803)	Indicates the difference between the Minute (see page 7802) property and the minute of the hour in local time.
	Second (see page 7803)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7804)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7802) and FractionalSeconds (see page 7801) is zero.

89.1.15.3.1 TXSTime.AsTime Property

Represents the object's value as a TDateTime (see page 3335) value.

Pascal

```
property AsTime: TDateTime;
```

C++

```
__property TDateTime AsTime;
```

Description

Use AsTime to read the value of the TXSTime (see page 7792) object into a variable of type TDateTime (see page 3335) or set its value from a TDateTime (see page 3335) value. AsTime handles the mapping between a TDateTime (see page 3335) value and the Hour (see page 7801), Minute (see page 7802), and Second (see page 7803) properties. When reading AsTime, the resulting TDateTime (see page 3335) value is given a data portion of 0. When setting AsTime, the data portion is ignored.

Note: AsTime represents the time using Coordinated Universal Time (see page 4338) (UTC). To convert this to a local time, add the offsets specified by HourOffset (see page 7802) and MinuteOffset (see page 7803) to the hours and minutes.

See Also

Hour (see page 7801)

Minute (see page 7802)

Second (see page 7803)

HourOffset (see page 7802)

MinuteOffset (see page 7803)

89.1.15.3.2 TXSTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7803) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 7802) property, but can provide greater precision.

See Also

AsTime (see page 7800)

Minute (see page 7802)

Second (see page 7803)

Millisecond (see page 7802)

UseZeroMilliseconds (see page 7804)

89.1.15.3.3 TXSTime.FractionalSecondString Property

Pascal

```
property FractionalSecondString: string;
```

C++

```
__property UnicodeString FractionalSecondString;
```

Description

This is FractionalSecondString, a member of class TXSTime.

89.1.15.3.4 TXSTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (see page 4338) (UTC). To convert the hour to local time, you must add the value of the HourOffset (see page 7802) property.

See Also

AsTime (see page 7800)

Minute (see page 7802)

Second (see page 7803)

Millisecond (see page 7802)

HourOffset (see page 7802)

89.1.15.3.5 TXSTime.HourOffset Property

Indicates the difference between the Hour (see page 7801) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 7801) property represents the hour of the day using Coordinated Universal Time (see page 4338) (UTC). HourOffset is the correction factor you can add to Hour (see page 7801) to obtain the hour of the day in local time.

See Also

Hour (see page 7801)

89.1.15.3.6 TXSTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the time value. Valid Millisecond values are 1 through 59.

Note: You can also use the FractionalSeconds (see page 7801) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 7801) returns a fraction of a second.

See Also

AsTime (see page 7800)

Hour (see page 7801)

Minute (see page 7802)

Second (see page 7803)

UseZeroMilliseconds (see page 7804)

FractionalSeconds (see page 7801)

89.1.15.3.7 TXSTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (see page 4338) (UTC). To convert the minutes to local time, you must add the value of the MinuteOffset (see page 7803) property.

See Also

AsTime (see page 7800)

Hour (see page 7801)

Second (see page 7803)

Millisecond (see page 7802)

MinuteOffset (see page 7803)

89.1.15.3.8 TXSTime.MinuteOffset Property

Indicates the difference between the Minute (see page 7802) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 7802) property represents the minute of the hour using Coordinated Universal Time (see page 4338) (UTC). MinuteOffset is the correction factor you can add to Minute (see page 7802) to obtain the minute of the hour in local time.

See Also

Minute (see page 7802)

89.1.15.3.9 TXSTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the time value. Valid Second values are 0 through 59.

See Also

AsTime (see page 7800)

Millisecond (see page 7802)

Hour (see page 7801)

Minute (see page 7802)

Second

89.1.15.3.10 TXSTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7802) and FractionalSeconds (see page 7801) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS (see page 7797) method does not include the fractional portion of a second in the string encoding of this object's value when the value represents an exact number of seconds. If you set UseZeroMilliseconds, NativeToXS (see page 7797) changes its encoding to include milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full time schema defined by W3C.

See Also

Millisecond (see page 7802)

NativeToXS (see page 7797)

FractionalSeconds (see page 7801)

89.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DateTimeToXMLTime (see page 7805)	Returns the XML string that represents a TDateTime (see page 3335) value.
	DateTimeToXSDateTime (see page 7805)	Returns a TXSDateTime (see page 7772) object that encodes a TDateTime (see page 3335) value.
	GetDataFromFile (see page 7806)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 7806)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 7807)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 7807)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 7807)	Returns a TDateTime (see page 3335) value to represent an XML datetime value.

89.2.1 XSBuiltIns.DateTimeToXMLTime Function

Returns the XML string that represents a TDateTime (see page 3335) value.

Pascal

```
function DateTimeToXMLTime(Value: TDateTime; ApplyLocalBias: Boolean = True): WideString;
```

C++

```
BSTR DateTimeToXMLTime(TDateTime Value, Boolean ApplyLocalBias = True);
```

File

XSBuiltIns

Description

DateTimeToXMLTime returns the XML representation of a specified TDateTime (see page 3335) value.

Value is the TDateTime (see page 3335) value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (see page 4338) (UTC).

CalcLocalBias indicates whether the XML string should include offsets for converting Value to the local time.

See Also

DateTimeToXSDateTime (see page 7805)

XMLTimeToDateTime (see page 7807)

89.2.2 XSBuiltIns.DateTimeToXSDateTime Function

Returns a TXSDateTime (see page 7772) object that encodes a TDateTime (see page 3335) value.

Pascal

```
function DateTimeToXSDateTime(const Value: TDateTime; ApplyLocalBias: Boolean = False):
TXSDateTime;
```

C++

```
TXSDateTime DateTimeToXSDateTime(const TDateTime Value, Boolean ApplyLocalBias = False);
```

File

XSBuiltIns

Description

DateTimeToXSDateTime returns a TXSDateTime (see page 7772) object that corresponds to a specified TDateTime (see page 3335) value. The TXSDateTime (see page 7772) object can be used to represent a date-and-time value on an invokable interface. As a remotable object, it can encoded in SOAP packets.

Value is the TDateTime (see page 3335) value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (see page 4338) (UTC).

CalcLocalBias indicates whether the TXSDateTime (see page 7772) object should include offsets for converting Value to the local time zone. If CalcLocalBias is true, DateTimeToXSDateTime looks up the local time zone and uses it to set the HourOffset and MinuteOffset properties of the return value.

See Also

IInvokable

TXSCustomDateTime (see page 7747)

TXSCustomDateTime (see page 7747)

89.2.3 XSBuiltIns.GetDataFromFile Function

Pascal

```
function GetDataFromFile(AFileName: string): string;
```

C++

```
UnicodeString GetDataFromFile(UnicodeString AFileName);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.GetDataFromFile.

89.2.4 XSBuiltIns.InitXSTypes Function

Pascal

```
procedure InitXSTypes;
```

C++

```
InitXSTypes();
```

File

XSBuiltIns

Description

This is function XSBuiltIns.InitXSTypes.

89.2.5 XSBuiltIns.SoapFloatToStr Function

Pascal

```
function SoapFloatToStr(Value: double): string;
```

C++

```
UnicodeString SoapFloatToStr(double Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapFloatToStr.

89.2.6 XSBuiltIns.SoapStrToFloat Function

Pascal

```
function SoapStrToFloat(Value: string): double;
```

C++

```
double SoapStrToFloat(UnicodeString Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapStrToFloat.

89.2.7 XSBuiltIns.XMLTimeToDateTime Function

Returns a TDateTime (see page 3335) value to represent an XML datetime value.

Pascal

```
function XMLTimeToDateTime(const XMLDateTime: WideString; AsUTCTime: Boolean = False):
TDateTime;
```

C++

```
TDateTime XMLTimeToDateTime(const BSTR XMLDateTime, Boolean AsUTCTime = False);
```

File

XSBuiltIns

Description

XMLTimeToDateTime returns a TDateTime (see page 3335) value for an XML date-and time value. This function is the inverse to the DateTimeToXMLTime (see page 7805) function.

Value is the XML string to convert.

See Also

DateTimeToXSDateTime (see page 7805)

DateTimeToXMLTime (see page 7805)

89.3 Types

The following table lists types in this documentation.

Types

Name	Description
TXSTimeInstant (see page 7809)	This is type XSBuiltIns.TXSTimeInstant.

89.3.1 XSBuiltIns.TXSTimeInstant Type

Pascal

```
TXSTimeInstant = TXSDateTime;
```

C++

```
TXSDateTime TXSTimeInstant;
```

File

XSBuiltIns

Description

This is type XSBuiltIns.TXSTimeInstant.

89.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IncrementAmount (see page 7810)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 7811)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 7811)	This is constant XSBuiltIns.MaxMonth.
MinDay (see page 7811)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 7812)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 7812)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 7812)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 7813)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 7813)	This is constant XSBuiltIns.SNAN.
SNegative (see page 7813)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 7814)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 7814)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 7814)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 7815)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 7815)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 7815)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 7816)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 7816)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 7816)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 7817)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 7817)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 7817)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 7818)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 7818)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 7818)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 7819)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 7819)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 7819)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 7820)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 7820)	This is constant XSBuiltIns.XMLYearPos.

89.4.1 XSBuiltIns.IncrementAmount Constant

Pascal

```
IncrementAmount: array[Boolean] of Integer = (1, -1);
```

C++

```
array[Boolean] of Integer IncrementAmount = (1, -1);
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.IncrementAmount.

89.4.2 XSBuiltIns.MaxDay Constant

Pascal

```
MaxDay = 31;
```

C++

```
MaxDay = 31;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxDay.

89.4.3 XSBuiltIns.MaxMonth Constant

Pascal

```
MaxMonth = 12;
```

C++

```
MaxMonth = 12;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxMonth.

89.4.4 XSBuiltIns.MinDay Constant

Pascal

```
MinDay = 1;
```

C++

```
MinDay = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinDay.

89.4.5 XSBuiltIns.MinMonth Constant

Pascal

```
MinMonth = 1;
```

C++

```
MinMonth = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinMonth.

89.4.6 XSBuiltIns.SDecimal Constant

Pascal

```
SDecimal: Char = '.';
```

C++

```
Char SDecimal = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SDecimal.

89.4.7 XSBuiltIns.SHexMarker Constant

Pascal

```
SHexMarker = '$';
```

C++

```
SHexMarker = '$';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SHexMarker.

89.4.8 XSBuiltIns.SLocalTimeMarker Constant

Pascal

```
SLocalTimeMarker = 'Z';
```

C++

```
SLocalTimeMarker = 'Z';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SLocalTimeMarker.

89.4.9 XSBuiltIns.SNAN Constant

Pascal

```
SNAN = 'NAN';
```

C++

```
SNAN = 'NAN';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNAN.

89.4.10 XSBuiltIns.SNegative Constant

Pascal

```
SNegative: Char = '-';
```

C++

```
Char SNegative = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNegative.

89.4.11 XSBuiltIns.SoapDecimalSeparator Constant

Pascal

```
SoapDecimalSeparator = '.';
```

C++

```
SoapDecimalSeparator = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapDecimalSeparator.

89.4.12 XSBuiltIns.SoapTimePrefix Constant

Pascal

```
SoapTimePrefix: InvString = 'T';
```

C++

```
InvString SoapTimePrefix = 'T';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapTimePrefix.

89.4.13 XSBuiltIns.SPlus Constant

Pascal

```
SPlus: Char = '+';
```

C++

```
Char SPlus = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SPlus.

89.4.14 XSBuiltIns.SSciNotationMarker Constant

Pascal

```
SSciNotationMarker: Char = 'E';
```

C++

```
Char SSciNotationMarker = 'E';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SSciNotationMarker.

89.4.15 XSBuiltIns.XMLDateSeparator Constant

Pascal

```
XMLDateSeparator = '-';
```

C++

```
XMLDateSeparator = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDateSeparator.

89.4.16 XSBuiltIns.XMLDayPos Constant

Pascal

```
XMLDayPos = 9;
```

C++

```
XMLDayPos = 9;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDayPos.

89.4.17 XSBuiltIns.XMLDefaultYearDigits Constant

Pascal

```
XMLDefaultYearDigits = 4;
```

C++

```
XMLDefaultYearDigits = 4;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDefaultYearDigits.

89.4.18 XSBuiltIns.XMLDurationDay Constant

Pascal

```
XMLDurationDay = 'D';
```

C++

```
XMLDurationDay = 'D';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationDay.

89.4.19 XSBuiltIns.XMLDurationHour Constant

Pascal

```
XMLDurationHour = 'H';
```

C++

```
XMLDurationHour = 'H';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationHour.

89.4.20 XSBuiltIns.XMLDurationMinute Constant

Pascal

```
XMLDurationMinute = 'M';
```

C++

```
XMLDurationMinute = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMinute.

89.4.21 XSBuiltIns.XMLDurationMonth Constant

Pascal

```
XMLDurationMonth = 'M';
```

C++

```
XMLDurationMonth = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMonth.

89.4.22 XSBuiltIns.XMLDurationSecond Constant

Pascal

```
XMLDurationSecond = 'S';
```

C++

```
XMLDurationSecond = 'S';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationSecond.

89.4.23 XSBuiltIns.XMLDurationStart Constant

Pascal

```
XMLDurationStart = 'P';
```

C++

```
XMLDurationStart = 'P';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationStart.

89.4.24 XSBuiltIns.XMLDurationYear Constant

Pascal

```
XMLDurationYear = 'Y';
```

C++

```
XMLDurationYear = 'Y';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationYear.

89.4.25 XSBuiltIns.XMLHourOffsetMinusMarker Constant

Pascal

```
XMLHourOffsetMinusMarker = '-';
```

C++

```
XMLHourOffsetMinusMarker = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetMinusMarker.

89.4.26 XSBuiltIns.XMLHourOffsetPlusMarker Constant

Pascal

```
XMLHourOffsetPlusMarker = '+';
```

C++

```
XMLHourOffsetPlusMarker = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetPlusMarker.

89.4.27 XSBuiltIns.XMLMilSecPos Constant

Pascal

```
XMLMilSecPos = 10;
```

C++

```
XMLMilSecPos = 10;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMilSecPos.

89.4.28 XSBuiltIns.XMLMonthPos Constant

Pascal

```
XMLMonthPos = 6;
```

C++

```
XMLMonthPos = 6;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMonthPos.

89.4.29 XSBuiltIns.XMLTimeSeparator Constant

Pascal

```
XMLTimeSeparator = ':';
```

C++

```
XMLTimeSeparator = ':';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLTimeSeparator.

89.4.30 XSBuiltIns.XMLYearPos Constant

Pascal

```
XMLYearPos = 1;
```

C++

```
XMLYearPos = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLYearPos.

90 XSLProd Namespace

This is namespace XSLProd.

Topics

Name	Description
Classes (see page 7822)	The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 7822)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7837) encounters a problem.
	TCustomXSLPageProducer (see page 7822)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 7837)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

90.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 7822)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7837) encounters a problem.
	TCustomXSLPageProducer (see page 7822)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 7837)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

90.1.1 EXSLPageProducerError Class

EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7837) encounters a problem.

Class Hierarchy

Pascal

```
EXSLPageProducerError = class(EWebBrokerException);
```

C++

```
class EXSLPageProducerError : public EWebBrokerException;
```

File

XSLProd

Description

TXSLPageProducer (see page 7837) raises an EXSLPageProducerError exception when it tries to produce content and the WebSnap application has not yet specified the source XML document or the XSL template that transforms that source into the final content.

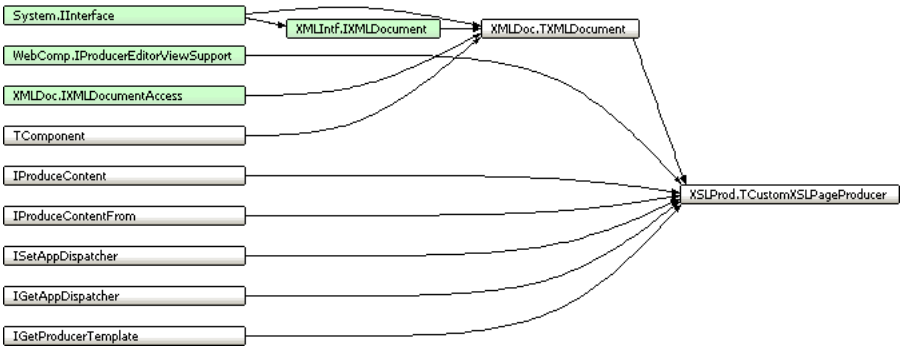
See Also

TXSLPageProducer (see page 7837)

90.1.2 TCustomXSLPageProducer Class

TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy



Pascal

```
TCustomXSLPageProducer = class(TXMLDocument, IProduceContent, IProduceContentFrom,
ISetAppDispatcher, IGetAppDispatcher, IProducerEditorViewSupport, IGetProducerTemplate);
```

C++

```
class TCustomXSLPageProducer : public TXMLDocument, public IProduceContent, public
IProduceContentFrom, public ISetAppDispatcher, public IGetAppDispatcher, public
IProducerEditorViewSupport, public IGetProducerTemplate;
```

File

XSLProd

Description

TCustomXSLPageProducer is the base class for components that transform the data in an XML document using an XSL template. This class introduces all the properties and methods necessary for specifying the XML document to transform, the XSL template that describes the transformation, and for performing the transformation.

The XMLData (see page 7836) property identifies the XML data that this component transforms. XMLData (see page 7836) must be set to a component that supports either the IXMLDocument (see page 7586) interface (for example, TXMLDocument (see page 7299) or one of its descendants) or the IGetXMLStream (see page 6056) interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 7829) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 7829) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 7830) and ContentFromWideString (see page 7830) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 7829) method performs the transformation using an XSL template that it reads from a specified stream.

90.1.2.1 TCustomXSLPageProducer Members

The following tables list the members exposed by TCustomXSLPageProducer.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.

	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7313)	Returns the value of the Active property.
	GetAppDispatcher (see page 7831)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 7831)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7831)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 7831)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7832)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	GetXMLData (see page 7832)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7832)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 7832)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7832)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7833)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7833)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7833)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7320)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
V	Notification (see page 7833)	Forwards notification messages to all owned components.
	ProduceContent (see page 7834)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7834)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 7834)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.

	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 7327)	Sets the value of the Active property.
	SetAppDispatcher (see page 7835)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7307)	Responds after the last constructor has executed.
V	Content (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property.
V	ContentFromStream (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.
V	ContentFromWideString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.
	Create (see page 7303)	This is the overview for the Create constructor overload.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
V	Destroy (see page 7304)	Disposes of a TXMLDocument (see page 7299) object.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.

	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.

	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
R	Dispatcher (see page 7835)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 7836)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.
	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.

	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.
	XMLData (see page 7836)	Specifies the component that supplies the XML data that this component transforms.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.
	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.

90.1.2.2 TCustomXSLPageProducer Methods

The methods of the TCustomXSLPageProducer class are listed here.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 7831)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetProducerTemplateFileName (see page 7831)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7831)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetTemplateFileType (see page 7831)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7832)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetXMLData (see page 7832)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7832)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 7832)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7832)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7833)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7833)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7833)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	Notification (see page 7833)	Forwards notification messages to all owned components.
	ProduceContent (see page 7834)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7834)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.

	ProduceContentFromString (see page 7834)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
	SetAppDispatcher (see page 7835)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

Public Methods

	Name	Description
	Content (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property.
	ContentFromStream (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.
	ContentFromWideString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.

90.1.2.2.1 TCustomXSLPageProducer.Content Method

Returns the transformed value of the XML supplied by the XMLData (see page 7836) property.

Pascal

```
function Content: string; virtual;
```

C++

```
virtual __fastcall UnicodeString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. Content transforms the XML data that is supplied by the XMLData (see page 7836) property using the XSL template associated with this component. The XSL template can be specified in several ways:

If the FileName property is set, it specifies a file that contains the XSL template.

If FileName is not set, you can assign an XSL template using the XML property.

At runtime, you can supply an XSL template by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

See Also

ContentFromStream (see page 7829)

ContentFromString (see page 7830)

ContentFromWideString (see page 7830)

XMLData (see page 7836)

FileName (see page 7339)

XML (see page 7344)

LoadFromFile (see page 7321)

LoadFromStream (see page 7321)

LoadFromXML (see page 7322)

90.1.2.2.2 TCustomXSLPageProducer.ContentFromStream Method

Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using an XSL template that can be read from a specified stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream InStream);
```

Description

Call Content (see page 7829) to retrieve the content for using in an HTTP response message. Content (see page 7829) transforms the XML data that is supplied by the XMLData (see page 7836) property using the XSL template that is reads from InStream.

See Also

Content (see page 7829)

ContentFromString (see page 7830)

ContentFromWideString (see page 7830)

XMLData (see page 7836)

90.1.2.2.3 TCustomXSLPageProducer.ContentFromString Method

Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.

Pascal

```
function ContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ContentFromString(const UnicodeString S);
```

Description

Call Content (see page 7829) to retrieve the content for using in an HTTP response message. Content (see page 7829) transforms the XML data that is supplied by the XMLData (see page 7836) property using the XSL template specified by S.

See Also

Content (see page 7829)

ContentFromStream (see page 7829)

ContentFromWideString (see page 7830)

XMLData (see page 7836)

90.1.2.2.4 TCustomXSLPageProducer.ContentFromWideString Method

Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.

Pascal

```
function ContentFromWideString(const S: WideString): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromWideString(const BSTR S);
```

Description

Call Content (see page 7829) to retrieve the content for using in an HTTP response message. Content (see page 7829) transforms the XML data that is supplied by the XMLData (see page 7836) property using the XSL template specified by S.

See Also

Content (see page 7829)

ContentFromStream (see page 7829)

ContentFromString (see page 7830)

XMLData (see page 7836)

90.1.2.2.5 TCustomXSLPageProducer.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class TCustomXSLPageProducer.

90.1.2.2.6 TCustomXSLPageProducer.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall UnicodeString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.

90.1.2.2.7 TCustomXSLPageProducer.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.

90.1.2.2.8 TCustomXSLPageProducer.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TCustomXSLPageProducer.

90.1.2.2.9 TCustomXSLPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomXSLPageProducer.

90.1.2.2.10 TCustomXSLPageProducer.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TCustomXSLPageProducer.

90.1.2.2.11 TCustomXSLPageProducer.GetXMLDocument Method

Pascal

```
function GetXMLDocument(out AXMLDocument: IXMLDocument): Boolean;
```

C++

```
__fastcall Boolean GetXMLDocument(IXMLDocument AXMLDocument);
```

Description

This is GetXMLDocument, a member of class TCustomXSLPageProducer.

90.1.2.2.12 TCustomXSLPageProducer.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TCustomXSLPageProducer.

90.1.2.2.13 TCustomXSLPageProducer.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.

90.1.2.2.14 TCustomXSLPageProducer.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TCustomXSLPageProducer.

90.1.2.2.15 TCustomXSLPageProducer.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.

90.1.2.2.16 TCustomXSLPageProducer.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.

90.1.2.2.17 TCustomXSLPageProducer.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component

has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

90.1.2.2.18 TCustomXSLPageProducer.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall UnicodeString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content

90.1.2.2.19 TCustomXSLPageProducer.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.

90.1.2.2.20 TCustomXSLPageProducer.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromString(const UnicodeString S);
```

Description

This is ProduceContentFromString, a member of class TCustomXSLPageProducer.

90.1.2.2.21 TCustomXSLPageProducer.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

90.1.2.3 TCustomXSLPageProducer Properties

The properties of the TCustomXSLPageProducer class are listed here.

Public Properties

	Name	Description
	Dispatcher (see page 7835)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DispatcherComponent (see page 7836)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	XMLData (see page 7836)	Specifies the component that supplies the XML data that this component transforms.

90.1.2.3.1 TCustomXSLPageProducer.Dispatcher Property

Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property Dispatcher: IWebDispatcherAccess;
```

C++

```
__property IWebDispatcherAccess Dispatcher;
```

Description

Read (see page 3727) Dispatcher to access the interface of the dispatcher. In Web Broker applications, this is the interface of a TWebDispatcher or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

The IWebDispatcherAccess interface has two methods:

Request, which returns the HTTP request message to which this page producer responds.

Response, which returns the HTTP response message for which this page producer generates content.

See Also

TWebModule

TWebDispatcher

TWebAppComponents (see page 6217)

TWebRequest

TWebResponse

DispatcherComponent (see page 7836)

90.1.2.3.2 TCustomXSLPageProducer.DispatcherComponent Property

Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property DispatcherComponent: TComponent;
```

C++

```
__property TComponent * DispatcherComponent;
```

Description

Read (see page 3727) DispatcherComponent to access the dispatcher that handles the HTTP request and response messages for this Web Server application. In Web Broker applications, this is the interface of a TWebDispatcherComponent or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

Dispatcher (see page 7835) implements the interface that is the value of the Dispatcher (see page 7835) property. The XSL page producer uses this property so that it can respond to notifications when the component is freed. To use the dispatcher, use the Dispatcher (see page 7835) property instead.

See Also

TWebModule

TWebDispatcher

TWebAppComponents (see page 6217)

TWebRequest

TWebResponse

Dispatcher (see page 7835)

90.1.2.3.3 TCustomXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument (see page 7586) interface (for example, TXMLDocument (see page 7299) or one of its descendants) or the IGetXMLStream (see page 6056) interface (for example, TXMLBroker).

See Also

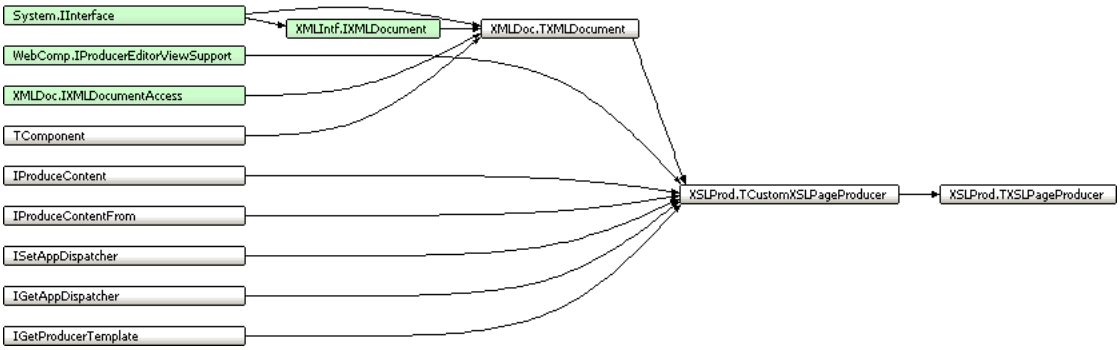
TXMLDocument (see page 7299)

TXMLBroker (see page 7255)

90.1.3 TXSLPageProducer Class

TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy



Pascal

```
TXSLPageProducer = class(TCustomXSLPageProducer);
```

C++

```
class TXSLPageProducer : public TCustomXSLPageProducer;
```

File

XSLProd

Description

Use the Web Page Module wizard to create a web page that uses TXSLPageProducer. To produce content, TXSLPageProducer uses an XSL template to transform the data in an XML document. Typically, the XSL transforms the XML document into another XML document that is HTML compliant (that is, into a document whose tags are actually HTML tags). However, as specified by the XSL template, TXSLPageProducer can return any arbitrary output as a result of the transformation.

Use the XMLData (see page 7843) property to specify the XML data that this component transforms. XMLData (see page 7843) must be set to a component that supports either the IXMLDocument (see page 7586) interface (for example, TXMLDocument (see page 7299) or one of its descendants) or the IGetXMLStream (see page 6056) interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 7829) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 7829) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 7830) and ContentFromWideString (see page 7830) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 7829) method performs the transformation using an XSL template that it reads from a specified stream.

At design time, Web page modules that use the TXMLPageProducer have editor views called XML Tree and XSL Tree. These views provide a read only tree of the XML and XSL respectively.

See Also

TXMLDocument (see page 7299)

TXMLBroker (see page 7255)

90.1.3.1 TXSLPageProducer Members

The following tables list the members exposed by TXSLPageProducer.

Protected Methods

	Name	Description
	_AddRef (see page 7306)	Called when an application uses a component interface.
	_Release (see page 7306)	Called when an application releases a component interface.
	AssignParseOptions (see page 7308)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7308)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7308)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7309)	This is CheckDOM, a member of class TXMLDocument.
	DefineProperties (see page 7311)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7311)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7311)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7312)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7312)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7312)	This is DoNodeChange, a member of class TXMLDocument.
	GetActive (see page 7313)	Returns the value of the Active property.
	GetAppDispatcher (see page 7831)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 7313)	Returns the value of the AsyncLoadState property.
	GetChildNodeClass (see page 7313)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7313)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7314)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7315)	Returns the value of the Node property.
	GetDocumentObject (see page 7315)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7315)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7315)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7316)	Returns the value of the Encoding property.
	GetFileName (see page 7316)	Returns the value of the FileName property.
	GetModified (see page 7316)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7317)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7317)	Returns the value of the Options property.
	GetParseOptions (see page 7317)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 7831)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7831)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetPrologNode (see page 7317)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7318)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7318)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7318)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 7831)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7832)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetVersion (see page 7318)	Returns the value of the Version property.
	GetXML (see page 7319)	Returns the value of the XML property.
	GetXMLData (see page 7832)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7832)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.

	GetXSLData (see page 7832)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7832)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7833)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7833)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7833)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 7319)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7320)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7320)	Initializes the component after the form file has been read into memory.
V	Notification (see page 7833)	Forwards notification messages to all owned components.
	ProduceContent (see page 7834)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7834)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 7834)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
	ReadDOMVendor (see page 7323)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7324)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7326)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7327)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 7327)	Sets the value of the Active property.
	SetAppDispatcher (see page 7835)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetDocumentElement (see page 7327)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7328)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7328)	Sets the value of the Encoding property.
	SetFileName (see page 7328)	Sets the value of the FileName property.
	SetModified (see page 7329)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7329)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7329)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7330)	Sets the value of the Options property.
	SetParseOptions (see page 7330)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7330)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7330)	Sets the value of the StandAlone property.
	SetVersion (see page 7331)	Sets the value of the Version property.
	SetXML (see page 7331)	Sets the value of the XML property.
	SetXMLStrings (see page 7331)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7332)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7332)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7335)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7335)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7337)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7337)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7342)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3504)	Increments the reference count for this interface.
	_Release (see page 3504)	Decrements the reference count for this interface.
	AddChild (see page 7307)	This is the overview for the AddChild method overload.
	AddChild (see page 7589)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7307)	Responds after the last constructor has executed.
V	Content (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property.

	ContentFromStream (see page 7829)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.
	ContentFromWideString (see page 7830)	Returns the transformed value of the XML supplied by the XMLData (see page 7836) property using a specified XSL template.
	Create (see page 7303)	This is the overview for the Create constructor overload.
	CreateElement (see page 7309)	Creates a new element node that is associated with this document.
	CreateElement (see page 7590)	Creates a new element node that is associated with this document.
	CreateNode (see page 7309)	Creates a new node that is associated with this document.
	CreateNode (see page 7591)	Creates a new node that is associated with this document.
	Destroy (see page 7304)	Disposes of a TXMLDocument (see page 7299) object.
	GeneratePrefix (see page 7312)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7592)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7592)	Returns the value of the Active (see page 7607) property.
	GetAsyncLoadState (see page 7593)	Returns the value of the AsyncLoadState (see page 7608) property.
	GetChildNodes (see page 7593)	Returns the value of the ChildNodes (see page 7609) property.
	GetDocBinding (see page 7314)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7593)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7594)	Returns the value of the DocumentElement (see page 7609) property.
	GetDocumentNode (see page 7594)	Returns the value of the Node (see page 7611) property.
	GetDocumentObject (see page 7430)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7594)	Returns the value of the DOMDocument (see page 7609) property.
	GetDOMPersist (see page 7430)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7595)	Returns the value of the Encoding (see page 7610) property.
	GetFileName (see page 7595)	Returns the value of the FileName (see page 7610) property.
	GetModified (see page 7595)	Returns the value of the Modified (see page 7611) property.
	GetNodeIndentStr (see page 7596)	Returns the value of the NodeIndentStr (see page 7612) property.
	GetOptions (see page 7596)	Returns the value of the Options (see page 7612) property.
	GetParseOptions (see page 7596)	Returns the value of the ParseOptions (see page 7613) property.
	GetSchemaRef (see page 7596)	Returns the value of the SchemaRef (see page 7613) property.
	GetStandAlone (see page 7597)	Returns the value of the StandAlone (see page 7614) property.
	GetTemplateFileType (see page 6063)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVersion (see page 7597)	Returns the value of the Version (see page 7614) property.
	GetXML (see page 7597)	Returns the value of the XML (see page 7614) property.
	GetXMLData (see page 6063)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6063)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6063)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6063)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6064)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6064)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IsEmptyDoc (see page 7319)	Indicates whether the TXMLDocument (see page 7299) instance represents an empty document.
	IsEmptyDoc (see page 7598)	Indicates whether the IXMLDocument (see page 7586) instance represents an empty document.
	LoadFromFile (see page 7321)	Loads an XML (see page 7344) document and activates it.
	LoadFromFile (see page 7598)	Loads an XML (see page 7614) document from disk.
	LoadFromStream (see page 7321)	Loads an XML (see page 7344) document from a stream and activates it.
	LoadFromStream (see page 7599)	Loads an XML (see page 7614) document from a stream and activates it.
	LoadFromXML (see page 7322)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7599)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7322)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 3505)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7323)	Updates the parsed XML (see page 7344) document to reflect external changes.
	Refresh (see page 7600)	Updates the parsed XML (see page 7614) document to reflect external changes.
	RegisterDocBinding (see page 7324)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7600)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7325)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7601)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7325)	Saves the XML (see page 7344) document to disk.
	SaveToFile (see page 7601)	Saves the XML (see page 7614) document to disk.
	SaveToStream (see page 7325)	Saves the XML (see page 7344) document to a stream.
	SaveToStream (see page 7602)	Saves the XML (see page 7614) document to a stream.
	SaveToXML (see page 7326)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7602)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7603)	Sets the value of the Active (see page 7607) property.
	SetDocumentElement (see page 7603)	Sets the value of the DocumentElement (see page 7609) property.
	SetDOMDocument (see page 7604)	Sets the value of the DOMDocument (see page 7609) property.
	SetEncoding (see page 7604)	Sets the value of the Encoding (see page 7610) property.
	SetFileName (see page 7604)	Sets the value of the FileName (see page 7610) property.
	SetNodeIndentStr (see page 7605)	Sets the value of the NodeIndentStr (see page 7612) property.
	SetOnAsyncLoad (see page 7605)	Supplies an event handler that responds when the AsyncLoadState (see page 7608) property changes.
	SetOptions (see page 7605)	Sets the value of the Options (see page 7612) property.
	SetParseOptions (see page 7606)	Sets the value of the ParseOptions (see page 7613) property.
	SetStandAlone (see page 7606)	Sets the value of the StandAlone (see page 7614) property.
	SetVersion (see page 7606)	Sets the value of the Version (see page 7614) property.
	SetXML (see page 7607)	Sets the value of the XML (see page 7614) property.

Public Properties

	Name	Description
	Active (see page 7607)	Specifies whether the XML (see page 7614) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7334)	Indicates the state of the DOM parser when it parses the XML (see page 7344) document asynchronously.
R	AsyncLoadState (see page 7608)	Indicates the state of the DOM parser when it parses the XML (see page 7614) document asynchronously.
R	ChildNodes (see page 7335)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7609)	Lists all of the child nodes of the document.
R	Dispatcher (see page 7835)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 7836)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 7335)	Provides access to the root node of the XML (see page 7344) document.
	DocumentElement (see page 7609)	Provides access to the root node of the XML (see page 7614) document.
R	DocumentObject (see page 7430)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7336)	Provides access to the DOM interface for the parsed XML (see page 7344) document.
	DOMDocument (see page 7609)	Provides access to the DOM interface for the parsed XML (see page 7614) document.
	DOMImplementation (see page 7337)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 7431)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7338)	Indicates the character set that the XML (see page 7344) document uses to encode characters.

	Encoding (see page 7610)	Indicates the character set that the XML (see page 7614) document uses to encode characters.
	FileName (see page 7610)	Indicates the XML (see page 7614) document file, if any, that this interface represents.
R	Modified (see page 7339)	Indicates whether the XML (see page 7344) document has been modified.
R	Modified (see page 7611)	Indicates whether the XML (see page 7614) document has been modified.
R	Node (see page 7340)	Provides access to the document node for the XML (see page 7344) document.
R	Node (see page 7611)	Provides access to the document node for the XML (see page 7614) document.
	NodeIndentStr (see page 7612)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7614) text.
	NSPrefixBase (see page 7341)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7612)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7613)	Specifies implementation-specific options that are followed when parsing the XML (see page 7614) document.
R	SchemaRef (see page 7343)	Returns the name of the schema associated with this XML (see page 7344) document.
R	SchemaRef (see page 7613)	Returns the name of the schema associated with this XML (see page 7614) document.
	StandAlone (see page 7343)	Indicates whether the XML (see page 7344) document includes any external markup declarations.
	StandAlone (see page 7614)	Indicates whether the XML (see page 7614) document includes any external markup declarations.
	Version (see page 7343)	Indicates the XML (see page 7344) version that the document uses.
	Version (see page 7614)	Indicates the XML (see page 7614) version that the document uses.
	XML (see page 7614)	Specifies the content of the XML document associated with this IXMLDocument (see page 7586) interface.

Published Events

	Name	Description
	AfterClose (see page 7344)	Occurs after the XML (see page 7344) document is closed.
	AfterNodeChange (see page 7345)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7345)	Occurs immediately after the XML (see page 7344) document is opened.
	BeforeClose (see page 7346)	Occurs immediately before the XML (see page 7344) document is closed.
	BeforeNodeChange (see page 7346)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7347)	Occurs immediately before the XML (see page 7344) document is opened.
	OnAsyncLoad (see page 7347)	Occurs at various stages during the asynchronous parsing of an XML (see page 7344) document.

Published Properties

	Name	Description
	Active (see page 7333)	Specifies whether the XML (see page 7344) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7337)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7344) document.
	FileName (see page 7339)	Specifies the file name for the XML (see page 7344) document associated with this TXMLDocument (see page 7299) object.
	NodeIndentStr (see page 7340)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7344) text.
	Options (see page 7341)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7342)	Specifies implementation-specific options that are followed when parsing the XML (see page 7344) document.
	XML (see page 7344)	Specifies the content of the XML document associated with this TXMLDocument (see page 7299) object.
	XMLData (see page 7843)	Specifies the component that supplies the XML data that this component transforms.

90.1.3.2 TXSLPageProducer Properties

The properties of the TXSLPageProducer class are listed here.

Published Properties

	Name	Description
	XMLData (see page 7843)	Specifies the component that supplies the XML data that this component transforms.

90.1.3.2.1 TXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

See Also

TXMLDocument (see page 7299)

TXMLBroker (see page 7255)

91 ZLib Namespace

This is namespace ZLib.

Topics

Name	Description
Classes (see page 7848)	The following table lists classes in this documentation.
Functions (see page 7859)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7868)	The following table lists structs, records, enums in this documentation.
Types (see page 7871)	The following table lists types in this documentation.
Constants (see page 7874)	The following table lists constants in this documentation.

Classes

	Name	Description
	EZCompressionError (see page 7848)	This is class ZLib.EZCompressionError.
	EZDecompressionError (see page 7848)	This is class ZLib.EZDecompressionError.
	EZLibError (see page 7849)	This is class ZLib.EZLibError.
	TCustomZStream (see page 7849)	This is class ZLib.TCustomZStream.
	TZCompressionStream (see page 7851)	This is class ZLib.TZCompressionStream.
	TZDecompressionStream (see page 7855)	This is class ZLib.TZDecompressionStream.

Constants

Name	Description
_z_errmsg (see page 7874)	This is constant ZLib._z_errmsg.
SZInvalid (see page 7875)	This is constant ZLib.SZInvalid.
Z_ASCII (see page 7875)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 7875)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 7876)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 7876)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 7876)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 7877)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 7877)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 7877)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 7878)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 7878)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 7878)	This is constant ZLib.Z_FILTERED.
Z_FINISH (see page 7879)	This is constant ZLib.Z_FINISH.

Z_FULL_FLUSH (see page 7879)	This is constant ZLib.Z_FULL_FLUSH.
Z_HUFFMAN_ONLY (see page 7879)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 7880)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 7880)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 7880)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 7881)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 7881)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 7881)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 7882)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 7882)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 7882)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 7883)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 7883)	This is constant ZLib.Z_VERSION_ERROR.
ZLevels (see page 7883)	This is constant ZLib.ZLevels.
ZLIB_VERSION (see page 7884)	This is constant ZLib.ZLIB_VERSION.

Functions

	Name	Description
	adler32 (see page 7859)	This is function ZLib.adler32.
	deflate (see page 7860)	This is function ZLib.deflate.
	deflateEnd (see page 7860)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 7860)	This is function ZLib.deflateInit_.
	DeflateInit2_ (see page 7861)	This is function ZLib.DeflateInit2_.
	inflate (see page 7861)	This is function ZLib.inflate.
	inflateEnd (see page 7861)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 7862)	This is function ZLib.inflateInit_.
	inflateInit2_ (see page 7862)	This is function ZLib.inflateInit2_.
	inflateReset (see page 7862)	This is function ZLib.inflateReset.
	MoveI32 (see page 7863)	This is function ZLib.MoveI32.
	ZCompress (see page 7863)	This is function ZLib.ZCompress.
	ZCompress (see page 7863)	This is function ZLib.ZCompress.
	ZCompressStr (see page 7864)	This is function ZLib.ZCompressStr.
	ZCompressStr (see page 7864)	This is function ZLib.ZCompressStr.
	ZCompressStream (see page 7864)	This is function ZLib.ZCompressStream.
	ZCompressStream (see page 7865)	This is function ZLib.ZCompressStream.
	ZDecompress (see page 7865)	This is function ZLib.ZDecompress.
	ZDecompressStr (see page 7865)	This is function ZLib.ZDecompressStr.
	ZDecompressStream (see page 7866)	This is function ZLib.ZDecompressStream.
	zlibAllocMem (see page 7866)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 7866)	This is function ZLib.zlibFreeMem.
	ZSendToBrowser (see page 7867)	This is function ZLib.ZSendToBrowser.

Structs, Records, Enums

	Name	Description
	TCompressionLevel (see page 7868)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.
	TZCompressionLevel (see page 7869)	This is record ZLib.TZCompressionLevel.
	TZStreamRec (see page 7869)	This is record ZLib.TZStreamRec.

Types

Name	Description
TCompressionStream (see page 7871)	Compresses data as it is written to the stream.
TCustomZLibStream (see page 7872)	This is type ZLib.TCustomZLibStream.
TDecompressionStream (see page 7872)	Decompresses data that was written to a TCompressionStream (see page 7871) object.

TZAlloc (see page 7872)	This is type ZLib.TZAlloc.
TZFree (see page 7873)	This is type ZLib.TZFree.

91.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EZCompressionError (see page 7848)	This is class ZLib.EZCompressionError.
	EZDecompressionError (see page 7848)	This is class ZLib.EZDecompressionError.
	EZLibError (see page 7849)	This is class ZLib.EZLibError.
	TCustomZStream (see page 7849)	This is class ZLib.TCustomZStream.
	TZCompressionStream (see page 7851)	This is class ZLib.TZCompressionStream.
	TZDecompressionStream (see page 7855)	This is class ZLib.TZDecompressionStream.

91.1.1 EZCompressionError Class

Class Hierarchy

Pascal

```
EZCompressionError = class(EZLibError);
```

C++

```
class EZCompressionError : public EZLibError;
```

File

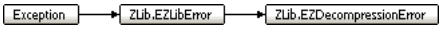
ZLib

Description

This is class ZLib.EZCompressionError.

91.1.2 EZDecompressionError Class

Class Hierarchy



Pascal

```
EZDecompressionError = class(EZLibError);
```

C++

```
class EZDecompressionError : public EZLibError;
```

File

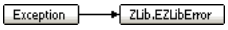
ZLib

Description

This is class ZLib.EZDecompressionError.

91.1.3 EZLibError Class

Class Hierarchy



Pascal

```
EZLibError = class(Exception);
```

C++

```
class EZLibError : public Exception;
```

File

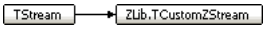
ZLib

Description

This is class ZLib.EZLibError.

91.1.4 TCustomZStream Class

Class Hierarchy



Pascal

```
TCustomZStream = class(TStream);
```

C++

```
class TCustomZStream : public TStream;
```

File

ZLib

Description

This is class ZLib.TCustomZStream.

91.1.4.1 TCustomZStream Members

The following tables list the members exposed by TCustomZStream.

Protected Events

	Name	Description
	OnProgress (see page 7850)	This is OnProgress, a member of class TCustomZStream.

Protected Methods

	Name	Description
	Create (see page 7850)	This is Create, a member of class TCustomZStream.
	DoProgress (see page 7850)	This is DoProgress, a member of class TCustomZStream.

91.1.4.2 TCustomZStream.Create Constructor

Pascal

```
constructor Create(stream: TStream);
```

C++

```
__fastcall TCustomZStream(TStream stream);
```

Description

This is Create, a member of class TCustomZStream.

91.1.4.3 TCustomZStream Methods

The methods of the TCustomZStream class are listed here.

Protected Methods

	Name	Description
	DoProgress (see page 7850)	This is DoProgress, a member of class TCustomZStream.

91.1.4.3.1 TCustomZStream.DoProgress Method

Pascal

```
procedure DoProgress; dynamic;
```

C++

```
__fastcall DoProgress();
```

Description

This is DoProgress, a member of class TCustomZStream.

91.1.4.4 TCustomZStream Events

The events of the TCustomZStream class are listed here.

Protected Events

	Name	Description
	OnProgress (see page 7850)	This is OnProgress, a member of class TCustomZStream.

91.1.4.4.1 TCustomZStream.OnProgress Event

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

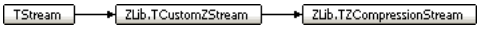
```
__property TNotifyEvent OnProgress;
```

Description

This is OnProgress, a member of class TCustomZStream.

91.1.5 TZCompressionStream Class

Class Hierarchy



Pascal

```
TZCompressionStream = class(TCustomZStream);
```

C++

```
class TZCompressionStream : public TCustomZStream;
```

File

ZLib

Description

This is class ZLib.TZCompressionStream.

91.1.5.1 TZCompressionStream Members

The following tables list the members exposed by TZCompressionStream.

Protected Methods

	Name	Description
	DoProgress (see page 7850)	This is DoProgress, a member of class TCustomZStream.

Public Events

	Name	Description
	OnProgress (see page 7855)	This is OnProgress, a member of class TZCompressionStream.

Public Methods

	Name	Description
	Create (see page 7852)	This is the overview for the Create constructor overload.
V	Destroy (see page 7852)	Disposes of an object instance.
V	Read (see page 7853)	Introduces an abstract or pure virtual method responsible for reading from the stream.
V	Seek (see page 7854)	This is Seek, a member of class TZCompressionStream.
V	Write (see page 7854)	Introduces a pure virtual method for writing to the stream.

Public Properties

	Name	Description
R	CompressionRate (see page 7854)	This is CompressionRate, a member of class TZCompressionStream.

91.1.5.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TZCompressionStream.Create (TCompressionLevel, TStream) (see page 7852)	This is Create, a member of class TZCompressionStream.
	TZCompressionStream.Create (TStream, TZCompressionLevel) (see page 7852)	This is Create, a member of class TZCompressionStream.

91.1.5.2.1 TZCompressionStream.Create Constructor (TCompressionLevel, TStream)

Pascal

```
constructor Create(compressionLevel: TCompressionLevel; dest: TStream); overload;
```

C++

```
__fastcall TZCompressionStream(TCompressionLevel compressionLevel, TStream dest);
```

Description

This is Create, a member of class TZCompressionStream.

91.1.5.2.2 TZCompressionStream.Create Constructor (TStream, TZCompressionLevel)

Pascal

```
constructor Create(dest: TStream; compressionLevel: TZCompressionLevel = zcDefault); overload;
```

C++

```
__fastcall TZCompressionStream(TStream dest, TZCompressionLevel compressionLevel = zcDefault);
```

Description

This is Create, a member of class TZCompressionStream.

91.1.5.3 TZCompressionStream.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TZCompressionStream();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil

before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

91.1.5.4 TZCompressionStream Methods

The methods of the TZCompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 7853)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	Seek (see page 7854)	This is Seek, a member of class TZCompressionStream.
	Write (see page 7854)	Introduces a pure virtual method for writing to the stream.

91.1.5.4.1 TZCompressionStream.Read Method

Introduces an abstract or pure virtual method responsible for reading from the stream.

Pascal

```
function Read(var buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( buffer, Longint count);
```

Description

Each descendant stream object defines a Read method that reads data from its particular storage medium (such as memory or a disk file).

Read is used in cases where the number of bytes to read from the stream is not necessarily fixed. It attempts to read up to Count bytes into buffer and returns the number of bytes actually read.

All the other data-reading methods of a stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

See Also

EReadError

ReadBuffer

ReadComponent

91.1.5.4.2 TZCompressionStream.Seek Method

Pascal

```
function Seek(offset: Longint; origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint offset, Word origin);
```

Description

This is Seek, a member of class TZCompressionStream.

91.1.5.4.3 TZCompressionStream.Write Method

Introduces a pure virtual method for writing to the stream.

Pascal

```
function Write(const buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  buffer, Longint count);
```

Description

Each descendant stream object defines a Write method that writes data to its particular storage medium (such as memory or a disk file). Write attempts to write up to Count bytes from Buffer, and returns the number of bytes actually written.

All the other data-writing methods of a stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

See Also

WriteBuffer

WriteComponent

91.1.5.5 TZCompressionStream Properties

The properties of the TZCompressionStream class are listed here.

Public Properties

	Name	Description
	CompressionRate (see page 7854)	This is CompressionRate, a member of class TZCompressionStream.

91.1.5.5.1 TZCompressionStream.CompressionRate Property

Pascal

```
property CompressionRate: Single;
```

C++

```
__property Single CompressionRate;
```

Description

This is CompressionRate, a member of class TZCompressionStream.

91.1.5.6 TZCompressionStream Events

The events of the TZCompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 7855)	This is OnProgress, a member of class TZCompressionStream.

91.1.5.6.1 TZCompressionStream.OnProgress Event

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

This is OnProgress, a member of class TZCompressionStream.

91.1.6 TZDecompressionStream Class

Class Hierarchy

Pascal

```
TZDecompressionStream = class(TCustomZStream);
```

C++

```
class TZDecompressionStream : public TCustomZStream;
```

File

ZLib

Description

This is class ZLib.TZDecompressionStream.

91.1.6.1 TZDecompressionStream Members

The following tables list the members exposed by TZDecompressionStream.

Protected Methods

	Name	Description
	DoProgress (see page 7850)	This is DoProgress, a member of class TCustomZStream.

Public Events

	Name	Description
	OnProgress (see page 7858)	This is OnProgress, a member of class TZDecompressionStream.

Public Methods

	Name	Description
	Create (see page 7856)	This is Create, a member of class TZDecompressionStream.
	Destroy (see page 7856)	Disposes of an object instance.
	Read (see page 7857)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	Seek (see page 7857)	This is Seek, a member of class TZDecompressionStream.
	Write (see page 7857)	Introduces a pure virtual method for writing to the stream.

91.1.6.2 TZDecompressionStream.Create Constructor

Pascal

```
constructor Create(source: TStream);
```

C++

```
__fastcall TZDecompressionStream(TStream source);
```

Description

This is Create, a member of class TZDecompressionStream.

91.1.6.3 TZDecompressionStream.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TZDecompressionStream();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3382)

91.1.6.4 TZDecompressionStream Methods

The methods of the TZDecompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 7857)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	Seek (see page 7857)	This is Seek, a member of class TZDecompressionStream.
	Write (see page 7857)	Introduces a pure virtual method for writing to the stream.

91.1.6.4.1 TZDecompressionStream.Read Method

Introduces an abstract or pure virtual method responsible for reading from the stream.

Pascal

```
function Read(var buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( buffer, Longint count);
```

Description

Each descendant stream object defines a Read method that reads data from its particular storage medium (such as memory or a disk file).

Read is used in cases where the number of bytes to read from the stream is not necessarily fixed. It attempts to read up to Count bytes into buffer and returns the number of bytes actually read.

All the other data-reading methods of a stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

See Also

EReadError

ReadBuffer

ReadComponent

91.1.6.4.2 TZDecompressionStream.Seek Method

Pascal

```
function Seek(offset: Longint; origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint offset, Word origin);
```

Description

This is Seek, a member of class TZDecompressionStream.

91.1.6.4.3 TZDecompressionStream.Write Method

Introduces a pure virtual method for writing to the stream.

Pascal

```
function Write(const buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  buffer, Longint count);
```

Description

Each descendant stream object defines a Write method that writes data to its particular storage medium (such as memory or a disk file). Write attempts to write up to Count bytes from Buffer, and returns the number of bytes actually written.

All the other data-writing methods of a stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

See Also

WriteBuffer

WriteComponent

91.1.6.5 TZDecompressionStream Events

The events of the TZDecompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 7858)	This is OnProgress, a member of class TZDecompressionStream.

91.1.6.5.1 TZDecompressionStream.OnProgress Event

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

This is OnProgress, a member of class TZDecompressionStream.

91.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	adler32 (see page 7859)	This is function ZLib.adler32.
	deflate (see page 7860)	This is function ZLib.deflate.
	deflateEnd (see page 7860)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 7860)	This is function ZLib.deflateInit_.
	DeflateInit2_ (see page 7861)	This is function ZLib.DeflateInit2_.
	inflate (see page 7861)	This is function ZLib.inflate.
	inflateEnd (see page 7861)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 7862)	This is function ZLib.inflateInit_.
	inflateInit2_ (see page 7862)	This is function ZLib.inflateInit2_.
	inflateReset (see page 7862)	This is function ZLib.inflateReset.
	MoveI32 (see page 7863)	This is function ZLib.MoveI32.
	ZCompress (see page 7863)	This is function ZLib.ZCompress.
	ZCompress (see page 7863)	This is function ZLib.ZCompress.
	ZCompressStr (see page 7864)	This is function ZLib.ZCompressStr.
	ZCompressStr (see page 7864)	This is function ZLib.ZCompressStr.
	ZCompressStream (see page 7864)	This is function ZLib.ZCompressStream.
	ZCompressStream (see page 7865)	This is function ZLib.ZCompressStream.
	ZDecompress (see page 7865)	This is function ZLib.ZDecompress.
	ZDecompressStr (see page 7865)	This is function ZLib.ZDecompressStr.
	ZDecompressStream (see page 7866)	This is function ZLib.ZDecompressStream.
	zlibAllocMem (see page 7866)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 7866)	This is function ZLib.zlibFreeMem.
	ZSendToBrowser (see page 7867)	This is function ZLib.ZSendToBrowser.

91.2.1 ZLib.adler32 Function

Pascal

```
function adler32(adler: LongInt; const buf: PChar; len: Integer): LongInt;
```

C++

```
LongInt adler32(LongInt adler, const const char * buf, int len);
```

File

ZLib

Description

This is function ZLib.adler32.

91.2.2 ZLib.deflate Function

Pascal

```
function deflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int deflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.deflate.

91.2.3 ZLib.deflateEnd Function

Pascal

```
function deflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int deflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.deflateEnd.

91.2.4 ZLib.deflateInit_ Function

Pascal

```
function deflateInit_(var strm: TZStreamRec; level: Integer; version: PChar; recsize:
Integer): Integer;
```

C++

```
int deflateInit_(TZStreamRec strm, int level, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.deflateInit_.

91.2.5 ZLib.DeflateInit2_ Function

Pascal

```
function DeflateInit2_(var strm: TZStreamRec; level: integer; method: integer; windowBits:
integer; memLevel: integer; strategy: integer; version: PChar; recsize: integer): integer;
```

C++

```
int DeflateInit2_(TZStreamRec strm, int level, int method, int windowBits, int memLevel, int
strategy, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.DeflateInit2_.

91.2.6 ZLib.inflate Function

Pascal

```
function inflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int inflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.inflate.

91.2.7 ZLib.inflateEnd Function

Pascal

```
function inflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int inflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateEnd.

91.2.8 ZLib.inflateInit_ Function

Pascal

```
function inflateInit_(var strm: TZStreamRec; version: PChar; recsize: Integer): Integer;
```

C++

```
int inflateInit_(TZStreamRec strm, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.inflateInit_.

91.2.9 ZLib.inflateInit2_ Function

Pascal

```
function inflateInit2_(var strm: TZStreamRec; windowBits: integer; version: PChar; recsize:
integer): integer;
```

C++

```
int inflateInit2_(TZStreamRec strm, int windowBits, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.inflateInit2_.

91.2.10 ZLib.inflateReset Function

Pascal

```
function inflateReset(var strm: TZStreamRec): Integer;
```

C++

```
int inflateReset(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateReset.

91.2.11 ZLib.MoveI32 Function

Pascal

```
procedure MoveI32(const Source; var Dest; Count: Integer);
```

C++

```
MoveI32(const  Source,  Dest, int Count);
```

File

ZLib

Description

This is function ZLib.MoveI32.

91.2.12 ZLib.ZCompress Function

Pascal

```
procedure ZCompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; level: TCompressionLevel); overload;
```

C++

```
ZCompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, TCompressionLevel
level);
```

File

ZLib

Description

This is function ZLib.ZCompress.

91.2.13 ZLib.ZCompress Function

Pascal

```
procedure ZCompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; level: TZCompressionLevel = zcDefault); overload;
```

C++

```
ZCompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, TZCompressionLevel
level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompress.

91.2.14 ZLib.ZCompressStr Function

Pascal

```
function ZCompressStr(const s: string; level: TCompressionLevel): TBytes; overload;
```

C++

```
TBytes ZCompressStr(const UnicodeString s, TCompressionLevel level);
```

File

ZLib

Description

This is function ZLib.ZCompressStr.

91.2.15 ZLib.ZCompressStr Function

Pascal

```
function ZCompressStr(const s: string; level: TZCompressionLevel = zcDefault): TBytes;
overload;
```

C++

```
TBytes ZCompressStr(const UnicodeString s, TZCompressionLevel level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompressStr.

91.2.16 ZLib.ZCompressStream Function

Pascal

```
procedure ZCompressStream(inStream: TStream; outStream: TStream; level: TCompressionLevel);
overload;
```

C++

```
ZCompressStream(TStream inStream, TStream outStream, TCompressionLevel level);
```

File

ZLib

Description

This is function ZLib.ZCompressStream.

91.2.17 ZLib.ZCompressStream Function

Pascal

```
procedure ZCompressStream(inStream: TStream; outStream: TStream; level: TZCompressionLevel =
zcDefault); overload;
```

C++

```
ZCompressStream(TStream inStream, TStream outStream, TZCompressionLevel level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompressStream.

91.2.18 ZLib.ZDecompress Function

Pascal

```
procedure ZDecompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; outEstimate: Integer = 0);
```

C++

```
ZDecompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, int outEstimate
= 0);
```

File

ZLib

Description

This is function ZLib.ZDecompress.

91.2.19 ZLib.ZDecompressStr Function

Pascal

```
function ZDecompressStr(const s: TBytes): string;
```

C++

```
UnicodeString ZDecompressStr(const TBytes s);
```

File

ZLib

Description

This is function ZLib.ZDecompressStr.

91.2.20 ZLib.ZDecompressStream Function

Pascal

```
procedure ZDecompressStream(inStream: TStream; outStream: TStream);
```

C++

```
ZDecompressStream(TStream inStream, TStream outStream);
```

File

ZLib

Description

This is function ZLib.ZDecompressStream.

91.2.21 ZLib.zlibAllocMem Function

Pascal

```
function zlibAllocMem(AppData: Pointer; Items: Integer; Size: Integer): Pointer;
```

C++

```
void * zlibAllocMem(void * AppData, int Items, int Size);
```

File

ZLib

Description

This is function ZLib.zlibAllocMem.

91.2.22 ZLib.zlibFreeMem Function

Pascal

```
procedure zlibFreeMem(AppData: Pointer; Block: Pointer);
```

C++

```
zlibFreeMem(void * AppData, void * Block);
```

File

ZLib

Description

This is function ZLib.zlibFreeMem.

91.2.23 ZLib.ZSendToBrowser Function

Pascal

```
procedure ZSendToBrowser(var s: string);
```

C++

```
ZSendToBrowser(UnicodeString s);
```

File

ZLib

Description

This is function ZLib.ZSendToBrowser.

91.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCompressionLevel (see page 7868)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.
	TZCompressionLevel (see page 7869)	This is record ZLib.TZCompressionLevel.

Records

	Name	Description
	TZStreamRec (see page 7869)	This is record ZLib.TZStreamRec.

91.3.1 ZLib.TCompressionLevel Enumeration

TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.

Pascal

```
TCompressionLevel = (
  clNone = Integer(zcNone
);
```

C++

```
enum TCompressionLevel {
  clNone = Integer(zcNone
};
```

File

ZLib

Description

The following table lists the possible values:

Value	Meaning
clNone	Specifies no compression; data is merely copied to the output stream.
clFastest	Specifies fastest compression, resulting in a larger archive.
clDefault	Compromise between speed and amount of compression.
clMax	Specifies maximum compression, resulting in a longer time to perform the operation.

See Also

TCustomZlibStream

TDecompressionStream (see page 7872)

TCompressionStream (see page 7871)

91.3.2 ZLib.TZCompressionLevel Enumeration

Pascal

```
TZCompressionLevel = (
  zcNone,
  zcFastest,
  zcDefault,
  zcMax
);
```

C++

```
enum TZCompressionLevel {
  zcNone,
  zcFastest,
  zcDefault,
  zcMax
};
```

File

ZLib

Description

This is record ZLib.TZCompressionLevel.

91.3.3 ZLib.TZStreamRec Record

Pascal

```
TZStreamRec = packed record
  next_in: PAnsiChar;
  avail_in: Longint;
  total_in: Longint;
  next_out: PAnsiChar;
  avail_out: Longint;
  total_out: Longint;
  msg: PAnsiChar;
  state: Pointer;
  zalloc: TZAlloc;
  zfree: TZFree;
  opaque: Pointer;
  data_type: Integer;
  adler: Longint;
  reserved: Longint;
end;
```

C++

```
struct TZStreamRec {
  PAnsiChar next_in;
  Longint avail_in;
  Longint total_in;
  PAnsiChar next_out;
  Longint avail_out;
  Longint total_out;
  PAnsiChar msg;
  void * state;
```

```
    TZAlloc zalloc;
    TZFree zfree;
    void * opaque;
    int data_type;
    Longint adler;
    Longint reserved;
  };
```

File

ZLib

Description

This is record ZLib.TZStreamRec.

91.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCompressionStream (see page 7871)	Compresses data as it is written to the stream.
TCustomZLibStream (see page 7872)	This is type ZLib.TCustomZLibStream.
TDecompressionStream (see page 7872)	Decompresses data that was written to a TCompressionStream (see page 7871) object.
TZAlloc (see page 7872)	This is type ZLib.TZAlloc.
TZFree (see page 7873)	This is type ZLib.TZFree.

91.4.1 ZLib.TCompressionStream Type

Compresses data as it is written to the stream.

Pascal

```
TCompressionStream = TZCompressionStream;
```

C++

```
TZCompressionStream TCompressionStream;
```

File

ZLib

Description

TCompressionStream is a write-only, sequential access stream that compresses data as it is written. The compressed data is written to a separate stream, which is passed to the TCompressionStream constructor. TCompressionStream does not take ownership of the output stream; you are responsible for creating, initializing, and destroying the stream when it is no longer needed.

Reading from a TCompressionStream object will raise an exception of type ECompressionError, with the message string set to "Invalid Stream Operation." Similarly, a Seek (see page 3741) operation causes an ECompressionError exception, however, you can perform a Seek (see page 3741) operation with a zero offset from the current position. This will return the number of raw, uncompressed bytes that have been written to the stream.

You set the compression level to be used when you create the stream object. The possible compression levels are: clNone, clFastest, clDefault, and clMax.

When the stream's output buffer is filled, a Write (see page 3789) operation on TCompressionStream will write the contents of the buffer to the output stream, and then call the OnProgress event. You can use the OnProgress event to update a user interface control such as a progress indicator.

See Also

TCustomZlibStream

TDecompressionStream (see page 7872)

TStream

91.4.2 ZLib.TCustomZLibStream Type

Pascal

```
TCustomZLibStream = TCustomZStream;
```

C++

```
TCustomZStream TCustomZLibStream;
```

File

ZLib

Description

This is type ZLib.TCustomZLibStream.

91.4.3 ZLib.TDecompressionStream Type

Decompresses data that was written to a TCompressionStream (see page 7871) object.

Pascal

```
TDecompressionStream = TZDecompressionStream;
```

C++

```
TZDecompressionStream TDecompressionStream;
```

File

ZLib

Description

TDecompressionStream is the analog of the TCompressionStream (see page 7871) class. It is a read-only, unidirectional stream; you can seek forward in the stream, but not backwards. A separate input stream provides the source data. The input stream will be a stream that was previously written to using a TCompressionStream (see page 7871) object. The input stream is passed to the constructor, however, TDecompressionStream does not take ownership of this object. You are responsible for creating, initializing, and destroying the input stream when it is no longer needed.

Writing to the stream will raise an exception, as will requesting the size of the stream, and attempting to perform a Seek (see page 3741) operation backwards or relative to the current position.

The TDecompressionStream object's OnProgress event is called when a block of data is read from the input stream. You can read the TDecompressionStream object's Position property to determine the number of raw, uncompressed bytes that have been read from the input stream so far.

91.4.4 ZLib.TZAlloc Type

Pascal

```
TZAlloc = function (opaque: Pointer; items, size: Integer): Pointer;
```

C++

```
(opaque: Pointer; items, size: Integer): Pointer ( TZAlloc)();
```

File

ZLib

Description

This is type ZLib.TZAlloc.

91.4.5 ZLib.TZFree Type

Pascal

```
TZFree = procedure (opaque, block: Pointer);
```

C++

```
(opaque, block: Pointer) ( TZFree)();
```

File

ZLib

Description

This is type ZLib.TZFree.

91.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
_z_errmsg (see page 7874)	This is constant ZLib._z_errmsg.
SZInvalid (see page 7875)	This is constant ZLib.SZInvalid.
Z_ASCII (see page 7875)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 7875)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 7876)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 7876)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 7876)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 7877)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 7877)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 7877)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 7878)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 7878)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 7878)	This is constant ZLib.Z_FILTERED.
Z_FINISH (see page 7879)	This is constant ZLib.Z_FINISH.
Z_FULL_FLUSH (see page 7879)	This is constant ZLib.Z_FULL_FLUSH.
Z_HUFFMAN_ONLY (see page 7879)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 7880)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 7880)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 7880)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 7881)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 7881)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 7881)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 7882)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 7882)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 7882)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 7883)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 7883)	This is constant ZLib.Z_VERSION_ERROR.
ZLevels (see page 7883)	This is constant ZLib.ZLevels.
ZLIB_VERSION (see page 7884)	This is constant ZLib.ZLIB_VERSION.

91.5.1 ZLib._z_errmsg Constant

Pascal

```
_z_errmsg: array[0..9] of PChar = ( 'need dictionary', 'stream end', '', 'file error', 'stream
error', 'data error', 'insufficient memory', 'buffer error', 'incompatible version', '' );
```

C++

```
array[0..9] of PChar _z_errmsg = ( 'need dictionary', 'stream end', '', 'file error', 'stream
error', 'data error', 'insufficient memory', 'buffer error', 'incompatible version', '' );
```

File

ZLib

Description

This is constant ZLib._z_errmsg.

91.5.2 ZLib.SZInvalid Constant

Pascal

```
SZInvalid = 'Invalid ZStream operation!';
```

C++

```
SZInvalid = 'Invalid ZStream operation!';
```

File

ZLib

Description

This is constant ZLib.SZInvalid.

91.5.3 ZLib.Z_ASCII Constant

Pascal

```
Z_ASCII = 1;
```

C++

```
Z_ASCII = 1;
```

File

ZLib

Description

This is constant ZLib.Z_ASCII.

91.5.4 ZLib.Z_BEST_COMPRESSION Constant

Pascal

```
Z_BEST_COMPRESSION = 9;
```

C++

```
Z_BEST_COMPRESSION = 9;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_COMPRESSION.

91.5.5 ZLib.Z_BEST_SPEED Constant

Pascal

```
Z_BEST_SPEED = 1;
```

C++

```
Z_BEST_SPEED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_SPEED.

91.5.6 ZLib.Z_BINARY Constant

Pascal

```
Z_BINARY = 0;
```

C++

```
Z_BINARY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_BINARY.

91.5.7 ZLib.Z_BUF_ERROR Constant

Pascal

```
Z_BUF_ERROR = (-5);
```

C++

```
Z_BUF_ERROR = (-5);
```

File

ZLib

Description

This is constant ZLib.Z_BUF_ERROR.

91.5.8 ZLib.Z_DATA_ERROR Constant

Pascal

```
Z_DATA_ERROR = (-3);
```

C++

```
Z_DATA_ERROR = (-3);
```

File

ZLib

Description

This is constant ZLib.Z_DATA_ERROR.

91.5.9 ZLib.Z_DEFAULT_COMPRESSION Constant

Pascal

```
Z_DEFAULT_COMPRESSION = (-1);
```

C++

```
Z_DEFAULT_COMPRESSION = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_COMPRESSION.

91.5.10 ZLib.Z_DEFAULT_STRATEGY Constant

Pascal

```
Z_DEFAULT_STRATEGY = 0;
```

C++

```
Z_DEFAULT_STRATEGY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_STRATEGY.

91.5.11 ZLib.Z_DEFLATED Constant

Pascal

```
Z_DEFLATED = 8;
```

C++

```
Z_DEFLATED = 8;
```

File

ZLib

Description

This is constant ZLib.Z_DEFLATED.

91.5.12 ZLib.Z_ERRNO Constant

Pascal

```
Z_ERRNO = (-1);
```

C++

```
Z_ERRNO = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_ERRNO.

91.5.13 ZLib.Z_FILTERED Constant

Pascal

```
Z_FILTERED = 1;
```

C++

```
Z_FILTERED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_FILTERED.

91.5.14 ZLib.Z_FINISH Constant

Pascal

```
Z_FINISH = 4;
```

C++

```
Z_FINISH = 4;
```

File

ZLib

Description

This is constant ZLib.Z_FINISH.

91.5.15 ZLib.Z_FULL_FLUSH Constant

Pascal

```
Z_FULL_FLUSH = 3;
```

C++

```
Z_FULL_FLUSH = 3;
```

File

ZLib

Description

This is constant ZLib.Z_FULL_FLUSH.

91.5.16 ZLib.Z_HUFFMAN_ONLY Constant

Pascal

```
Z_HUFFMAN_ONLY = 2;
```

C++

```
Z_HUFFMAN_ONLY = 2;
```

File

ZLib

Description

This is constant ZLib.Z_HUFFMAN_ONLY.

91.5.17 ZLib.Z_MEM_ERROR Constant

Pascal

```
Z_MEM_ERROR = (-4);
```

C++

```
Z_MEM_ERROR = (-4);
```

File

ZLib

Description

This is constant ZLib.Z_MEM_ERROR.

91.5.18 ZLib.Z_NEED_DICT Constant

Pascal

```
Z_NEED_DICT = 2;
```

C++

```
Z_NEED_DICT = 2;
```

File

ZLib

Description

This is constant ZLib.Z_NEED_DICT.

91.5.19 ZLib.Z_NO_COMPRESSION Constant

Pascal

```
Z_NO_COMPRESSION = 0;
```

C++

```
Z_NO_COMPRESSION = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_COMPRESSION.

91.5.20 ZLib.Z_NO_FLUSH Constant

Pascal

```
Z_NO_FLUSH = 0;
```

C++

```
Z_NO_FLUSH = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_FLUSH.

91.5.21 ZLib.Z_OK Constant

Pascal

```
Z_OK = 0;
```

C++

```
Z_OK = 0;
```

File

ZLib

Description

This is constant ZLib.Z_OK.

91.5.22 ZLib.Z_PARTIAL_FLUSH Constant

Pascal

```
Z_PARTIAL_FLUSH = 1;
```

C++

```
Z_PARTIAL_FLUSH = 1;
```

File

ZLib

Description

This is constant ZLib.Z_PARTIAL_FLUSH.

91.5.23 ZLib.Z_STREAM_END Constant

Pascal

```
Z_STREAM_END = 1;
```

C++

```
Z_STREAM_END = 1;
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_END.

91.5.24 ZLib.Z_STREAM_ERROR Constant

Pascal

```
Z_STREAM_ERROR = (-2);
```

C++

```
Z_STREAM_ERROR = (-2);
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_ERROR.

91.5.25 ZLib.Z_SYNC_FLUSH Constant

Pascal

```
Z_SYNC_FLUSH = 2;
```

C++

```
Z_SYNC_FLUSH = 2;
```

File

ZLib

Description

This is constant ZLib.Z_SYNC_FLUSH.

91.5.26 ZLib.Z_UNKNOWN Constant

Pascal

```
Z_UNKNOWN = 2;
```

C++

```
Z_UNKNOWN = 2;
```

File

ZLib

Description

This is constant ZLib.Z_UNKNOWN.

91.5.27 ZLib.Z_VERSION_ERROR Constant

Pascal

```
Z_VERSION_ERROR = (-6);
```

C++

```
Z_VERSION_ERROR = (-6);
```

File

ZLib

Description

This is constant ZLib.Z_VERSION_ERROR.

91.5.28 ZLib.ZLevels Constant

Pascal

```
ZLevels: array[TZCompressionLevel] of Shortint = ( Z_NO_COMPRESSION, Z_BEST_SPEED,
Z_DEFAULT_COMPRESSION, Z_BEST_COMPRESSION );
```

C++

```
array[TZCompressionLevel] of Shortint ZLevels = ( Z_NO_COMPRESSION, Z_BEST_SPEED,
Z_DEFAULT_COMPRESSION, Z_BEST_COMPRESSION );
```

File

ZLib

Description

This is constant ZLib.ZLevels.

91.5.29 ZLib.ZLIB_VERSION Constant

Pascal

```
ZLIB_VERSION = '1.2.3';
```

C++

```
ZLIB_VERSION = '1.2.3';
```

File

ZLib

Description

This is constant ZLib.ZLIB_VERSION.

92 ZLibConst Namespace

This is namespace ZLibConst.

Topics

Name	Description
Constants (see page 7886)	The following table lists constants in this documentation.

Constants

Name	Description
sError (see page 7886)	This is constant ZLibConst.sError.
sInvalidStreamOp (see page 7886)	This is constant ZLibConst.sInvalidStreamOp.
sTargetBufferTooSmall (see page 7886)	This is constant ZLibConst.sTargetBufferTooSmall.

92.1 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sError (see page 7886)	This is constant ZLibConst.sError.
sInvalidStreamOp (see page 7886)	This is constant ZLibConst.sInvalidStreamOp.
sTargetBufferTooSmall (see page 7886)	This is constant ZLibConst.sTargetBufferTooSmall.

92.1.1 ZLibConst.sError Constant

Pascal

```
sError = 'Error';
```

C++

```
sError = 'Error';
```

File

ZLibConst

Description

This is constant ZLibConst.sError.

92.1.2 ZLibConst.sInvalidStreamOp Constant

Pascal

```
sInvalidStreamOp = 'Invalid stream operation';
```

C++

```
sInvalidStreamOp = 'Invalid stream operation';
```

File

ZLibConst

Description

This is constant ZLibConst.sInvalidStreamOp.

92.1.3 ZLibConst.sTargetBufferTooSmall Constant

Pascal

```
sTargetBufferTooSmall = 'ZLib error: target buffer may be too small';
```

C++

```
sTargetBufferTooSmall = 'ZLib error: target buffer may be too small';
```

File

ZLibConst

Description

This is constant ZLibConst.sTargetBufferTooSmall.

Index

_

_SearchAssistantEvents interface 571
- _SearchAssistantEvents Members 572
- _SearchAssistantEvents methods 572
- about _SearchAssistantEvents interface 571
- OnNewSearch 572
- OnNextMenuSelect 572

A

adCantView enumeration member 6239

adLoginRequired enumeration member 6239

AnsiStringBase class 3185
- != 3215
- [] 3216
- [] operator 3216
- _AnsiCat 3196
- _AnsiCat method 3195
- _AnsiFromPWChar 3196
- ~AnsiStringBase 3189
- + 3216
- < 3216
- <= 3217
- = 3217
- == 3218
- > 3218
- >= 3218
- about AnsiStringBase class 3185
- AnsiCompare 3196
- AnsiCompareIC 3197
- AnsiLastChar 3197
- AnsiPos 3197
- AnsiStringBase 3190, 3191, 3192
- AnsiStringBase constructor 3189
- AnsiStringBase data members 3194
- AnsiStringBase enumerations 3193
- AnsiStringBase Members 3187
- AnsiStringBase methods 3194
- AnsiStringBase operators 3215
- AnsiStringBase structures 3192
- BytesOf 3198
- ByteType 3198
- cat_vprintf 3199
- CodePage 3199
- CurrToStr 3199
- CurrToStrF 3200
- Data 3194
- Delete 3200
- ElementSize 3200
- EnsureAnsi 3201
- FloatToStrF 3201
- FmtLoadStr 3202
- Format 3202
- FormatFloat 3203
- GetRec 3204
- GetRec method 3204
- Insert 3205
- IntToHex 3205
- IsDelimiter 3206
- IsEmpty 3206
- IsLeadByte 3206
- IsPathDelimiter 3206
- IsTrailByte 3207
- LastChar 3207
- LastDelimiter 3207
- Length 3208
- LoadStr 3208
- LoadStr method 3208
- LoadString 3209
- LowerCase 3209
- Pos 3209
- RefCount 3210
- SetLength 3210
- StringOfChar 3210
- SubString 3211
- swap 3211

ThrowIfOutOfRange 3211
ToDouble 3211
ToInt 3212
ToIntDef 3212
Trim 3212
TrimLeft 3212
TrimRight 3213
Unique 3213
UpperCase 3213
vprintf 3214
WideChar 3214
WideCharBufSize 3214
AnsiStringT template 3219
!= 3240
+ 3240
+= 3240
< 3240
<= 3241
= 3241
== 3241
> 3241
>= 3241
about AnsiStringT template 3219
AnsiCompare 3229
AnsiCompareIC 3229
AnsiPos 3229
AnsiStringT 3223, 3224, 3225, 3226, 3227, 3228
AnsiStringT constructor 3222
AnsiStringT Members 3220
AnsiStringT methods 3228
AnsiStringT operators 3239
c_str 3229
cat_printf 3230
cat_sprintf 3230
cat_vprintf 3230
CodePage 3230
CurrToStr 3231
CurrToStrF 3231
data 3231
Delete 3232
ElementSize 3232
EnsureAnsi 3232
FloatToStrF 3233
FmtLoadStr 3233
Format 3233
FormatFloat 3233
Insert 3234
IntToHex 3234
IsDelimiter 3234
LastDelimiter 3234
LoadStr 3235
LoadStr method 3234
LoadString 3235
LowerCase 3235
Pos 3236
printf 3236
RefCount 3236
SetLength 3236
sprintf 3237
StringOfChar 3237
SubString 3237
swap 3237
Trim 3238
TrimLeft 3238
TrimRight 3238
Unique 3238
UpperCase 3238
vprintf 3239
WideChar 3239
WideCharBufSize 3239
argIn enumeration member 6706
argInOut enumeration member 6706
argOut enumeration member 6706
argReturn enumeration member 6706
asAccept enumeration member 134
asClose enumeration member 134
asConnect enumeration member 134
asOOB enumeration member 134

asRead enumeration member 134
asWrite enumeration member 134

B

bhoHideAlways enumeration member 6488
bhoHideOnActionNotVisible enumeration member 6488
bhoHideOnDisabledAction enumeration member 6488
bhoHideOnNoExecuteAccess enumeration member 6488
bifBrowseForComputer enumeration member 1875
bifBrowseForPrinter enumeration member 1875
bifBrowseIncludeFiles enumeration member 1875
bifBrowseIncludeURLS enumeration member 1875
bifDontGoBelowDomain enumeration member 1875
bifEditBox enumeration member 1875
bifNewDialogStyle enumeration member 1875
bifNoNewFolderButton enumeration member 1875
bifNoTranslateTargets enumeration member 1875
bifReturnFSAncestors enumeration member 1875
bifReturnOnlyFSDirs enumeration member 1875
bifShareable enumeration member 1875
bifStatusText enumeration member 1875
bifUAHint enumeration member 1875
bifUseNewUI enumeration member 1875
bifValidate enumeration member 1875
bmBlocking enumeration member 1404
bmNonBlocking enumeration member 1404
bmThreadBlocking enumeration member 1404
bsCommandLink enumeration member 2181
bsPushButton enumeration member 2181
bsrAsIs enumeration member 5105
bsrLower enumeration member 5105
bsrUpper enumeration member 5105
bsSplitButton enumeration member 2181
btDIME enumeration member 7137
btHttp enumeration member 7137
btMime enumeration member 7137
btSoap enumeration member 7137
btUnknown enumeration member 7137

C

cbChecked enumeration member 2906
cbGrayed enumeration member 2906
cbUnchecked enumeration member 2906
clNone enumeration member 7868
CoCScriptErrorList class 371
  about CoCScriptErrorList class 371
  CoCScriptErrorList Members 371
  CoCScriptErrorList methods 372
  Create 372
  CreateRemote 372
CoInternetExplorer class 372
  about CoInternetExplorer class 372
  CoInternetExplorer Members 373
  CoInternetExplorer methods 373
  Create 373
  CreateRemote 373
coLocalPaths enumeration member 6082
coltDefault enumeration member 6489
coltImage enumeration member 6489
coltList enumeration member 6489
coltText enumeration member 6489
coNoExecuteScript enumeration member 6082
coNoScript enumeration member 6082
CoSearchAssistantOC class 373
  about CoSearchAssistantOC class 373
  CoSearchAssistantOC Members 374
  CoSearchAssistantOC methods 374
  Create 374
  CreateRemote 374
CoShellBrowserWindow class 375
  about CoShellBrowserWindow class 375
  CoShellBrowserWindow Members 375
  CoShellBrowserWindow methods 375
  Create 375
  CreateRemote 376
CoShellNameSpace class 376
  about CoShellNameSpace class 376

CoShellNameSpace Members 376
CoShellNameSpace methods 376
Create 377
CreateRemote 377
CoShellUIHelper class 377
about CoShellUIHelper class 377
CoShellUIHelper Members 377
CoShellUIHelper methods 378
Create 378
CreateRemote 378
CoShellWindows class 378
about CoShellWindows class 378
CoShellWindows Members 379
CoShellWindows methods 379
Create 379
CreateRemote 379
crEqual enumeration member 5106
crGreaterThan enumeration member 5106
crLessThan enumeration member 5106
csAllocated enumeration member 3796
csBottom enumeration member 358
csContinuePending enumeration member 3048
csDropDown enumeration member 2907
csDropDownList enumeration member 2907
csOwnerDrawFixed enumeration member 2907
csOwnerDrawVariable enumeration member 2907
csPaused enumeration member 3048
csPausePending enumeration member 3048
csReserved enumeration member 3796
csRight enumeration member 358
csRunning enumeration member 3048
csSimple enumeration member 2907
csStartPending enumeration member 3048
csStateClosed enumeration member 1627
csStateConnecting enumeration member 1627
csStateDisconnecting enumeration member 1627
csStateExecuting enumeration member 1627
csStateFetching enumeration member 1627
csStateOpen enumeration member 1627
csStopped enumeration member 3048
csStopPending enumeration member 3048
csSysAllocated enumeration member 3796
csSysReserved enumeration member 3796
csUnallocated enumeration member 3796
ctAnchor enumeration member 6489
ctbLeadSurrogate enumeration member 3399
ctBlocking enumeration member 135
ctButton enumeration member 6489
ctDefault enumeration member 6489
ctEventImages enumeration member 6489
ctImage enumeration member 6489
ctNonBlocking enumeration member 135
ctNotLeadChar enumeration member 3399
ctTrailSurrogate enumeration member 3399
Currency class 3242
- 3246
-- 3247
- operator 3245
-- operator 3246
! 3247
!= 3248
!= operator 3248
% 3248
%= 3249
* 3249
* operator 3249
*= 3250
/ 3251
/ operator 3250
/= 3251
+ 3252
+ operator 3252
++ 3253
++ operator 3253
+= 3253
< 3254
< operator 3254
<= 3255

<= operator 3255
= 3256
-= 3256
= operator 3255
== 3257
== operator 3257
> 3258
> operator 3257
>= 3258
>= operator 3258
about Currency class 3242
Currency 3244
Currency __fastcall operator -(double lhs, const Currency& rhs) 3260
Currency __fastcall operator -(int lhs, const Currency& rhs) 3261
Currency __fastcall operator *(double lhs, const Currency& rhs) 3261
Currency __fastcall operator *(int lhs, const Currency& rhs) 3261
Currency __fastcall operator /(double lhs, const Currency& rhs) 3261
Currency __fastcall operator /(int lhs, const Currency& rhs) 3262
Currency __fastcall operator +(double lhs, const Currency& rhs) 3262
Currency __fastcall operator +(int lhs, const Currency& rhs) 3262
Currency constructor 3244
Currency friends 3260
Currency Members 3242
Currency operators 3245
double 3259
int 3259
System::String 3260
CurrencyBase class 3262
about CurrencyBase class 3262
CurrencyBase data members 3263
CurrencyBase Members 3263
Val 3263

D

Date enumeration member 3340
DateTime enumeration member 3340
dcChangeCurrentRecordView enumeration member 1062
dcCurrentRecordView enumeration member 1062
dcMultipleRecordView enumeration member 1062
DelphiInterface template 3263
! 3266
& 3266
* 3267
~DelphiInterface 3264
= 3267
= operator 3267
-> 3268
about DelphiInterface template 3263
DelphiInterface 3265
DelphiInterface constructor 3265
DelphiInterface Members 3264
DelphiInterface methods 3265
DelphiInterface operators 3266
Release 3266
T* 3268
dftDefault enumeration member 6491
dftImage enumeration member 6491
dftList enumeration member 6491
dftText enumeration member 6491
doAttrNull enumeration member 7664
doAutoColResize enumeration member 4921
doAutoPrefix enumeration member 7664
doAutoSave enumeration member 7664
doColumnTitles enumeration member 4921
doKeyColFixed enumeration member 4921
DOMException class 7458
about DOMException class 7458
code 7460
DOMException fields 7460
DOMException Members 7458
doNamespaceDecl enumeration member 7664

doNodeAutoCreate enumeration member 7664
doNodeAutoIndent enumeration member 7664
dpPublished enumeration member 1065
DShellNameSpaceEvents interface 573
about DShellNameSpaceEvents interface 573
DoubleClick 573
DShellNameSpaceEvents Members 573
DShellNameSpaceEvents methods 573
FavoritesSelectionChange 574
Initialized 574
SelectionChange 574
DShellWindowsEvents interface 574
about DShellWindowsEvents interface 574
DShellWindowsEvents Members 575
DShellWindowsEvents methods 575
WindowRegistered 575
WindowRevoked 575
dtHelp enumeration member 317
dtManual enumeration member 317
dtResource enumeration member 317
DWebBrowserEvents interface 576
about DWebBrowserEvents interface 576
BeforeNavigate 577
CommandStateChange 577
DownloadBegin 577
DownloadComplete 578
DWebBrowserEvents Members 576
DWebBrowserEvents methods 576
FrameBeforeNavigate 578
FrameNavigateComplete 578
FrameNewWindow 578
NavigateComplete 579
NewWindow 579
ProgressChange 579
PropertyChange 579
Quit 579
StatusTextChange 580
TitleChange 580
WindowActivate 580
WindowMove 580
WindowResize 581
DWebBrowserEvents2 interface 581
about DWebBrowserEvents2 interface 581
BeforeNavigate2 583
ClientToHostWindow 583
CommandStateChange 583
DocumentComplete 583
DownloadBegin 584
DownloadComplete 584
DWebBrowserEvents2 Members 581
DWebBrowserEvents2 methods 582
FileDownload 584
NavigateComplete2 584
NavigateError 584
NewWindow2 585
OnFullScreen 585
OnMenuBar 585
OnQuit 585
OnStatusBar 586
OnTheaterMode 586
OnToolBar 586
OnVisible 586
PrintTemplateInstantiation 586
PrintTemplateTeardown 587
PrivacyImpactedStateChange 587
ProgressChange 587
PropertyChange 587
SetSecureLockIcon 588
StatusTextChange 588
TitleChange 588
UpdatePageStatus 588
WindowClosing 588
WindowSetHeight 589
WindowSetLeft 589
WindowSetResizable 589
WindowSetTop 589
WindowSetWidth 590
DynamicArray template 3268

[] 3276
[] operator 3276
~DynamicArray 3271
= 3276
== 3277
about DynamicArray template 3268
AllocData 3272
Copy 3273
Copy method 3272
CopyRange 3273
CopyRange method 3273
DecRefCount 3274
DimCount 3274
DynamicArray 3271
DynamicArray constructor 3271
DynamicArray Members 3270
DynamicArray methods 3272
DynamicArray operators 3276
DynamicArray properties 3277
FreeData 3274
get_high 3274
get_length 3275
get_low 3275
High 3277
IncRefCount 3275
Length 3278
Low 3278
set_length 3275
SetData 3275
DynArrayException class 3279
about DynArrayException class 3279
DynArrayNullData class 3279
about DynArrayNullData class 3279
DynArrayOutOfRange class 3279
about DynArrayOutOfRange class 3279
DynArrayOutOfRange 3280
DynArrayOutOfRange data members 3280
DynArrayOutOfRange Members 3280
m_Count 3280
m_Index 3281

E

EAbort class 3982
about EAbort class 3982
EAbstractError class 3982
about EAbstractError class 3982
EAccessViolation class 3983
about EAccessViolation class 3983
EAdapterDispatcherException class 6120
about EAdapterDispatcherException class 6120
EAdapterException class 5166
about EAdapterException class 5166
CreateID 5167
EAdapterException Members 5166
EAdapterException properties 5167
ID 5167
EAdapterFieldException class 5167
about EAdapterFieldException class 5167
Create 5168
EAdapterFieldException Members 5168
EAdapterFieldException properties 5169
FieldName 5169
EAllocatedActionsException class 251
about EAllocatedActionsException class 251
EArgumentException class 3984
about EArgumentException class 3984
EArgumentOutOfRangeException class 3984
about EArgumentOutOfRangeException class 3984
EAssertionFailed class 3984
about EAssertionFailed class 3984
ebBottom enumeration member 4678
ebLeft enumeration member 4678
ebRight enumeration member 4678
ebTop enumeration member 4678
ecLowerCase enumeration member 2907
ecNormal enumeration member 2907
eConnect enumeration member 1624
EControlC class 3985

about EControlC class 3985
EConvertError class 3985
about EConvertError class 3985
ecUpperCase enumeration member 2907
ECustomException class 1241
about ECustomException class 1241
CatchException 1246
Create 1243
Destroy 1245
ECustomException Members 1241
ECustomException methods 1246
InstanceCount 1246
ThrowExcept 1246
eDisconnect enumeration member 1624
EDivByZero class 3987
about EDivByZero class 3987
EDOMParseError class 7460
about EDOMParseError class 7460
Create 7462
EDOMParseError Members 7461
EDOMParseError properties 7463
ErrorCode 7463
FilePos 7463
Line 7464
LinePos 7464
ParseError 7464
Reason 7465
SrcText 7465
URL 7465
eeAccept enumeration member 136
eeConnect enumeration member 136
eeDisconnect enumeration member 136
eeGeneral enumeration member 136
eeLookup enumeration member 136
EEncodingError class 3988
about EEncodingError class 3988
eeReceive enumeration member 136
eeSend enumeration member 136
EExternal class 3988
about EExternal class 3988
EExternal fields 3991
EExternal Members 3989
ExceptionRecord 3991
EExternalException class 3991
about EExternalException class 3991
EHeapException class 3992
about EHeapException class 3992
EHeapException Members 3993
EHeapException methods 3994
FreeInstance 3994
RaisingException 3995
EInOutError class 3995
about EInOutError class 3995
EInOutError fields 3998
EInOutError Members 3997
ErrorCode 3998
EInterpreterError class 147
about EInterpreterError class 147
EIntError class 3999
about EIntError class 3999
EIntfCastError class 3999
about EIntfCastError class 3999
EIntOverflow class 4000
about EIntOverflow class 4000
EInvalidCast class 4000
about EInvalidCast class 4000
EInvalidContainer class 4001
about EInvalidContainer class 4001
EInvalidInsert class 4001
about EInvalidInsert class 4001
EInvalidOp class 4002
about EInvalidOp class 4002
EInvalidPointer class 4002
about EInvalidPointer class 4002
EMathError class 4003
about EMathError class 4003
EMonitor class 4004
about EMonitor class 4004

EMonitorLockException class 4004
about EMonitorLockException class 4004
ENamedVariantsError class 6090
about ENamedVariantsError class 6090
ENoConstructException class 4004
about ENoConstructException class 4004
ENoMonitorSupportException class 4005
about ENoMonitorSupportException class 4005
EOSError class 4005
about EOSError class 4005
EOSError fields 4007
EOSError Members 4006
ErrorCode 4007
EOutOfMemory class 4008
about EOutOfMemory class 4008
EOverflow class 4008
about EOverflow class 4008
EPackageError class 4009
about EPackageError class 4009
EPageDispatcherException class 6121
about EPageDispatcherException class 6121
epEndEllipsis enumeration member 2908
epNone enumeration member 2908
epPathEllipsis enumeration member 2908
EPrivilege class 4009
about EPrivilege class 4009
EProgrammerNotFound class 4010
about EProgrammerNotFound class 4010
EPropertyConvertError class 4727
about EPropertyConvertError class 4727
EPropertyError class 4727
about EPropertyError class 4727
EPropReadOnly class 4010
about EPropReadOnly class 4010
EPropWriteOnly class 4011
about EPropWriteOnly class 4011
epSelectedLeft enumeration member 4573
epSelectedRight enumeration member 4573
epUnselectedLeft enumeration member 4573
epUnselectedRight enumeration member 4573
epWordEllipsis enumeration member 2908
ERangeError class 4011
about ERangeError class 4011
eReconnect enumeration member 1624
ERecursiveIncludeFile class 6613
about ERecursiveIncludeFile class 6613
ESafeArrayBoundsError class 5120
about ESafeArrayBoundsError class 5120
ESafeArrayCreateError class 5120
about ESafeArrayCreateError class 5120
ESafeArrayError class 5121
about ESafeArrayError class 5121
CreateHResult 5122
ErrorCode 5123
ESafeArrayError Members 5121
ESafeArrayError properties 5123
ESafeArrayLockedError class 5123
about ESafeArrayLockedError class 5123
ESafecallException class 4012
about ESafecallException class 4012
EScriptEvalError class 6614
about EScriptEvalError class 6614
esCritical enumeration member 3049
ESessionException class 6122
about ESessionException class 6122
ESessionsException class 328
about ESessionsException class 328
esIgnore enumeration member 3049
esLowered enumeration member 4679
esNone enumeration member 4679
esNormal enumeration member 3049
ESOAPHTTPException class 1210
about ESOAPHTTPException class 1210
Create 1212
CreateFmt 1212
ESOAPHTTPException Members 1210
ESOAPHTTPException properties 1213
StatusCode 1213

ESocketConnectionError class 148
about ESocketConnectionError class 148
ESocketError class 3, 1301
about ESocketError class 3, 1301
esRaised enumeration member 4679
esSevere enumeration member 3049
EStackOverflow class 4013
about EStackOverflow class 4013
esTerminate enumeration member 6739
esTimeout enumeration member 6739
etANSI enumeration member 6842
etFirstIsSel enumeration member 4573
etFirstNotSel enumeration member 4573
etLastIsSel enumeration member 4573
etLastNotSel enumeration member 4573
etNone enumeration member 4573
etNotSelToNotSel enumeration member 4573
etNotSelToSel enumeration member 4573
etSelToNotSel enumeration member 4573
etUSASCII enumeration member 6842
etUTF8 enumeration member 6842
ETypeTransException class 4714
about ETypeTransException class 4714
EUnderflow class 4013
about EUnderflow class 4013
eUserDataSet enumeration member 1627
EUserIDNotFoundException class 6742
about EUserIDNotFoundException class 6742
eUserMonitor enumeration member 1627
EValidateUserException class 6742
about EValidateUserException class 6742
EVariantArrayCreateError class 5002
about EVariantArrayCreateError class 5002
EVariantArrayLockedError class 5003
about EVariantArrayLockedError class 5003
EVariantBadIndexError class 5004
about EVariantBadIndexError class 5004
EVariantBadVarTypeError class 5004
about EVariantBadVarTypeError class 5004
EVariantDispatchError class 5005
about EVariantDispatchError class 5005
EVariantError class 4014
about EVariantError class 4014
EVariantInvalidArgError class 5005
about EVariantInvalidArgError class 5005
EVariantInvalidNullOpError class 5006
about EVariantInvalidNullOpError class 5006
EVariantInvalidOpError class 5007
about EVariantInvalidOpError class 5007
EVariantNotAnArrayError class 5007
about EVariantNotAnArrayError class 5007
EVariantNotImplError class 5008
about EVariantNotImplError class 5008
EVariantOutOfMemoryError class 5008
about EVariantOutOfMemoryError class 5008
EVariantOverflowError class 5009
about EVariantOverflowError class 5009
EVariantRangeCheckError class 5010
about EVariantRangeCheckError class 5010
EVariantTypeCastError class 5010
about EVariantTypeCastError class 5010
EVariantUnexpectedError class 5011
about EVariantUnexpectedError class 5011
evConst enumeration member 1272
evNormal enumeration member 1272
evVar enumeration member 1272
EWebModuleException class 6501
about EWebModuleException class 6501
EWebUsersListException class 6743
about EWebUsersListException class 6743
EWin32Error class 4015
about EWin32Error class 4015
EWSDLLoadException class 7218
about EWSDLLoadException class 7218
exceptCommand enumeration member 1629
exceptConnection enumeration member 1629
exceptCursor enumeration member 1629
Exception class 4015

about Exception class 4015
BaseException 4030
CleanUpStackInfoProc 4026
Create 4018
CreateFmt 4020
CreateFmtHelp 4021
CreateHelp 4021
CreateRes 4022
CreateRes constructor 4022
CreateResFmt 4023
CreateResFmt constructor 4022
CreateResFmtHelp 4024
CreateResFmtHelp constructor 4023
CreateResHelp 4025
CreateResHelp constructor 4025
Destroy 4026
Exception fields 4026
Exception Members 4016
Exception methods 4027
Exception properties 4030
GetBaseException 4027
GetExceptionStackInfoProc 4027
GetStackInfoStringProc 4027
GetStackTrace 4028
HelpContext 4030
InnerException 4031
Message 4031
RaiseOuterException 4028
RaisingException 4028
SetInnerException 4028
SetStackInfo 4029
StackInfo 4031
StackTrace 4032
ThrowOuterException 4029
ToString 4029
exceptMetaData enumeration member 1629
exceptUseLast enumeration member 1629
EXMLDocError class 7585
about EXMLDocError class 7585
EXSDateTimeException class 7739
about EXSDateTimeException class 7739
EXSDecimalException class 7740
about EXSDecimalException class 7740
EXSHexBinaryException class 7740
about EXSHexBinaryException class 7740
EXSLPageProducerError class 7822
about EXSLPageProducerError class 7822
EZCompressionError class 7848
about EZCompressionError class 7848
EZDecompressionError class 7848
about EZDecompressionError class 7848
EZeroDivide class 4032
about EZeroDivide class 4032
EZLibError class 7849
about EZLibError class 7849

F

ffCurrency enumeration member 4364
ffExponent enumeration member 4364
ffFixed enumeration member 4364
ffGeneral enumeration member 4364
ffNumber enumeration member 4364
fpLoginRequired enumeration member 5978
fpPublished enumeration member 5978
friend Currency __fastcall operator -(double lhs, const Currency& rhs) 3260
friend Currency __fastcall operator -(int lhs, const Currency& rhs) 3261
friend Currency __fastcall operator *(double lhs, const Currency& rhs) 3261
friend Currency __fastcall operator *(int lhs, const Currency& rhs) 3261
friend Currency __fastcall operator /(double lhs, const Currency& rhs) 3261
friend Currency __fastcall operator /(int lhs, const Currency& rhs) 3262
friend Currency __fastcall operator +(double lhs, const Currency& rhs) 3262
friend Currency __fastcall operator +(int lhs, const Currency& rhs) 3262
friend UnicodeString __fastcall operator +(const char*, const

UnicodeString& rhs) 3429
friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) 3430
ftComp enumeration member 4779
ftCurr enumeration member 4779
ftDouble enumeration member 4779
ftExtended enumeration member 4779
ftSingle enumeration member 4779
fvCurrency enumeration member 4366
fvExtended enumeration member 4366

H

hbsNormal enumeration member 319
hbsRounded enumeration member 319
hoHideAlways enumeration member 6490
hoHideOnFieldNotVisible enumeration member 6490
hoHideOnNoDisplayAccess enumeration member 6490
hoHideOnNoInputAccess enumeration member 6490
hoMode enumeration member 5887
hoOrigValues enumeration member 5887
hoRecordKeys enumeration member 5887
htmlaAnchor enumeration member 1062
htmlaButton enumeration member 1062
htmlaEventImages enumeration member 1062
htmlaImage enumeration member 1062
htmldImage enumeration member 1063
htmldList enumeration member 1063
htmldNone enumeration member 1063
htmldText enumeration member 1063
htmliCheckBox enumeration member 1064
htmliFile enumeration member 1064
htmliNone enumeration member 1064
htmliPasswordInput enumeration member 1064
htmliRadio enumeration member 1064
htmliSelect enumeration member 1064
htmliSelectMultiple enumeration member 1064
htmliTextArea enumeration member 1064
htmliTextInput enumeration member 1064
htmlvmDisplay enumeration member 1063
htmlvmInput enumeration member 1063
htmlvmNone enumeration member 1063
htmlvmToggleOnAccess enumeration member 1063

I

iaBottom enumeration member 2909
iaCenter enumeration member 2909
IActionImageProducer interface 920
  about IActionImageProducer interface 920
  GetAdapterEventImage 922
  GetDisplayStyle 922
  IActionImageProducer Members 921
  IActionImageProducer methods 921
IAdapterAccess interface 923
  about IAdapterAccess interface 923
  HasExecuteAccess 924
  HasModifyAccess 924
  HasViewAccess 924
  IAdapterAccess Members 923
  IAdapterAccess methods 923
IAdapterActionAccess interface 924
  about IAdapterActionAccess interface 924
  HasExecuteAccess 925
  IAdapterActionAccess Members 925
  IAdapterActionAccess methods 925
IAdapterActionAttributes interface 925
  about IAdapterActionAttributes interface 925
  GetVisible 926
  IAdapterActionAttributes Members 926
  IAdapterActionAttributes methods 926
  IAdapterActionAttributes properties 926
  Visible 926
IAdapterConverter interface 1293
  about IAdapterConverter interface 1293
  GetDomConverter 1294
  IAdapterConverter Members 1293
  IAdapterConverter methods 1293
IAdapterDispatcher interface 927
  about IAdapterDispatcher interface 927

EncodeAdapterItemRequestAsFieldValue 928
EncodeAdapterItemRequestAsHREF 928
GetAdapterItemRequestParams 929
IAdapterDispatcher Members 927
IAdapterDispatcher methods 928
IAdapterDispatchParams interface 929
about IAdapterDispatchParams interface 929
GetHandler 931
GetObjectIdentifier 931
GetParams 931
GetRequestIdentifier 931
Handler 932
IAdapterDispatchParams Members 930
IAdapterDispatchParams methods 930
IAdapterDispatchParams properties 932
ObjectIdentifier 932
Params 933
RequestIdentifier 933
IAdapterDisplay interface 6476
about IAdapterDisplay interface 6476
IAdapterEditor interface 5860
about IAdapterEditor interface 5860
CanAddActionClass 5861
CanAddFieldClass 5861
IAdapterEditor Members 5861
IAdapterEditor methods 5861
IAdapterError interface 934
about IAdapterError interface 934
ErrorText 936
Field 936
GetErrorText 934
GetField 935
GetID 935
IAdapterError Members 934
IAdapterError methods 934
IAdapterError properties 935
ID 936
IAdapterErrors interface 937
about IAdapterErrors interface 937
ClearErrors 938
DefineLabel 938
ErrorCount 940
Errors 940
GetError 938
GetErrorCount 939
GetObjectErrors 939
HasObjectErrors 939
IAdapterErrors Members 937
IAdapterErrors methods 937
IAdapterErrors properties 939
IAdapterFieldAccess interface 940
about IAdapterFieldAccess interface 940
HasModifyAccess 941
HasViewAccess 941
IAdapterFieldAccess Members 941
IAdapterFieldAccess methods 941
IAdapterFieldAttributes interface 941
about IAdapterFieldAttributes interface 941
GetRequired 942
GetVisible 942
IAdapterFieldAttributes Members 942
IAdapterFieldAttributes methods 942
IAdapterFieldAttributes properties 943
Required 943
Visible 943
IAdapterHiddenField interface 943
about IAdapterHiddenField interface 943
GetName 944
GetValue 944
IAdapterHiddenField Members 944
IAdapterHiddenField methods 944
IAdapterHiddenField properties 945
Name 945
Value 945
IAdapterHiddenFields interface 945
about IAdapterHiddenFields interface 945
FieldCount 947
Fields 947

GetField 946
GetFieldCount 946
IAdapterHiddenFields Members 946
IAdapterHiddenFields methods 946
IAdapterHiddenFields properties 946
IAdapterItemRequestParams interface 947
about IAdapterItemRequestParams interface 947
FailurePage 950
GetFailurePage 948
GetParamValues 949
GetSuccessPage 949
IAdapterItemRequestParams Members 948
IAdapterItemRequestParams methods 948
IAdapterItemRequestParams properties 950
ParamValues 951
SetFailurePage 949
SetSuccessPage 950
SuccessPage 951
IAdapterMode interface 951
about IAdapterMode interface 951
AdapterMode 953
GetAdapterMode 952
GetNeedAdapterMode 952
IAdapterMode Members 952
IAdapterMode methods 952
IAdapterMode properties 953
NeedAdapterMode 953
RestoreDefaultAdapterMode 953
SetAdapterMode 953
IAdapterNotifyAdapterChange interface 5861
about IAdapterNotifyAdapterChange interface 5861
IAdapterNotifyAdapterChange Members 5862
IAdapterNotifyAdapterChange methods 5862
NotifyAdapterChange 5862
IAdapterRequestHandler interface 954
about IAdapterRequestHandler interface 954
CreateRequestContext 955
HandleRequest 955
IAdapterRequestHandler Members 954
IAdapterRequestHandler methods 954
IAddFieldsEditor interface 6052
about IAddFieldsEditor interface 6052
GetAddAllFieldsItem 6053
GetAddFieldsDlgCaption 6053
GetAddFieldsItem 6053
GetNewFieldPrefix 6054
IAddFieldsEditor Members 6052
IAddFieldsEditor methods 6053
iaLeft enumeration member 317, 2909
IAppServerSOAP interface 1276
about IAppServerSOAP interface 1276
IAppServerSOAP Members 1276
IAppServerSOAP methods 1277
SAS_ApplyUpdates 1277
SAS_DataRequest 1277
SAS_Execute 1278
SAS_GetParams 1279
SAS_GetProviderNames 1279
SAS_GetRecords 1279
SAS_RowRequest 1280
iaRight enumeration member 317, 2909
IAsFormatted interface 955
about IAsFormatted interface 955
AsFormatted 956
IAsFormatted Members 956
IAsFormatted methods 956
iaTop enumeration member 2909
IBinding interface 7029
about IBinding interface 7029
BindingOperations 7033
GetBindingOperations 7031
GetName 7032
GetType_ 7032
IBinding Members 7030
IBinding methods 7031
IBinding properties 7032
Name 7033
SetName 7032

SetType_ 7032
Type_ 7033
IBindingInput interface 7033
about IBindingInput interface 7033
IBindingOperation interface 7034
about IBindingOperation interface 7034
Faults 7037
GetFaults 7036
GetInput 7036
GetName 7036
GetOutput 7036
IBindingOperation Members 7034
IBindingOperation methods 7036
IBindingOperation properties 7037
Input 7037
Name 7038
Output 7038
SetName 7037
IBindingOperations interface 7038
about IBindingOperations interface 7038
Add 7040
BindingOperation 7041
GetBindingOperation 7040
IBindingOperations Members 7038
IBindingOperations methods 7040
IBindingOperations properties 7041
Insert 7041
IBindingOutput interface 7041
about IBindingOutput interface 7041
IBindings interface 7042
about IBindings interface 7042
Add 7044
Binding 7045
GetBinding 7044
IBindings Members 7042
IBindings methods 7043
IBindings properties 7044
Insert 7044
ICheckOrigValue interface 5862
about ICheckOrigValue interface 5862
CheckOrigValue 5863
ICheckOrigValue Members 5863
ICheckOrigValue methods 5863
ICheckValueChange interface 5863
about ICheckValueChange interface 5863
CheckValueChange 5864
ICheckValueChange Members 5864
ICheckValueChange methods 5864
IClearAdapterValues interface 5864
about IClearAdapterValues interface 5864
ClearAdapterValues 5865
IClearAdapterValues Members 5865
IClearAdapterValues methods 5865
IComparable interface 3497
about IComparable interface 3497
CompareTo 3498
IComparable Members 3497
IComparable methods 3497
ICreateActionRequestContext interface 5865
about ICreateActionRequestContext interface 5865
CreateActionRequestContext 5866
ICreateActionRequestContext Members 5865
ICreateActionRequestContext methods 5866
IDataBlock interface 221
about IDataBlock interface 221
BytesReserved 227
Clear 222
GetBytesReserved 223
GetMemory 223
GetSignature 223
GetSize 224
GetStream 224
IDataBlock Members 222
IDataBlock methods 222
IDataBlock properties 227
IgnoreStream 224
InitData 225
Memory 227

Read 225
SetSignature 226
SetSize 226
Signature 228
Size 228
Stream 229
Write 226
IDataBroker interface 2928
about IDataBroker interface 2928
GetProviderNames 2929
IDataBroker Members 2928
IDataBroker methods 2928
IDataBrokerDisp interface 2929
about IDataBrokerDisp interface 2929
GetProviderNames 2930
IDataBrokerDisp Members 2929
IDataBrokerDisp methods 2929
IDataIntercept interface 229
about IDataIntercept interface 229
DataIn 230
DataOut 231
IDataIntercept Members 230
IDataIntercept methods 230
IDefaultPageFileName interface 956
about IDefaultPageFileName interface 956
IDefaultPageFileName Members 957
IDefaultPageFileName methods 957
SetDefaultPageFileName 957
IDefinition interface 7045
about IDefinition interface 7045
Bindings 7050
GetBindings 7047
GetImports 7047
GetMessages 7048
GetName 7048
GetPortTypes 7048
GetServices 7048
GetTargetNamespace 7048
GetTypes 7049
IDefinition Members 7045
IDefinition methods 7047
IDefinition properties 7049
Imports 7050
Messages 7050
Name 7050
PortTypes 7050
Services 7051
SetName 7049
SetTargetNamespace 7049
TargetNamespace 7051
Types 7051
IDispatch interface 3498
about IDispatch interface 3498
GetIDsOfNames 3499
GetTypeInfo 3499
GetTypeInfoCount 3499
IDispatch Members 3498
IDispatch methods 3498
Invoke 3499
IDocumentation interface 7051
about IDocumentation interface 7051
IDocumentationCollection interface 7052
about IDocumentationCollection interface 7052
Add 7054
Documentation 7055
GetDocumentation 7054
IDocumentationCollection Members 7052
IDocumentationCollection methods 7054
IDocumentationCollection properties 7054
Insert 7054
IDocumented interface 7055
about IDocumented interface 7055
Documentation 7057
GetDocumentationCollection 7057
IDocumented Members 7055
IDocumented methods 7056
IDocumented properties 7057
IDOMAttr interface 7473

about IDOMAttr interface 7473
get_name 7475
get_ownerElement 7475
get_specified 7475
get_value 7476
IDOMAttr Members 7474
IDOMAttr methods 7475
IDOMAttr properties 7476
name 7476
ownerElement 7476
set_value 7476
specified 7477
value 7477
IDOMCDATASection interface 7477
about IDOMCDATASection interface 7477
IDOMCharacterData interface 7478
about IDOMCharacterData interface 7478
appendData 7479
data 7481
deleteData 7480
get_data 7480
get_length 7480
IDOMCharacterData Members 7478
IDOMCharacterData methods 7479
IDOMCharacterData properties 7481
insertData 7480
length 7481
replaceData 7480
set_data 7481
substringData 7481
IDOMComment interface 7482
about IDOMComment interface 7482
IDOMDocument interface 7482
about IDOMDocument interface 7482
createAttribute 7484
createAttributeNS 7485
createCDATASection 7485
createComment 7485
createDocumentFragment 7485
createElement 7485
createElementNS 7486
createEntityReference 7486
createProcessingInstruction 7486
createTextNode 7486
doctype 7489
documentElement 7489
domImplementation 7489
get_doctype 7487
get_documentElement 7487
get_domImplementation 7487
getElementById 7487
getElementsByTagName 7488
getElementsByTagNameNS 7488
IDOMDocument Members 7482
IDOMDocument methods 7484
IDOMDocument properties 7488
importNode 7488
set_documentElement 7488
IDOMDocumentFragment interface 7489
about IDOMDocumentFragment interface 7489
IDOMDocumentType interface 7490
about IDOMDocumentType interface 7490
entities 7493
get_entities 7492
get_internalSubset 7492
get_name 7492
get_notations 7492
get_publicId 7492
get_systemId 7493
IDOMDocumentType Members 7490
IDOMDocumentType methods 7491
IDOMDocumentType properties 7493
internalSubset 7493
name 7494
notations 7494
publicId 7494
systemId 7494
IDOMElement interface 7494

about IDOMElement interface 7494
get_tagName 7497
getAttribute 7497
getAttributeNode 7497
getAttributeNodeNS 7497
getAttributeNS 7497
getElementsByTagName 7498
getElementsByTagNameNS 7498
hasAttribute 7498
hasAttributeNS 7498
IDOMElement Members 7495
IDOMElement methods 7496
IDOMElement properties 7500
removeAttribute 7499
removeAttributeNode 7499
removeAttributeNS 7499
setAttribute 7499
setAttributeNode 7499
setAttributeNodeNS 7500
setAttributeNS 7500
tagName 7500
IDOMEntity interface 7501
about IDOMEntity interface 7501
get_notationName 7502
get_publicId 7502
get_systemId 7503
IDOMEntity Members 7501
IDOMEntity methods 7502
IDOMEntity properties 7503
notationName 7503
publicId 7503
systemId 7503
IDOMEntityReference interface 7504
about IDOMEntityReference interface 7504
IDOMHeaderProcessor interface 1179
about IDOMHeaderProcessor interface 1179
CreateHeader 1180
IDOMHeaderProcessor Members 1180
IDOMHeaderProcessor methods 1180
ProcessHeader 1180
IDOMImplementation interface 7504
about IDOMImplementation interface 7504
createDocument 7505
createDocumentType 7505
hasFeature 7505
IDOMImplementation Members 7504
IDOMImplementation methods 7505
IDOMNamedNodeMap interface 7506
about IDOMNamedNodeMap interface 7506
get_item 7507
get_length 7507
getNamedItem 7507
getNamedItemNS 7507
IDOMNamedNodeMap Members 7506
IDOMNamedNodeMap methods 7506
IDOMNamedNodeMap properties 7508
item 7509
length 7509
removeNamedItem 7508
removeNamedItemNS 7508
setNamedItem 7508
setNamedItemNS 7508
IDOMNode interface 7509
about IDOMNode interface 7509
appendChild 7511
attributes 7520
childNodes 7520
cloneNode 7512
firstChild 7520
get_attributes 7512
get_childNodes 7512
get_firstChild 7512
get_lastChild 7513
get_localName 7513
get_namespaceURI 7513
get_nextSibling 7514
get_nodeName 7514
get_nodeType 7514

get_nodeValue 7515
get_ownerDocument 7515
get_parentNode 7515
get_prefix 7515
get_previousSibling 7516
hasChildNodes 7516
IDOMNode Members 7510
IDOMNode methods 7511
IDOMNode properties 7519
insertBefore 7516
lastChild 7520
localName 7521
namespaceURI 7521
nextSibling 7521
nodeName 7521
nodeType 7521
nodeValue 7522
normalize 7517
ownerDocument 7522
parentNode 7522
prefix 7522
previousSibling 7523
removeChild 7518
replaceChild 7518
set_nodeValue 7519
supports 7519
IDOMNodeEx interface 7523
about IDOMNodeEx interface 7523
get_text 7524
get_xml 7525
IDOMNodeEx Members 7523
IDOMNodeEx methods 7524
IDOMNodeEx properties 7526
set_text 7525
text 7526
transformNode 7525, 7526
transformNode method 7525
xml 7526
IDOMNodeList interface 7526
about IDOMNodeList interface 7526
get_item 7527
get_length 7527
IDOMNodeList Members 7527
IDOMNodeList methods 7527
IDOMNodeList properties 7528
item 7528
length 7528
IDOMNodeSelect interface 7528
about IDOMNodeSelect interface 7528
IDOMNodeSelect Members 7529
IDOMNodeSelect methods 7529
selectNode 7529
selectNodes 7529
IDOMNotation interface 7530
about IDOMNotation interface 7530
get_publicId 7531
get_systemId 7531
IDOMNotation Members 7530
IDOMNotation methods 7531
IDOMNotation properties 7532
publicId 7532
systemId 7532
IDOMParseError interface 7532
about IDOMParseError interface 7532
errorCode 7535
filePos 7535
get_errorCode 7533
get_filepos 7534
get_line 7534
get_linepos 7534
get_reason 7534
get_srcText 7534
get_url 7535
IDOMParseError Members 7533
IDOMParseError methods 7533
IDOMParseError properties 7535
line 7536
linePos 7536

reason 7536
srcText 7536
url 7536
IDOMParseOptions interface 7537
about IDOMParseOptions interface 7537
async 7540
get_async 7538
get_preserveWhiteSpace 7538
get_resolveExternals 7538
get_validate 7538
IDOMParseOptions Members 7537
IDOMParseOptions methods 7537
IDOMParseOptions properties 7539
preserveWhiteSpace 7540
resolveExternals 7540
set_async 7539
set_preserveWhiteSpace 7539
set_resolveExternals 7539
set_validate 7539
validate 7540
IDOMPersist interface 7541
about IDOMPersist interface 7541
asyncLoadState 7542
get_xml 7542
IDOMPersist Members 7541
IDOMPersist methods 7541
IDOMPersist properties 7544
load 7542
loadFromStream 7542, 7543
loadFromStream method 7542
loadxml 7543
save 7543
saveToStream 7544
saveToStream method 7543
set_OnAsyncLoad 7544
xml 7544
IDOMProcessingInstruction interface 7545
about IDOMProcessingInstruction interface 7545
data 7547
get_data 7546
get_target 7546
IDOMProcessingInstruction Members 7545
IDOMProcessingInstruction methods 7546
IDOMProcessingInstruction properties 7547
set_data 7547
target 7547
IDOMText interface 7548
about IDOMText interface 7548
IDOMText Members 7548
IDOMText methods 7549
splitText 7549
IDOMXMLProlog interface 7550
about IDOMXMLProlog interface 7550
Encoding 7552
get_Encoding 7551
get_Standalone 7551
get_Version 7551
IDOMXMLProlog Members 7550
IDOMXMLProlog methods 7550
IDOMXMLProlog properties 7552
set_Encoding 7551
set_Standalone 7551
set_Version 7552
Standalone 7552
Version 7552
IDOMXSLProcessor interface 7553
about IDOMXSLProcessor interface 7553
Get_input 7554
Get_output 7554
Get_stylesheet 7554
IDOMXSLProcessor Members 7553
IDOMXSLProcessor methods 7554
IDOMXSLProcessor properties 7556
input 7556
output 7556
reset 7555
Set_input 7555
Set_output 7555

setParameter 7555
stylesheet 7556
transform 7555
IEchoActionFieldValue interface 5866
about IEchoActionFieldValue interface 5866
EchoActionFieldValue 5868
GetEchoActionFieldValue 5867
IEchoActionFieldValue Members 5866
IEchoActionFieldValue methods 5867
IEchoActionFieldValue properties 5867
SetEchoActionFieldValue 5867
IEchoAdapterFieldValues interface 5868
about IEchoAdapterFieldValues interface 5868
EchoActionFieldValues 5869
GetEchoActionFieldValues 5869
IEchoAdapterFieldValues Members 5868
IEchoAdapterFieldValues methods 5868
IEchoAdapterFieldValues properties 5869
SetEchoActionFieldValues 5869
IEnumerable interface 3500
about IEnumerable interface 3500
GetEnumerator 3500
IEnumerable Members 3500
IEnumerable methods 3500
IEnumerator interface 3501
about IEnumerator interface 3501
Current 3502
GetCurrent 3501
IEnumerator Members 3501
IEnumerator methods 3501
IEnumerator properties 3502
MoveNext 3502
Reset 3502
IExecuteValue interface 5869
about IExecuteValue interface 5869
ExecuteValue 5870
IExecuteValue Members 5870
IExecuteValue methods 5870
IFault interface 7057
about IFault interface 7057
IFaults interface 7058
about IFaults interface 7058
Add 7060
Fault 7061
GetFault 7060
IFaults Members 7058
IFaults methods 7060
IFaults properties 7060
Insert 7060
ifDispatch enumeration member 4780
ifDispInterface enumeration member 4780
ifHasGuid enumeration member 4780
IFieldValuesAdapter interface 957
about IFieldValuesAdapter interface 957
GetField 958
GetValueIndex 958
IFieldValuesAdapter Members 958
IFieldValuesAdapter methods 958
IFindIncludeFile interface 958
about IFindIncludeFile interface 958
FindIncludeFile 959
IFindIncludeFile Members 959
IFindIncludeFile methods 959
iftCheckBox enumeration member 6492
iftDefault enumeration member 6492
iftFile enumeration member 6492
iftPasswordInput enumeration member 6492
iftRadio enumeration member 6492
iftSelect enumeration member 6492
iftSelectMultiple enumeration member 6492
iftTextArea enumeration member 6492
iftTextInput enumeration member 6492
IFullyQualifyInputNames interface 5871
about IFullyQualifyInputNames interface 5871
FullyQualifyInputNames 5871
IFullyQualifyInputNames Members 5871
IFullyQualifyInputNames methods 5871
IGetAdapterActions interface 959

about IGetAdapterActions interface 959
GetAdapterActions 960
IGetAdapterActions Members 960
IGetAdapterActions methods 960
IGetAdapterDisplayCharacteristics interface 960
about IGetAdapterDisplayCharacteristics interface 960
GetAdapterDisplayCharacteristics 961
IGetAdapterDisplayCharacteristics Members 961
IGetAdapterDisplayCharacteristics methods 961
IGetAdapterErrors interface 961
about IGetAdapterErrors interface 961
GetAdapterErrors 962
IGetAdapterErrors Members 962
IGetAdapterErrors methods 962
IGetAdapterErrorsList interface 5872
about IGetAdapterErrorsList interface 5872
GetAdapterErrorsList 5872
IGetAdapterErrorsList Members 5872
IGetAdapterErrorsList methods 5872
IGetAdapterFields interface 962
about IGetAdapterFields interface 962
GetAdapterFields 963
IGetAdapterFields Members 963
IGetAdapterFields methods 963
IGetAdapterFormLayout interface 6476
about IGetAdapterFormLayout interface 6476
GetAdapterFormLayout 6477
IGetAdapterFormLayout Members 6477
IGetAdapterFormLayout methods 6477
IGetAdapterHiddenFields interface 963
about IGetAdapterHiddenFields interface 963
GetAdapterHiddenFields 964
GetAdapterHiddenRecordFields 964
IGetAdapterHiddenFields Members 963
IGetAdapterHiddenFields methods 964
IGetAdapterImage interface 964
about IGetAdapterImage interface 964
GetAdapterImage 965
IGetAdapterImage Members 965
IGetAdapterImage methods 965
IGetAdapterItemRequestParams interface 965
about IGetAdapterItemRequestParams interface 965
GetAdapterItemRequestParams 966
IGetAdapterItemRequestParams Members 966
IGetAdapterItemRequestParams methods 966
IGetAdapterModeNames interface 5873
about IGetAdapterModeNames interface 5873
GetAdapterModeNames 5873
IGetAdapterModeNames Members 5873
IGetAdapterModeNames methods 5873
IGetAdapterModeOfDisplay interface 6477
about IGetAdapterModeOfDisplay interface 6477
GetAdapterModeOfAdapter 6478
GetAdapterModeOfDisplay 6478
IGetAdapterModeOfDisplay Members 6478
IGetAdapterModeOfDisplay methods 6478
IGetAdapterOfDisplay interface 6478
about IGetAdapterOfDisplay interface 6478
GetAdapter 6479
IGetAdapterOfDisplay Members 6479
IGetAdapterOfDisplay methods 6479
IGetAdapterRequestDefaultResponse interface 966
about IGetAdapterRequestDefaultResponse interface 966
FailurePage 968
GetFailurePage 967
GetSuccessPage 967
IGetAdapterRequestDefaultResponse Members 967
IGetAdapterRequestDefaultResponse methods 967
IGetAdapterRequestDefaultResponse properties 968
SuccessPage 968
IGetAdapterValuesList interface 968
about IGetAdapterValuesList interface 968
GetAdapterValuesList 969
IGetAdapterValuesList Members 969
IGetAdapterValuesList methods 969
IGetAdapterVariableName interface 6479
about IGetAdapterVariableName interface 6479
AdapterVariableName 6481

GetAdapterVariableName 6480
IGetAdapterVariableName Members 6480
IGetAdapterVariableName methods 6480
IGetAdapterVariableName properties 6480
IGetControlRequirements interface 6481
about IGetControlRequirements interface 6481
GetControlRequirements 6482
IGetControlRequirements Members 6481
IGetControlRequirements methods 6481
IGetDefaultAction interface 969
about IGetDefaultAction interface 969
DefaultAction 970
GetDefaultAction 970
IGetDefaultAction Members 969
IGetDefaultAction methods 970
IGetDefaultAction properties 970
IGetDesigntimeWarnings interface 971
about IGetDesigntimeWarnings interface 971
GetDesigntimeWarnings 971
IGetDesigntimeWarnings Members 971
IGetDesigntimeWarnings methods 971
IGetFieldValuesAdapter interface 972
about IGetFieldValuesAdapter interface 972
GetFieldValuesAdapter 972
IGetFieldValuesAdapter Members 972
IGetFieldValuesAdapter methods 972
IGetFormRequirements interface 6482
about IGetFormRequirements interface 6482
GetFormRequirements 6483
IGetFormRequirements Members 6482
IGetFormRequirements methods 6482
IGetHTMLStyle interface 973
about IGetHTMLStyle interface 973
GetDisplayStyle 974
GetInputStyle 974
GetViewMode 975
IGetHTMLStyle Members 973
IGetHTMLStyle methods 973
IGetProducerComponent interface 976
about IGetProducerComponent interface 976
GetProducerComponent 976
IGetProducerComponent Members 976
IGetProducerComponent methods 976
IGetRecordKeys interface 5874
about IGetRecordKeys interface 5874
GetRecordKeys 5874
IGetRecordKeys Members 5874
IGetRecordKeys methods 5874
IGetScriptObject interface 977
about IGetScriptObject interface 977
GetScriptObject 977
IGetScriptObject Members 977
IGetScriptObject methods 977
IGetScriptObjectClassKey interface 978
about IGetScriptObjectClassKey interface 978
GetScriptObjectClassKey 978
IGetScriptObjectClassKey Members 978
IGetScriptObjectClassKey methods 978
IGetVariablesContainerOfDisplay interface 6483
about IGetVariablesContainerOfDisplay interface 6483
GetVariablesContainer 6484
IGetVariablesContainerOfDisplay Members 6483
IGetVariablesContainerOfDisplay methods 6483
IGetWebAppComponents interface 979
about IGetWebAppComponents interface 979
GetAdapterDispatcher 980
GetLocateFileService 980
GetPageDispatcher 980
GetWebApplicationInfo 980
GetWebEndUser 980
GetWebUserList 981
IGetWebAppComponents Members 979
IGetWebAppComponents methods 979
IGetWebAppComponents properties 981
WebApplicationInfo 981
WebEndUser 981
IGetWebComponentList interface 6054
about IGetWebComponentList interface 6054

GetComponentList 6055
GetDefaultComponentList 6055
IGetWebComponentList Members 6054
IGetWebComponentList methods 6054
IGetWebComponentNodeName interface 6055
about IGetWebComponentNodeName interface 6055
GetWebComponentNodeName 6056
IGetWebComponentNodeName Members 6055
IGetWebComponentNodeName methods 6056
IGetXMLStream interface 6056
about IGetXMLStream interface 6056
GetXMLStream 6057
IGetXMLStream Members 6056
IGetXMLStream methods 6057
IHeaderInfo interface 7199
about IHeaderInfo interface 7199
EncodingStyle 7203
GetEncodingStyle 7200
GetName 7200
GetNamespace 7201
GetPart 7201
GetRequired 7201
GetUse 7201
IHeaderInfo Members 7199
IHeaderInfo methods 7200
IHeaderInfo properties 7203
Name 7203
Namespace 7203
Part 7204
Required 7204
SetEncodingStyle 7201
SetName 7202
SetNamespace 7202
SetPart 7202
SetRequired 7202
SetUse 7203
Use 7204
IHTTPReqResp interface 1234
about IHTTPReqResp interface 1234
GetHTTPReqResp 1235
IHTTPReqResp Members 1234
IHTTPReqResp methods 1234
IHTTPSoapDispatch interface 1200
about IHTTPSoapDispatch interface 1200
DispatchSOAP 1201
IHTTPSoapDispatch Members 1200
IHTTPSoapDispatch methods 1201
iiaCenter enumeration member 318
iiaLeft enumeration member 318
iiaRight enumeration member 318
IIDEFileManager interface 982
about IIDEFileManager interface 982
IIDEFileManager Members 982
IIDEFileManager methods 982
SearchFile 982
IIdentifyAdapter interface 5875
about IIdentifyAdapter interface 5875
IImageProducer interface 983
about IImageProducer interface 983
GetAdapterImage 984
IImageProducer Members 983
IImageProducer methods 984
IImport interface 7061
about IImport interface 7061
GetLocation 7063
GetNamespace 7063
IImport Members 7061
IImport methods 7063
IImport properties 7064
Location 7064
Namespace 7064
SetLocation 7063
SetNamespace 7063
IImports interface 7064
about IImports interface 7064
Add 7066
GetImport 7067
IImports Members 7065

IImports methods 7066
IImports properties 7067
Import 7067
Insert 7067
IIntegerTest interface 1268
about IIntegerTest interface 1268
Add 1269
Doubler 1269
IIntegerTest Members 1268
IIntegerTest methods 1268
IInterface interface 3503
_AddRef 3504
_Release 3504
about IInterface interface 3503
IInterface Members 3503
IInterface methods 3504
QueryInterface 3505
IInternetEnvOptions interface 984
about IInternetEnvOptions interface 984
DisableDebugger 986
GetDisableDebugging 985
GetHTMLExt 985
GetSampleImageFile 985
GetScriptTimeoutSecs 986
HTMLExt 986
IInternetEnvOptions Members 984
IInternetEnvOptions methods 985
IInternetEnvOptions properties 986
SampleImageFile 986
ScriptTimeoutSecs 987
IInternetExecuteScript interface 987
about IInternetExecuteScript interface 987
CanExecuteScript 988
ExecuteScript 988
IInternetExecuteScript Members 987
IInternetExecuteScript methods 987
IInternetFileTypes interface 988
about IInternetFileTypes interface 988
FindFileExt 989
IInternetFileTypes Members 988
IInternetFileTypes methods 989
IInternetScriptDebugger interface 989
about IInternetScriptDebugger interface 989
CanDebugScript 990
DebugScript 990
IInternetScriptDebugger Members 989
IInternetScriptDebugger methods 990
IInvokable interface 3505
about IInvokable interface 3505
IInvokerConverter interface 1294
about IInvokerConverter interface 1294
GetInvoker 1295
IInvokerConverter Members 1294
IInvokerConverter methods 1294
IIsAdapterActionList interface 990
about IIsAdapterActionList interface 990
GetIsAdapterActionList 991
IIsAdapterActionList Members 991
IIsAdapterActionList methods 991
IIsAdapterActionList properties 991
IsAdapterActionList 991
IIterateIntfSupport interface 992
about IIterateIntfSupport interface 992
GetCurrentIntf 992
IIterateIntfSupport Members 992
IIterateIntfSupport methods 992
IIterateObjectSupport interface 993
about IIterateObjectSupport interface 993
GetCurrentObject 993
IIterateObjectSupport Members 993
IIterateObjectSupport methods 993
IIteratorIndex interface 994
about IIteratorIndex interface 994
GetIteratorIndex 995
IIteratorIndex Members 994
IIteratorIndex methods 994
IIteratorIndex properties 995
InIterator 995

IteratorIndex 995
IIteratorSupport interface 996
about IIteratorSupport interface 996
EndIterator 997
IIteratorSupport Members 997
IIteratorSupport methods 997
NextIteration 997
StartIterator 998
ILayoutWebContent interface 6057
about ILayoutWebContent interface 6057
ILayoutWebContent Members 6058
ILayoutWebContent methods 6058
LayoutButton 6058
LayoutField 6058
LayoutLabelAndField 6058
LayoutTable 6059
IMessage interface 7067
about IMessage interface 7067
GetName 7069
GetParts 7070
IMessage Members 7068
IMessage methods 7069
IMessage properties 7070
Name 7070
Parts 7070
SetName 7070
IMessages interface 7071
about IMessages interface 7071
Add 7073
GetMessage 7073
IMessages Members 7071
IMessages methods 7073
IMessages properties 7073
Insert 7073
Message 7074
IMimeAttachmentHandler interface 1116
about IMimeAttachmentHandler interface 1116
AddSoapHeader 1117
CreateMimeStream 1118
FinalizeStream 1118
GetMIMEBoundary 1119
GetMIMEStream 1119
GetOnGetAttachmentEvent 1120
GetOnSendAttachmentEvent 1120
IMimeAttachmentHandler events 1123
IMimeAttachmentHandler Members 1116
IMimeAttachmentHandler methods 1117
IMimeAttachmentHandler properties 1122
MIMEBoundary 1122
OnGetAttachment 1123
OnSendAttachment 1123
ProcessMultiPartForm 1120
SetMIMEBoundary 1121
SetOnGetAttachmentEvent 1121
SetOnSendAttachmentEvent 1122
incFile enumeration member 6687
incPage enumeration member 6687
INotifyActivateModule interface 6059
about INotifyActivateModule interface 6059
INotifyActivateModule Members 6060
INotifyActivateModule methods 6060
NotifyActivateModule 6060
NotifyDeactivateModule 6060
INotifyAdapterChange interface 998
about INotifyAdapterChange interface 998
INotifyAdapterChange Members 998
INotifyAdapterChange methods 999
NotifyAdapterChange 999
INotifyConnectionChange interface 7289
about INotifyConnectionChange interface 7289
ConnectionChange 7290
INotifyConnectionChange Members 7289
INotifyConnectionChange methods 7289
INotifyList interface 999
about INotifyList interface 999
AddNotify 1000
INotifyList Members 1000
INotifyList methods 1000

RemoveNotify 1000
INotifyWebActivate interface 6060
about INotifyWebActivate interface 6060
INotifyWebActivate Members 6061
INotifyWebActivate methods 6061
NotifyActivate 6061
NotifyDeactivate 6061
ioBeforeLoad enumeration member 7213
IObjectGetter interface 1269
about IObjectGetter interface 1269
GetAsObject 1270
IObjectGetter Members 1269
IObjectGetter methods 1270
ioLoaded enumeration member 7213
IOperation interface 7074
about IOperation interface 7074
Faults 7077
GetFaults 7076
GetInput 7076
GetName 7076
GetOutput 7077
Input 7077
IOperation Members 7074
IOperation methods 7076
IOperation properties 7077
Name 7078
Output 7078
SetName 7077
IOperations interface 7078
about IOperations interface 7078
Add 7080, 7081
Add method 7080
GetOperation 7081
Insert 7081
IOperations Members 7078
IOperations methods 7080
IOperations properties 7081
Operation 7081
IPageDispatcher interface 1001
about IPageDispatcher interface 1001
DefaultPageName 1005
DispatchPageName 1002
GetCanView 1003
GetDefaultPageName 1003
GetLoginRequired 1003
IPageDispatcher Members 1001
IPageDispatcher methods 1002
IPageDispatcher properties 1005
PageNameOfRequest 1004
RedirectToPageName 1004
IPageResult interface 1006
about IPageResult interface 1006
DispatchPage 1006
IncludePage 1007
IPageResult Members 1006
IPageResult methods 1006
RedirectToPage 1007
IParam interface 7082
about IParam interface 7082
GetMessage 7084
GetName 7084
IParam Members 7082
IParam methods 7083
IParam properties 7085
Message 7085
Name 7085
SetMessage 7084
SetName 7084
IPart interface 7085
about IPart interface 7085
Element 7089
GetElement 7087
GetName 7088
GetType_ 7088
IPart Members 7086
IPart methods 7087
IPart properties 7089
Name 7089

SetElement 7088
SetName 7088
SetType_ 7088
Type_ 7089
IParts interface 7090
about IParts interface 7090
Add 7092
GetPart 7092
Insert 7092
IParts Members 7090
IParts methods 7091
IParts properties 7092
Part 7093
IPort interface 7093
about IPort interface 7093
Binding 7096
GetBinding 7095
GetName 7095
IPort Members 7093
IPort methods 7095
IPort properties 7096
Name 7096
SetBinding 7095
SetName 7095
IPorts interface 7096
about IPorts interface 7096
Add 7098
GetPort 7099
Insert 7099
IPorts Members 7097
IPorts methods 7098
IPorts properties 7099
Port 7099
IPortType interface 7099
about IPortType interface 7099
GetName 7101
GetOperations 7102
IPortType Members 7100
IPortType methods 7101
IPortType properties 7102
Name 7102
Operations 7102
SetName 7102
IPortTypes interface 7103
about IPortTypes interface 7103
Add 7105
GetPortType 7105
Insert 7105
IPortTypes Members 7103
IPortTypes methods 7105
IPortTypes properties 7105
PortType 7106
IProducerEditorViewSupport interface 6062
about IProducerEditorViewSupport interface 6062
GetTemplateFileType 6063
GetXMLData 6063
GetXSLData 6063
HasHTMLBrowserView 6063
HasScriptView 6063
HasXMLBrowserView 6064
HasXSLBrowserView 6064
IProducerEditorViewSupport Members 6062
IProducerEditorViewSupport methods 6062
IProvider interface 2930
about IProvider interface 2930
ApplyUpdates 2931
Constraints 2933
Data 2933
DataRequest 2931
Get_Constraints 2931
Get_Data 2932
GetMetaData 2932
GetRecords 2932
IProvider Members 2930
IProvider methods 2931
IProvider properties 2933
Reset 2932
Set_Constraints 2933

SetParams 2933
IProviderDisp interface 2934
about IProviderDisp interface 2934
ApplyUpdates 2935
Constraints 2936
Data 2936
DataRequest 2935
GetMetaData 2935
GetRecords 2935
IProviderDisp Members 2934
IProviderDisp methods 2934
IProviderDisp properties 2936
Reset 2936
SetParams 2936
IQualifiedName interface 7204
about IQualifiedName interface 7204
GetName 7205
GetNamespace 7206
IQualifiedName Members 7205
IQualifiedName methods 7205
IQualifiedName properties 7207
Name 7207
Namespace 7207
SetName 7206
SetNamespace 7206
IReadWriteSync interface 4101
about IReadWriteSync interface 4101
BeginRead 4102
BeginWrite 4102
EndRead 4102
EndWrite 4102
IReadWriteSync Members 4101
IReadWriteSync methods 4101
IScreenTipItem interface 315
about IScreenTipItem interface 315
GetScreenTipItem 316
IScreenTipItem Members 315
IScreenTipItem methods 315
IScriptErrorList interface 590
about IScriptErrorList interface 590
advanceError 591
canAdvanceError 591
canRetreatError 591
getAlwaysShowLockState 592
getDetailsPaneOpen 592
getErrorChar 592
getErrorCode 592
getErrorLine 593
getErrorMsg 593
getErrorUrl 593
getPerErrorDisplay 593
IScriptErrorList Members 590
IScriptErrorList methods 591
retreatError 593
setDetailsPaneOpen 594
setPerErrorDisplay 594
IScriptErrorListDisp interface 594
about IScriptErrorListDisp interface 594
advanceError 595
canAdvanceError 596
canRetreatError 596
getAlwaysShowLockState 596
getDetailsPaneOpen 596
getErrorChar 596
getErrorCode 597
getErrorLine 597
getErrorMsg 597
getErrorUrl 597
getPerErrorDisplay 598
IScriptErrorListDisp Members 594
IScriptErrorListDisp methods 595
retreatError 598
setDetailsPaneOpen 598
setPerErrorDisplay 598
ISearch interface 598
about ISearch interface 598
Get_Id 599
Get_Title 600

Get_URL 600
Id 600
ISearch Members 599
ISearch methods 599
ISearch properties 600
Title 600
URL 601
ISearchAssistantOC interface 601
about ISearchAssistantOC interface 601
AddNextMenuItem 603
ASProvider 609
ASSetting 609
EncodeString 603
EventHandled 610
FindComputer 603
FindFilesOrFolders 604
FindOnWeb 604
FindPeople 604
FindPrinter 604
Get_ASProvider 604
Get_ASSetting 605
Get_InWebFolder 605
Get_SearchAssistantDefault 605
Get_Searches 605
Get_ShellFeaturesEnabled 606
Get_Version 606
GetProperty 606
GetSearchAssistantURL 606
InWebFolder 610
ISearchAssistantOC Members 601
ISearchAssistantOC methods 602
ISearchAssistantOC properties 609
IsRestricted 606
NavigateToDefaultSearch 607
NETDetectNextNavigate 607
NotifySearchSettingsChanged 607
PutFindText 607
PutProperty 608
ResetNextMenu 608
SearchAssistantDefault 610
Searches 610
Set_ASProvider 608
Set_ASSetting 608
Set_EventHandled 608
SetDefaultSearchUrl 609
ShellFeaturesEnabled 611
Version 611
ISearchAssistantOC2 interface 611
about ISearchAssistantOC2 interface 611
Get_ShowFindPrinter 613
ISearchAssistantOC2 Members 611
ISearchAssistantOC2 methods 612
ISearchAssistantOC2 properties 613
ShowFindPrinter 613
ISearchAssistantOC2Disp interface 613
about ISearchAssistantOC2Disp interface 613
AddNextMenuItem 615
ASProvider 619
ASSetting 619
EncodeString 615
EventHandled 619
FindComputer 615
FindFilesOrFolders 615
FindOnWeb 616
FindPeople 616
FindPrinter 616
GetProperty 616
GetSearchAssistantURL 617
InWebFolder 620
ISearchAssistantOC2Disp Members 614
ISearchAssistantOC2Disp methods 614
ISearchAssistantOC2Disp properties 619
IsRestricted 617
NavigateToDefaultSearch 617
NETDetectNextNavigate 617
NotifySearchSettingsChanged 617
PutFindText 618
PutProperty 618

ResetNextMenu 618
SearchAssistantDefault 620
Searches 620
SetDefaultSearchUrl 618
ShellFeaturesEnabled 620
ShowFindPrinter 620
Version 621
ISearchAssistantOC3 interface 621
about ISearchAssistantOC3 interface 621
Get_SearchCompanionAvailable 623
Get_UseSearchCompanion 623
ISearchAssistantOC3 Members 621
ISearchAssistantOC3 methods 622
ISearchAssistantOC3 properties 623
SearchCompanionAvailable 623
Set_UseSearchCompanion 623
UseSearchCompanion 624
ISearchAssistantOC3Disp interface 624
about ISearchAssistantOC3Disp interface 624
AddNextMenuItem 625
ASProvider 630
ASSetting 630
EncodeString 626
EventHandled 630
FindComputer 626
FindFilesOrFolders 626
FindOnWeb 626
FindPeople 627
FindPrinter 627
GetProperty 627
GetSearchAssistantURL 627
InWebFolder 630
ISearchAssistantOC3Disp Members 624
ISearchAssistantOC3Disp methods 625
ISearchAssistantOC3Disp properties 629
IsRestricted 627
NavigateToDefaultSearch 628
NETDetectNextNavigate 628
NotifySearchSettingsChanged 628
PutFindText 628
PutProperty 629
ResetNextMenu 629
SearchAssistantDefault 630
SearchCompanionAvailable 631
Searches 631
SetDefaultSearchUrl 629
ShellFeaturesEnabled 631
ShowFindPrinter 631
UseSearchCompanion 632
Version 632
ISearchAssistantOCDisp interface 632
about ISearchAssistantOCDisp interface 632
AddNextMenuItem 634
ASProvider 638
ASSetting 638
EncodeString 634
EventHandled 638
FindComputer 634
FindFilesOrFolders 634
FindOnWeb 634
FindPeople 635
FindPrinter 635
GetProperty 635
GetSearchAssistantURL 635
InWebFolder 638
ISearchAssistantOCDisp Members 632
ISearchAssistantOCDisp methods 633
ISearchAssistantOCDisp properties 637
IsRestricted 636
NavigateToDefaultSearch 636
NETDetectNextNavigate 636
NotifySearchSettingsChanged 636
PutFindText 636
PutProperty 637
ResetNextMenu 637
SearchAssistantDefault 638
Searches 639
SetDefaultSearchUrl 637

ShellFeaturesEnabled 639
Version 639
ISearchDisp interface 639
about ISearchDisp interface 639
Id 640
ISearchDisp Members 640
ISearchDisp properties 640
Title 640
URL 641
ISearches interface 641
_NewEnum 642
about ISearches interface 641
Count 643
Default 643
Get_Count 642
Get_Default 642
ISearches Members 641
ISearches methods 642
ISearches properties 643
Item 642
ISearchesDisp interface 643
_NewEnum 644
about ISearchesDisp interface 643
Count 645
Default 645
ISearchesDisp Members 644
ISearchesDisp methods 644
ISearchesDisp properties 645
Item 644
ISendDataBlock interface 231
about ISendDataBlock interface 231
ISendDataBlock Members 232
ISendDataBlock methods 232
Send 232
IService interface 7106
about IService interface 7106
GetName 7108
GetPorts 7108
IService Members 7106
IService methods 7108
IService properties 7108
Name 7109
Ports 7109
SetName 7108
IServices interface 7109
about IServices interface 7109
Add 7111
GetService 7111
Insert 7112
IServices Members 7109
IServices methods 7111
IServices properties 7112
Service 7112
ISessionsService interface 1007
about ISessionsService interface 1007
CheckStatusChange 1008
Clear 1008
ISessionsService Members 1008
ISessionsService methods 1008
NewSession 1009
NotifyStatusChange 1009
ISessionsServiceConnectID interface 1010
about ISessionsServiceConnectID interface 1010
Connect 1010
ConnectActive 1011
ISessionsServiceConnectID Members 1010
ISessionsServiceConnectID methods 1010
ISessionsServicePersist interface 1011
about ISessionsServicePersist interface 1011
ISessionsServicePersist Members 1011
ISessionsServicePersist methods 1011
LoadFromFile 1012
LoadFromStream 1012
SaveToFile 1012
SaveToStream 1012
ISessionsServiceStoreID interface 1013
about ISessionsServiceStoreID interface 1013
ISessionsServiceStoreID Members 1013

ISessionsServiceStoreID methods 1013
ReadSessionID 1013
WriteSessionID 1014
ISetSubComponents interface 5875
about ISetSubComponents interface 5875
ISetSubComponents Members 5875
ISetSubComponents methods 5876
SetActions 5876
SetData 5876
ISetWebContentOptions interface 6064
about ISetWebContentOptions interface 6064
ISetWebContentOptions Members 6064
ISetWebContentOptions methods 6065
SetWebContentOptions 6065
ISetWebFieldValue interface 5876
about ISetWebFieldValue interface 5876
ISetWebFieldValue Members 5877
ISetWebFieldValue methods 5877
SetValue 5877
IShellFavoritesNameSpace interface 645
about IShellFavoritesNameSpace interface 645
CreateSubscriptionForSelection 647
DeleteSubscriptionForSelection 647
Export 647
Get_SubscriptionsEnabled 647
Import 647
InvokeContextMenuCommand 648
IShellFavoritesNameSpace Members 646
IShellFavoritesNameSpace methods 646
IShellFavoritesNameSpace properties 649
MoveSelectionDown 648
MoveSelectionTo 648
MoveSelectionUp 648
NewFolder 649
ResetSort 649
SetRoot 649
SubscriptionsEnabled 650
Synchronize 649
IShellFavoritesNameSpaceDisp interface 650
about IShellFavoritesNameSpaceDisp interface 650
CreateSubscriptionForSelection 651
DeleteSubscriptionForSelection 651
Export 652
Import 652
InvokeContextMenuCommand 652
IShellFavoritesNameSpaceDisp Members 650
IShellFavoritesNameSpaceDisp methods 651
IShellFavoritesNameSpaceDisp properties 654
MoveSelectionDown 652
MoveSelectionTo 653
MoveSelectionUp 653
NewFolder 653
ResetSort 653
SetRoot 653
SubscriptionsEnabled 654
Synchronize 654
IShellNameSpace interface 654
about IShellNameSpace interface 654
Columns 661
CountViewTypes 662
Depth 662
EnumOptions 662
Expand 656
Flags 662
Get_Columns 657
Get_CountViewTypes 657
Get_Depth 657
Get_EnumOptions 657
Get_Flags 657
Get_Mode 658
Get_Root 658
Get_SelectedItem 658
Get_TVFlags 658
IShellNameSpace Members 655
IShellNameSpace methods 656
IShellNameSpace properties 661
Mode 662
Root 663

SelectedItem 663
SelectedItems 659
Set_Columns 659
Set_Depth 659
Set_EnumOptions 659
Set_Flags 659
Set_Mode 660
Set_Root 660
Set_SelectedItem 660
Set_TVFlags 660
SetViewType 661
TVFlags 663
UnselectAll 661
IShellNameSpaceDisp interface 663
about IShellNameSpaceDisp interface 663
Columns 669
CountViewTypes 669
CreateSubscriptionForSelection 665
DeleteSubscriptionForSelection 665
Depth 669
EnumOptions 669
Expand 665
Export 665
Flags 669
Import 666
InvokeContextMenuCommand 666
IShellNameSpaceDisp Members 664
IShellNameSpaceDisp methods 664
IShellNameSpaceDisp properties 668
Mode 670
MoveSelectionDown 666
MoveSelectionTo 666
MoveSelectionUp 667
NewFolder 667
ResetSort 667
Root 670
SelectedItem 670
SelectedItems 667
SetRoot 667
SetViewType 668
SubscriptionsEnabled 670
Synchronize 668
TVFlags 671
UnselectAll 668
IShellUIHelper interface 671
about IShellUIHelper interface 671
AddChannel 672
AddDesktopComponent 672
AddFavorite 672
AutoCompleteAttach 673
AutoCompleteSaveForm 673
AutoScan 673
ImportExportFavorites 673
IShellUIHelper Members 671
IShellUIHelper methods 672
IsSubscribed 674
NavigateAndFind 674
RefreshOfflineDesktop 674
ResetFirstBootMode 674
ResetSafeMode 674
ShowBrowserUI 675
IShellUIHelperDisp interface 675
about IShellUIHelperDisp interface 675
AddChannel 676
AddDesktopComponent 676
AddFavorite 677
AutoCompleteAttach 677
AutoCompleteSaveForm 677
AutoScan 677
ImportExportFavorites 677
IShellUIHelperDisp Members 675
IShellUIHelperDisp methods 676
IsSubscribed 678
NavigateAndFind 678
RefreshOfflineDesktop 678
ResetFirstBootMode 678
ResetSafeMode 679
ShowBrowserUI 679

IShellWindows interface 679
_NewEnum 680
about IShellWindows interface 679
Count 683
FindWindowSW 680
Get_Count 681
IShellWindows Members 679
IShellWindows methods 680
IShellWindows properties 683
Item 681
OnActivated 681
OnCreated 681
OnNavigate 682
ProcessAttachDetach 682
Register 682
RegisterPending 682
Revoke 682
IShellWindowsDisp interface 683
_NewEnum 684
about IShellWindowsDisp interface 683
Count 687
FindWindowSW 684
IShellWindowsDisp Members 683
IShellWindowsDisp methods 684
IShellWindowsDisp properties 686
Item 685
OnActivated 685
OnCreated 685
OnNavigate 685
ProcessAttachDetach 686
Register 686
RegisterPending 686
Revoke 686
IStrings interface 2937
_NewEnum 2938
about IStrings interface 2937
Add 2938
Clear 2938
ControlDefault 2940
Count 2938
Get_ControlDefault 2939
Get_Item 2939
IStrings Members 2937
IStrings methods 2937
IStrings properties 2940
Item 2940
Remove 2939
Set_ControlDefault 2939
Set_Item 2940
IStringsDisp interface 2940
_NewEnum 2941
about IStringsDisp interface 2940
Add 2942
Clear 2942
ControlDefault 2943
Count 2942
IStringsDisp Members 2941
IStringsDisp methods 2941
IStringsDisp properties 2942
Item 2943
Remove 2942
IStringTest interface 1270
about IStringTest interface 1270
Append 1271
IStringTest Members 1270
IStringTest methods 1271
UpCase 1271
itEnd enumeration member 5887
itNext enumeration member 5887
ITopLevelWebComponent interface 6065
about ITopLevelWebComponent interface 6065
ITransport interface 233
about ITransport interface 233
Connected 236
GetConnected 234
GetWaitEvent 234
ITransport Members 233
ITransport methods 234

ITransport properties 236
Receive 235
Send 235
SetConnected 235
itStart enumeration member 5887
ITypes interface 7112
about ITypes interface 7112
GetSchemaDefs 7114
ITypes Members 7113
ITypes methods 7114
ITypes properties 7114
SchemaDefs 7114
IUpdateValue interface 5877
about IUpdateValue interface 5877
IUpdateValue Members 5877
IUpdateValue methods 5878
UpdateValue 5878
IValidateValue interface 5878
about IValidateValue interface 5878
IValidateValue Members 5878
IValidateValue methods 5879
ValidateValue 5879
IValuesListAdapter interface 1014
about IValuesListAdapter interface 1014
GetListImage 1015
GetListImageOfValue 1016
GetListName 1016
GetListNameOfValue 1016
GetListValue 1017
IValuesListAdapter Members 1015
IValuesListAdapter methods 1015
IVarInstanceReference interface 5045
about IVarInstanceReference interface 5045
GetInstance 5046
IVarInstanceReference Members 5045
IVarInstanceReference methods 5045
IVarInvokeable interface 5046
about IVarInvokeable interface 5046
DoFunction 5047
DoProcedure 5047
GetProperty 5047
IVarInvokeable Members 5046
IVarInvokeable methods 5046
SetProperty 5047
IWebActionsList interface 1017
about IWebActionsList interface 1017
FindAction 1018
GetActionsList 1018
GetVisibleActions 1019
IsActionInUse 1019
IWebActionsList Members 1018
IWebActionsList methods 1018
IWebActionsList properties 1019
VisibleActions 1019
IWebActionsSubcomponent interface 5879
about IWebActionsSubcomponent interface 5879
IWebApplicationInfo interface 1019
about IWebApplicationInfo interface 1019
ApplicationAdapter 1021
ApplicationTitle 1022
GetApplicationAdapter 1021
GetApplicationTitle 1021
IWebApplicationInfo Members 1020
IWebApplicationInfo methods 1020
IWebApplicationInfo properties 1021
IWebBrowser interface 687
about IWebBrowser interface 687
Application 695
Busy 695
Container 695
Document 695
Get_Application 689
Get_Busy 689
Get_Container 689
Get_Document 689
Get_Height 690
Get_Left 690
Get_LocationName 690

Get_LocationURL 690
Get_Parent 691
Get_Top 691
Get_TopLevelContainer 691
Get_type_ 691
Get_Width 691
GoBack 692
GoForward 692
GoHome 692
GoSearch 692
Height 696
IWebBrowser Members 687
IWebBrowser methods 688
IWebBrowser properties 694
Left 696
LocationName 696
LocationURL 696
Navigate 693
Parent 696
Refresh 693
Refresh2 693
Set_Height 693
Set_Left 693
Set_Top 694
Set_Width 694
Stop 694
Top 697
TopLevelContainer 697
type_ 697
Width 697
IWebBrowser2 interface 698
about IWebBrowser2 interface 698
AddressBar 705
ExecWB 700
Get_AddressBar 701
Get_Offline 701
Get_ReadyState 701
Get_RegisterAsBrowser 701
Get_RegisterAsDropTarget 702
Get_Resizable 702
Get_Silent 702
Get_TheaterMode 702
IWebBrowser2 Members 698
IWebBrowser2 methods 700
IWebBrowser2 properties 705
Navigate2 702
Offline 705
QueryStatusWB 703
ReadyState 705
RegisterAsBrowser 706
RegisterAsDropTarget 706
Resizable 706
Set_AddressBar 703
Set_Offline 703
Set_RegisterAsBrowser 703
Set_RegisterAsDropTarget 704
Set_Resizable 704
Set_Silent 704
Set_TheaterMode 704
ShowBrowserBar 704
Silent 706
TheaterMode 707
IWebBrowser2Disp interface 707
about IWebBrowser2Disp interface 707
AddressBar 713
Application 713
Busy 713
ClientToWindow 709
Container 714
Document 714
ExecWB 709
FullName 714
FullScreen 714
GetProperty 709
GoBack 709
GoForward 710
GoHome 710
GoSearch 710

Height 714
HWnd 715
IWebBrowser2Disp Members 707
IWebBrowser2Disp methods 708
IWebBrowser2Disp properties 712
Left 715
LocationName 715
LocationURL 715
MenuBar 716
Name 716
Navigate 710
Navigate2 710
Offline 716
Parent 716
Path 716
PutProperty 711
QueryStatusWB 711
Quit 711
ReadyState 717
Refresh 711
Refresh2 712
RegisterAsBrowser 717
RegisterAsDropTarget 717
Resizable 717
ShowBrowserBar 712
Silent 718
StatusBar 718
StatusText 718
Stop 712
TheaterMode 718
ToolBar 718
Top 719
TopLevelContainer 719
type_ 719
Visible 719
Width 720
IWebBrowserApp interface 720
about IWebBrowserApp interface 720
ClientToWindow 722
FullName 727
FullScreen 727
Get_FullName 722
Get_FullScreen 723
Get_HWnd 723
Get_MenuBar 723
Get_Name 723
Get_Path 724
Get_StatusBar 724
Get_StatusText 724
Get_ToolBar 724
Get_Visible 724
GetProperty 725
HWnd 727
IWebBrowserApp Members 720
IWebBrowserApp methods 722
IWebBrowserApp properties 727
MenuBar 728
Name 728
Path 728
PutProperty 725
Quit 725
Set_FullScreen 725
Set_MenuBar 726
Set_StatusBar 726
Set_StatusText 726
Set_ToolBar 726
Set_Visible 726
StatusBar 728
StatusText 729
ToolBar 729
Visible 729
IWebBrowserAppDisp interface 729
about IWebBrowserAppDisp interface 729
Application 734
Busy 734
ClientToWindow 731
Container 734
Document 735

FullName 735
FullScreen 735
GetProperty 731
GoBack 731
GoForward 731
GoHome 732
GoSearch 732
Height 735
HWnd 736
IWebBrowserAppDisp Members 730
IWebBrowserAppDisp methods 730
IWebBrowserAppDisp properties 733
Left 736
LocationName 736
LocationURL 736
MenuBar 736
Name 737
Navigate 732
Parent 737
Path 737
PutProperty 732
Quit 733
Refresh 733
Refresh2 733
StatusBar 737
StatusText 738
Stop 733
ToolBar 738
Top 738
TopLevelContainer 738
type_ 738
Visible 739
Width 739
IWebBrowserDisp interface 739
about IWebBrowserDisp interface 739
Application 743
Busy 743
Container 743
Document 743
GoBack 740
GoForward 741
GoHome 741
GoSearch 741
Height 743
IWebBrowserDisp Members 739
IWebBrowserDisp methods 740
IWebBrowserDisp properties 742
Left 744
LocationName 744
LocationURL 744
Navigate 741
Parent 744
Refresh 741
Refresh2 742
Stop 742
Top 745
TopLevelContainer 745
type_ 745
Width 745
IWebComponent interface 6065
about IWebComponent interface 6065
ComponentList 6068
Container 6069
GetContainer 6067
GetIndex 6067
Index 6069
IWebComponent Members 6066
IWebComponent methods 6066
IWebComponent properties 6068
SetComponentList 6067
SetContainer 6067
SetIndex 6068
IWebComponentCommands interface 6069
about IWebComponentCommands interface 6069
ExecuteVerb 6070
GetVerb 6070
GetVerbCount 6071
IWebComponentCommands Members 6070

IWebComponentCommands methods 6070
IWebComponentContainer interface 6071
about IWebComponentContainer interface 6071
ComponentCount 6072
Components 6073
GetComponent 6072
GetComponentCount 6072
IWebComponentContainer Members 6071
IWebComponentContainer methods 6072
IWebComponentContainer properties 6072
IWebComponentEditor interface 6073
about IWebComponentEditor interface 6073
CanAddClass 6074
IWebComponentEditor Members 6073
IWebComponentEditor methods 6074
IWebComponentEditorHelper interface 6074
about IWebComponentEditorHelper interface 6074
CanAddClassHelper 6075
IWebComponentEditorHelper Members 6075
IWebComponentEditorHelper methods 6075
IWebContainerSubComponent interface 1022
about IWebContainerSubComponent interface 1022
IWebContent interface 6075
about IWebContent interface 6075
Content 6076
IWebContent Members 6076
IWebContent methods 6076
IWebDataFields interface 1023
about IWebDataFields interface 1023
FindField 1023
GetFieldsList 1024
GetVisibleFields 1024
IsFieldInUse 1024
IWebDataFields Members 1023
IWebDataFields methods 1023
IWebDataFields properties 1024
VisibleFields 1025
IWebEnabled interface 1025
about IWebEnabled interface 1025
IWebEnabled Members 1025
IWebEnabled methods 1025
WebEnabled 1026
IWebEndUser interface 5962
about IWebEndUser interface 5962
DisplayName 5968
EndUserAdapter 5968
GetDisplayName 5963
GetEndUserAdapter 5964
GetLoggedIn 5964
GetLoginFormAction 5964
GetLogoutAction 5965
HasExecuteRights 5965
HasModifyRights 5966
HasViewRights 5966
IWebEndUser Members 5963
IWebEndUser methods 5963
IWebEndUser properties 5968
LoggedIn 5968
LogIn 5967
LoginFormAction 5969
LogoutAction 5970
ProduceLoginPage 5967
IWebFieldsSubcomponent interface 5879
about IWebFieldsSubcomponent interface 5879
IWebFormComponentContainer interface 1026
about IWebFormComponentContainer interface 1026
Count 1027
GetCount 1027
GetItem 1027
Items 1028
IWebFormComponentContainer Members 1026
IWebFormComponentContainer methods 1026
IWebFormComponentContainer properties 1027
IWebGetActionName interface 1028
about IWebGetActionName interface 1028
ActionName 1029
GetActionName 1029
IWebGetActionName Members 1028

IWebGetActionName methods 1028
IWebGetActionName properties 1029
IWebGetDisplayLabel interface 1029
about IWebGetDisplayLabel interface 1029
GetDisplayLabel 1030
IWebGetDisplayLabel Members 1030
IWebGetDisplayLabel methods 1030
IWebGetDisplayWidth interface 1030
about IWebGetDisplayWidth interface 1030
GetDisplayWidth 1031
IWebGetDisplayWidth Members 1030
IWebGetDisplayWidth methods 1031
IWebGetFieldName interface 1031
about IWebGetFieldName interface 1031
FieldName 1032
GetFieldName 1032
IWebGetFieldName Members 1031
IWebGetFieldName methods 1032
IWebGetFieldName properties 1032
IWebGetFieldPrevValue interface 5880
about IWebGetFieldPrevValue interface 5880
GetNeedPrevValue 5881
GetPrevValue 5881
IWebGetFieldPrevValue Members 5880
IWebGetFieldPrevValue methods 5880
IWebGetFieldPrevValue properties 5881
NeedPrevValue 5881
PrevValue 5881
IWebGetFieldValue interface 1032
about IWebGetFieldValue interface 1032
GetValue 1033
IWebGetFieldValue Members 1033
IWebGetFieldValue methods 1033
IWebGetFieldValue properties 1033
Value 1034
IWebGetFieldValues interface 1034
about IWebGetFieldValues interface 1034
EndValueIterator 1035
GetCurrentValue 1035
HasValue 1035
IWebGetFieldValues Members 1034
IWebGetFieldValues methods 1034
NextValueIteration 1036
StartValueIterator 1036
IWebGetMaxLength interface 1037
about IWebGetMaxLength interface 1037
GetMaxLength 1037
IWebGetMaxLength Members 1037
IWebGetMaxLength methods 1037
IWebGetSessionIDs interface 1038
about IWebGetSessionIDs interface 1038
GetSessionIDs 1038
IWebGetSessionIDs Members 1038
IWebGetSessionIDs methods 1038
IWebImageHREF interface 1039
about IWebImageHREF interface 1039
IWebImageHREF Members 1039
IWebImageHREF methods 1039
WebImageHREF 1040
IWebInputName interface 1040
about IWebInputName interface 1040
GetInputName 1041
InputName 1041
IWebInputName Members 1040
IWebInputName methods 1041
IWebInputName properties 1041
IWebIsDefaultAction interface 1041
about IWebIsDefaultAction interface 1041
IsDefaultAction 1042
IWebIsDefaultAction Members 1042
IWebIsDefaultAction methods 1042
IWebIsDefaultField interface 1042
about IWebIsDefaultField interface 1042
IsDefaultField 1043
IWebIsDefaultField Members 1043
IWebIsDefaultField methods 1043
IWebNode interface 6582
about IWebNode interface 6582

BeforeExecute 6583
Execute 6584
Execute method 6584
GetMimeBoundary 6585
IWebNode Members 6583
IWebNode methods 6583
IWebNode properties 6585
MimeBoundary 6585
SetMimeBoundary 6585
IWebPageInfo interface 5970
about IWebPageInfo interface 5970
GetHasViewAccess 5971
GetLoginRequired 5972
GetPageDescription 5972
GetPageHREF 5972
GetPageIsPublished 5973
GetPageName 5973
GetPageTitle 5973
HasViewAccess 5974
IWebPageInfo Members 5971
IWebPageInfo methods 5971
IWebPageInfo properties 5973
LoginRequired 5974
PageDescription 5974
PageHREF 5975
PageIsPublished 5975
PageName 5976
PageTitle 5976
IWebServExp interface 6701
about IWebServExp interface 6701
FindOrAddSchema 6702
GenerateXMLSchema 6702
IWebServExp Members 6701
IWebServExp methods 6702
IWebServExpAccess interface 6703
about IWebServExpAccess interface 6703
GetWebServExp 6703
IWebServExpAccess Members 6703
IWebServExpAccess methods 6703
IWebSession interface 1043
about IWebSession interface 1043
IWebSession Members 1044
IWebSession methods 1044
Terminate 1044
IWebSessionAccess interface 6231
about IWebSessionAccess interface 6231
GetSessionsService 6232
GetWebSession 6232
IWebSessionAccess Members 6231
IWebSessionAccess methods 6231
IWebSessionAttributes interface 1045
about IWebSessionAttributes interface 1045
ExpirationTime 1048
Expired 1048
GetExpirationTime 1046
GetExpired 1046
GetIsActive 1046
GetStartTime 1047
GetTerminated 1047
GetTimeoutMinutes 1047
GetTouchTime 1047
IsActive 1048
IWebSessionAttributes Members 1045
IWebSessionAttributes methods 1046
IWebSessionAttributes properties 1048
SetTimeoutMinutes 1047
StartTime 1049
Terminated 1049
TimeoutMinutes 1049
TouchTime 1049
IWebSessionID interface 1049
about IWebSessionID interface 1049
GetID 1050
ID 1051
IWebSessionID Members 1050
IWebSessionID methods 1050
IWebSessionID properties 1050
IWebSessionIDs interface 1051

about IWebSessionIDs interface 1051
GetID 1052
GetIDCount 1052
IDCount 1053
IDs 1053
IWebSessionIDs Members 1051
IWebSessionIDs methods 1052
IWebSessionIDs properties 1052
IWebSessionValues interface 1053
about IWebSessionValues interface 1053
GetItemCount 1054
GetItems 1054
GetValue 1055
IWebSessionValues Members 1054
IWebSessionValues methods 1054
IWebSessionValues properties 1055
SetValue 1055
Values 1055
IWebSetActionName interface 1055
about IWebSetActionName interface 1055
IWebSetActionName Members 1056
IWebSetActionName methods 1056
SetActionName 1056
IWebSetFieldName interface 1056
about IWebSetFieldName interface 1056
IWebSetFieldName Members 1057
IWebSetFieldName methods 1057
SetFieldName 1057
IWebUserList interface 1057
about IWebUserList interface 1057
CheckAccessRights 1058
IWebUserList Members 1058
IWebUserList methods 1058
ValidateUser 1059
IWebValueIsEqual interface 1059
about IWebValueIsEqual interface 1059
IsEqual 1060
IWebValueIsEqual Members 1060
IWebValueIsEqual methods 1060
IWebVariablesContainerName interface 6077
about IWebVariablesContainerName interface 6077
ContainerName 6078
GetContainerName 6077
IWebVariablesContainerName Members 6077
IWebVariablesContainerName methods 6077
IWebVariablesContainerName properties 6078
IWideStringsAdapter interface 6819
about IWideStringsAdapter interface 6819
IWideStringsAdapter Members 6819
IWideStringsAdapter methods 6819
ReferenceStrings 6820
ReleaseStrings 6820
IWinHelpTester interface 6846
about IWinHelpTester interface 6846
CanShowALink 6847
CanShowContext 6847
CanShowTopic 6847
GetDefaultHelpFile 6847
GetHelpPath 6848
GetHelpStrings 6848
IWinHelpTester Members 6846
IWinHelpTester methods 6846
IWSDLDocument interface 7115
about IWSDLDocument interface 7115
Definition 7118
GetDefinition 7117
IWSDLDocument Members 7115
IWSDLDocument methods 7117
IWSDLDocument properties 7118
SetDefinition 7117
IWSDLItems interface 7208
about IWSDLItems interface 7208
GetWSDLDoc 7209
GetWSDLItems 7209
IWSDLItems Members 7208
IWSDLItems methods 7208
Load 7209
IWSDLPublish interface 7247

about IWSDLPublish interface 7247
GetPortTypeList 7248
GetTypeSystemsList 7248
GetWSDLForPortType 7248
GetXSDForTypeSystem 7248
IWSDLPublish Members 7247
IWSDLPublish methods 7247
IXMLDocument interface 7586
about IXMLDocument interface 7586
Active 7607
AddChild 7589
AddChild method 7589
AsyncLoadState 7608
ChildNodes 7609
CreateElement 7590
CreateNode 7591
DocumentElement 7609
DOMDocument 7609
Encoding 7610
FileName 7610
GeneratePrefix 7592
GetActive 7592
GetAsyncLoadState 7593
GetChildNodes 7593
GetDocBinding 7593
GetDocumentElement 7594
GetDocumentNode 7594
GetDOMDocument 7594
GetEncoding 7595
GetFileName 7595
GetModified 7595
GetNodeIndentStr 7596
GetOptions 7596
GetParseOptions 7596
GetSchemaRef 7596
GetStandAlone 7597
GetVersion 7597
GetXML 7597
IsEmptyDoc 7598
IXMLDocument Members 7587
IXMLDocument methods 7588
IXMLDocument properties 7607
LoadFromFile 7598
LoadFromStream 7599
LoadFromXML 7599
LoadFromXML method 7599
Modified 7611
Node 7611
NodeIndentStr 7612
Options 7612
ParseOptions 7613
Refresh 7600
RegisterDocBinding 7600
Resync 7601
SaveToFile 7601
SaveToStream 7602
SaveToXML 7602
SaveToXML method 7602
SchemaRef 7613
SetActive 7603
SetDocumentElement 7603
SetDOMDocument 7604
SetEncoding 7604
SetFileName 7604
SetNodeIndentStr 7605
SetOnAsyncLoad 7605
SetOptions 7605
SetParseOptions 7606
SetStandAlone 7606
SetVersion 7606
SetXML 7607
StandAlone 7614
Version 7614
XML 7614
IXMLDocumentAccess interface 7429
about IXMLDocumentAccess interface 7429
DocumentObject 7430
DOMPersist 7431

GetDocumentObject 7430
GetDOMPersist 7430
IXMLDocumentAccess Members 7429
IXMLDocumentAccess methods 7429
IXMLDocumentAccess properties 7430
IXMLNode interface 7615
about IXMLNode interface 7615
AddChild 7618
AddChild method 7618
AttributeNodes 7634
Attributes 7634
ChildNodes 7635
ChildValues 7636
CloneNode 7619
Collection 7636
DeclareNamespace 7620
DOMNode 7637
FindNamespaceDecl 7620
FindNamespaceURI 7620
GetAttribute 7621
GetAttributeNodes 7621
GetAttributeNS 7622
GetChildNodes 7622
GetChildValue 7622
GetCollection 7623
GetDOMNode 7623
GetHasChildNodes 7624
GetIsTextElement 7624
GetLocalName 7624
GetNamespaceURI 7625
GetNodeName 7625
GetNodeType 7625
GetNodeValue 7626
GetOwnerDocument 7626
GetParentNode 7626
GetPrefix 7626
GetReadOnly 7627
GetText 7627
GetXML 7627
HasAttribute 7628
HasAttribute method 7628
HasChildNodes 7637
IsTextElement 7638
IXMLNode Members 7616
IXMLNode methods 7617
IXMLNode properties 7633
LocalName 7638
NamespaceURI 7639
NextSibling 7628
NodeName 7639
NodeType 7640
NodeValue 7641
Normalize 7629
OwnerDocument 7641
ParentNode 7641
Prefix 7642
PreviousSibling 7629
ReadOnly 7642
Resync 7630
SetAttribute 7630
SetAttributeNS 7630
SetChildValue 7631
SetNodeValue 7631
SetReadOnly 7632
SetText 7632
Text 7643
TransformNode 7633
TransformNode method 7632
XML 7643
IXMLNodeAccess interface 7431
about IXMLNodeAccess interface 7431
AddChild 7433
CheckTextNode 7434
ChildNodeClasses 7438
ClearDocumentRef 7434
CreateAttributeNode 7434
CreateChildNode 7434
CreateCollection 7434

DOMElement 7435
FindHostedNode 7435
GetChildNodeClasses 7435
GetHostNode 7435
GetNodeObject 7436
HasChildNode 7436
HasChildNode method 7436
HostNode 7438
InternalAddChild 7436
IXMLNodeAccess Members 7431
IXMLNodeAccess methods 7433
IXMLNodeAccess properties 7438
NestingLevel 7437
RegisterChildNode 7437
RegisterChildNodes 7437
SetCollection 7437
SetParentNode 7438
IXMLNodeCollection interface 7644
about IXMLNodeCollection interface 7644
Clear 7646
Count 7648
Delete 7646
GetCount 7647
GetNode 7647
IXMLNodeCollection Members 7644
IXMLNodeCollection methods 7646
IXMLNodeCollection properties 7648
Nodes 7649
Remove 7648
IXMLNodeList interface 7649
about IXMLNodeList interface 7649
Add 7651
BeginUpdate 7651
Clear 7652
Count 7660
Delete 7653
Delete method 7652
EndUpdate 7653
FindNode 7654
FindNode method 7654
FindSibling 7655
First 7655
Get 7656
GetCount 7656
GetNode 7656
GetUpdateCount 7657
IndexOf 7657
IndexOf method 7657
Insert 7658
IXMLNodeList Members 7650
IXMLNodeList methods 7651
IXMLNodeList properties 7660
Last 7658
Nodes 7660
Remove 7659
ReplaceNode 7659
UpdateCount 7661
IXMLSchemaDefs interface 7119
about IXMLSchemaDefs interface 7119
Add 7121
GetXMLSchemaDef 7121
IXMLSchemaDefs Members 7119
IXMLSchemaDefs methods 7120
IXMLSchemaDefs properties 7121
SchemaDef 7121

K

keyAdd enumeration member 4922
keyDelete enumeration member 4922
keyEdit enumeration member 4922
keyUnique enumeration member 4922

L

lbOwnerDrawFixed enumeration member 2910
lbOwnerDrawVariable enumeration member 2910
lbStandard enumeration member 2910
lbVirtual enumeration member 2910
lbVirtualOwnerDraw enumeration member 2910

loInvariantLocale enumeration member 4369
loUserLocale enumeration member 4369
lposAbove enumeration member 6082
lposBelow enumeration member 6082
lposLeft enumeration member 6082
lposRight enumeration member 6082
lsButtons enumeration member 6493
lsFields enumeration member 6493
lsIdle enumeration member 137
lsLookupAddress enumeration member 137
lsLookupService enumeration member 137
lsUnknown enumeration member 6493

M

mba16Byte enumeration member 3807
mba8Byte enumeration member 3807
mbLeadByte enumeration member 3193, 4369
mbSingleByte enumeration member 3193, 4369
mbTrailByte enumeration member 3193, 4369
mkAmbiguous enumeration member 4364
mkClassConstructor enumeration member 4780
mkClassFunction enumeration member 4780
mkClassProcedure enumeration member 4780
mkConstructor enumeration member 4780
mkDestructor enumeration member 4780
mkExactMatch enumeration member 4364
mkFunction enumeration member 4780
mkNone enumeration member 4364
mkOperatorOverload enumeration member 4780
mkProcedure enumeration member 4780
mkSafeFunction enumeration member 4780
mkSafeProcedure enumeration member 4780
mkSingleMatch enumeration member 4364
mtFault enumeration member 6706
mtHeaderInput enumeration member 6706
mtHeaderOutput enumeration member 6706
mtInput enumeration member 6706
mtOutput enumeration member 6706

N

ncAddAttribute enumeration member 7445
ncInsertChild enumeration member 7445
ncRemoveAttribute enumeration member 7445
ncRemoveChild enumeration member 7445
ncrError enumeration member 5105
ncrLoose enumeration member 5105
ncrStrict enumeration member 5105
ncUpdateValue enumeration member 7445
nlCreateNode enumeration member 7446
nlInsert enumeration member 7446
nlRemove enumeration member 7446
ntAttribute enumeration member 7662
ntCData enumeration member 7662
ntComment enumeration member 7662
ntContainsUnit enumeration member 4370
ntDcpBpiName enumeration member 4370
ntDocFragment enumeration member 7662
ntDocType enumeration member 7662
ntDocument enumeration member 7662
ntElement enumeration member 7662
ntEntity enumeration member 7662
ntEntityRef enumeration member 7662
ntNotation enumeration member 7662
ntProcessingInstr enumeration member 7662
ntRequiresPackage enumeration member 4370
ntReserved enumeration member 7662
ntText enumeration member 7662

O

OleVariant class 3281
    != 3316
    != operator 3315
    ~OleVariant 3286
    < 3316, 3317
    < operator 3316
    <= 3317

<= operator 3317
= 3319
= operator 3317
== 3320
== operator 3320
> 3321
> operator 3320
>= 3321
>= operator 3321
about OleVariant class 3281
OleVariant 3288, 3289, 3290, 3291, 3292, 3293, 3294, 3295, 3296
OleVariant constructor 3286
OleVariant data members 3296
OleVariant Members 3282
OleVariant operators 3315
TVarData::Reserved1 3298
TVarData::Reserved2 3298
TVarData::Reserved3 3298
TVarData::VArray 3298
TVarData::VBoolean 3299
TVarData::VByte 3299
TVarData::VCurrency 3299
TVarData::VDate 3299
TVarData::VDispatch 3299
TVarData::VDouble 3300
TVarData::VError 3300
TVarData::VInteger 3300
TVarData::VOleStr 3300
TVarData::VPointer 3301
TVarData::VSingle 3301
TVarData::VSmallint 3301
TVarData::VType 3301
TVarData::VUnknown 3301
Variant::- 3302
Variant::! 3302
Variant::%= 3302
Variant::&= 3302
Variant::*= 3303
Variant::/= 3303
Variant::^= 3303
Variant::__int64 3303
Variant::|= 3303
Variant::+= 3304
Variant::<<= 3304
Variant::-= 3304
Variant::>>= 3304
Variant::AnsiString 3305
Variant::ArrayDimCount 3305
Variant::ArrayHighBound 3305
Variant::ArrayLock 3305
Variant::ArrayLowBound 3305
Variant::ArrayRedim 3306
Variant::ArrayUnlock 3306
Variant::AsType 3306
Variant::bool 3306
Variant::Byte* 3307
Variant::ChangeType 3307
Variant::char 3307
Variant::Clear 3307
Variant::Currency 3307
Variant::Currency* 3308
Variant::double 3308
Variant::double* 3308
Variant::float 3308
Variant::float* 3309
Variant::GetElement 3309
Variant::GetError 3309
Variant::IDispatch* 3309
Variant::IInterface* 3309
Variant::int 3310
Variant::int* 3310
Variant::IsArray 3310
Variant::IsEmpty 3310
Variant::IsNull 3311
Variant::IUnknown* 3311
Variant::long 3311
Variant::long* 3311
Variant::PutElement 3311

Variant::SetError 3312
Variant::short 3312
Variant::short* 3312
Variant::signed char 3312
Variant::signed char* 3313
Variant::TDateTime 3313
Variant::TDateTime* 3313
Variant::Type 3313
Variant::UnicodeString 3313
Variant::unsigned char 3314
Variant::unsigned int 3314
Variant::unsigned int* 3314
Variant::unsigned long 3314
Variant::unsigned short 3315
Variant::unsigned short* 3315
Variant::WideString 3315
otSByte enumeration member 4782
otSLong enumeration member 4782
otSWord enumeration member 4782
otUByte enumeration member 4782
otULong enumeration member 4782
otUWord enumeration member 4782

P

pfAddress enumeration member 4783
pfAppleTalk enumeration member 1404
pfArray enumeration member 4783
pfBan enumeration member 1404
pfCcitt enumeration member 1404
pfChaos enumeration member 1404
pfConst enumeration member 4783
pfDataKit enumeration member 1404
pfDecNet enumeration member 1404
pfDli enumeration member 1404
pfEcma enumeration member 1404
pfFireFox enumeration member 1404
pfHylink enumeration member 1404
pfImpLink enumeration member 1404
pfInet enumeration member 1404
pfIpx enumeration member 1404
pfIso enumeration member 1404
pfLat enumeration member 1404
pfMax enumeration member 1404
pfNs enumeration member 1404
pfOsi enumeration member 1404
pfOut enumeration member 4783
pfPup enumeration member 1404
pfReference enumeration member 4783
pfSna enumeration member 1404
pfUnix enumeration member 1404
pfUnknown1 enumeration member 1404
pfUnspec enumeration member 1404
pfVar enumeration member 4783
pfVoiceView enumeration member 1404
poAsyncLoad enumeration member 7664
poDefault enumeration member 7250
poPreserveWhiteSpace enumeration member 7664
poPublishLocationAsSecure enumeration member 7250
poResolveExternals enumeration member 7664
poUTF8ContentType enumeration member 7250
poValidateOnParse enumeration member 7664
ptCustom enumeration member 319
ptHelp enumeration member 319
ptInfo enumeration member 319

Q

qaNoModuleName enumeration member 1065
qaNoModulesVar enumeration member 1065
qaUseModulesVar enumeration member 1065

R

reAccessViolation enumeration member 3809
reAssertionFailed enumeration member 3809
reControlBreak enumeration member 3809
reDivByZero enumeration member 3809
reExternalException enumeration member 3809
reIntfCastError enumeration member 3809
reIntOverflow enumeration member 3809

reInvalidCast enumeration member 3809
reInvalidOp enumeration member 3809
reInvalidPtr enumeration member 3809
reMonitorNotLocked enumeration member 3809
reNoMonitorSupport enumeration member 3809
reNone enumeration member 3809
reOutOfMemory enumeration member 3809
reOverflow enumeration member 3809
rePrivInstruction enumeration member 3809
reRangeError enumeration member 3809
reSafeCallError enumeration member 3809
reStackOverflow enumeration member 3809
reUnderflow enumeration member 3809
reVarArrayBounds enumeration member 3809
reVarArrayCreate enumeration member 3809
reVarDispatch enumeration member 3809
reVarInvalidOp enumeration member 3809
reVarNotArray enumeration member 3809
reVarTypeCast enumeration member 3809
reZeroDivide enumeration member 3809
rnActivateModule enumeration member 6609
rnCreateModule enumeration member 6609
rnDeactivateModule enumeration member 6609
rnFinishRequest enumeration member 6609
rnFreeModule enumeration member 6609
rnStartRequest enumeration member 6609
roExcludeID enumeration member 1065

S

sbLeft enumeration member 4574
sbRight enumeration member 4574
sbsNone enumeration member 2913
sbsSingle enumeration member 2913
sbsSunken enumeration member 2913
scBottom enumeration member 2911
scEndScroll enumeration member 2911
ScktComp 1
ScktComp namespace 1
Classes 3
Constants 143
Functions 133
Structs, Records, Enums 134
Types 139
ScktComp.CM_DEFERFREE 143
ScktComp.CM_DEFERFREE constant 143
ScktComp.CM_LOOKUPCOMPLETE 143
ScktComp.CM_LOOKUPCOMPLETE constant 143
ScktComp.CM_SOCKETMESSAGE 143
ScktComp.CM_SOCKETMESSAGE constant 143
ScktComp.ESocketError 3
ScktComp.SetErrorProc 133
ScktComp.SetErrorProc function 133
ScktComp.TAbstractSocket 4
ScktComp.TAbstractSocket.Active 11
ScktComp.TAbstractSocket.Address 12
ScktComp.TAbstractSocket.Close 6
ScktComp.TAbstractSocket.DoActivate 6
ScktComp.TAbstractSocket.Error 6
ScktComp.TAbstractSocket.Event 7
ScktComp.TAbstractSocket.Host 12
ScktComp.TAbstractSocket.InitSocket 8
ScktComp.TAbstractSocket.Loaded 8
ScktComp.TAbstractSocket.Open 9
ScktComp.TAbstractSocket.Port 13
ScktComp.TAbstractSocket.Service 13
ScktComp.TAbstractSocket.SetActive 9
ScktComp.TAbstractSocket.SetAddress 10
ScktComp.TAbstractSocket.SetHost 10
ScktComp.TAbstractSocket.SetPort 10
ScktComp.TAbstractSocket.SetService 11
ScktComp.TAsyncStyle 134
ScktComp.TAsyncStyle enumeration 134
ScktComp.TAsyncStyles 139
ScktComp.TAsyncStyles type 139
ScktComp.TClientSocket 14
ScktComp.TClientSocket.Active 18
ScktComp.TClientSocket.Address 18
ScktComp.TClientSocket.ClientType 19

ScktComp.TClientSocket.Create 16
ScktComp.TClientSocket.Destroy 16
ScktComp.TClientSocket.DoActivate 17
ScktComp.TClientSocket.GetClientType 17
ScktComp.TClientSocket.Host 20
ScktComp.TClientSocket.OnConnect 22
ScktComp.TClientSocket.OnConnecting 22
ScktComp.TClientSocket.OnDisconnect 23
ScktComp.TClientSocket.OnError 23
ScktComp.TClientSocket.OnLookup 24
ScktComp.TClientSocket.OnRead 24
ScktComp.TClientSocket.OnWrite 25
ScktComp.TClientSocket.Port 20
ScktComp.TClientSocket.Service 21
ScktComp.TClientSocket.SetClientType 17
ScktComp.TClientSocket.Socket 21
ScktComp.TClientType 135
ScktComp.TClientType enumeration 135
ScktComp.TClientWinSocket 25
ScktComp.TClientWinSocket.ClientType 28
ScktComp.TClientWinSocket.Connect 28
ScktComp.TClientWinSocket.SetClientType 28
ScktComp.TCMLookupComplete 135
ScktComp.TCMLookupComplete record 135
ScktComp.TCMSocketMessage 136
ScktComp.TCMSocketMessage record 136
ScktComp.TCustomServerSocket 29
ScktComp.TCustomServerSocket.Destroy 31
ScktComp.TCustomServerSocket.DoActivate 32
ScktComp.TCustomServerSocket.FServerSocket 32
ScktComp.TCustomServerSocket.GetGetSocketEvent 33
ScktComp.TCustomServerSocket.GetGetThreadEvent 33
ScktComp.TCustomServerSocket.GetOnClientError 34
ScktComp.TCustomServerSocket.GetOnClientEvent 34
ScktComp.TCustomServerSocket.GetOnThreadEnd 34
ScktComp.TCustomServerSocket.GetOnThreadStart 35
ScktComp.TCustomServerSocket.GetServerType 35
ScktComp.TCustomServerSocket.GetThreadCacheSize 35
ScktComp.TCustomServerSocket.OnClientConnect 41
ScktComp.TCustomServerSocket.OnClientDisconnect 42
ScktComp.TCustomServerSocket.OnClientError 42
ScktComp.TCustomServerSocket.OnClientRead 43
ScktComp.TCustomServerSocket.OnClientWrite 43
ScktComp.TCustomServerSocket.OnGetSocket 44
ScktComp.TCustomServerSocket.OnGetThread 44
ScktComp.TCustomServerSocket.OnThreadEnd 45
ScktComp.TCustomServerSocket.OnThreadStart 45
ScktComp.TCustomServerSocket.ServerType 39
ScktComp.TCustomServerSocket.SetGetSocketEvent 36
ScktComp.TCustomServerSocket.SetGetThreadEvent 36
ScktComp.TCustomServerSocket.SetOnClientError 37
ScktComp.TCustomServerSocket.SetOnClientEvent 37
ScktComp.TCustomServerSocket.SetOnThreadEnd 37
ScktComp.TCustomServerSocket.SetOnThreadStart 38
ScktComp.TCustomServerSocket.SetServerType 38
ScktComp.TCustomServerSocket.SetThreadCacheSize 39
ScktComp.TCustomServerSocket.ThreadCacheSize 40
ScktComp.TCustomSocket 46
ScktComp.TCustomSocket.Error 48
ScktComp.TCustomSocket.Event 48
ScktComp.TCustomSocket.OnAccept 49
ScktComp.TCustomSocket.OnConnect 50
ScktComp.TCustomSocket.OnConnecting 50
ScktComp.TCustomSocket.OnDisconnect 51
ScktComp.TCustomSocket.OnError 51
ScktComp.TCustomSocket.OnListen 51
ScktComp.TCustomSocket.OnLookup 52
ScktComp.TCustomSocket.OnRead 53
ScktComp.TCustomSocket.OnWrite 53
ScktComp.TCustomWinSocket 54
ScktComp.TCustomWinSocket.Accept 58
ScktComp.TCustomWinSocket.Addr 72
ScktComp.TCustomWinSocket.AsyncInitSocket 58
ScktComp.TCustomWinSocket.ASyncStyles 72
ScktComp.TCustomWinSocket.Close 59
ScktComp.TCustomWinSocket.Connect 59
ScktComp.TCustomWinSocket.Connected 73
ScktComp.TCustomWinSocket.Create 56

ScktComp.TCustomWinSocket.Data 73
ScktComp.TCustomWinSocket.DefaultHandler 60
ScktComp.TCustomWinSocket.Destroy 56
ScktComp.TCustomWinSocket.Disconnect 60
ScktComp.TCustomWinSocket.DoListen 60
ScktComp.TCustomWinSocket.DoOpen 61
ScktComp.TCustomWinSocket.Error 61
ScktComp.TCustomWinSocket.Event 62
ScktComp.TCustomWinSocket.Handle 73
ScktComp.TCustomWinSocket.InitSocket 62
ScktComp.TCustomWinSocket.Listen 63
ScktComp.TCustomWinSocket.LocalAddress 74
ScktComp.TCustomWinSocket.LocalHost 74
ScktComp.TCustomWinSocket.LocalPort 75
ScktComp.TCustomWinSocket.Lock 64
ScktComp.TCustomWinSocket.LookupName 64
ScktComp.TCustomWinSocket.LookupService 65
ScktComp.TCustomWinSocket.LookupState 75
ScktComp.TCustomWinSocket.OnErrorEvent 79
ScktComp.TCustomWinSocket.OnSocketEvent 79
ScktComp.TCustomWinSocket.Open 65
ScktComp.TCustomWinSocket.Read 66
ScktComp.TCustomWinSocket.ReceiveBuf 66
ScktComp.TCustomWinSocket.ReceiveLength 67
ScktComp.TCustomWinSocket.ReceiveText 67
ScktComp.TCustomWinSocket.RemoteAddr 76
ScktComp.TCustomWinSocket.RemoteAddress 76
ScktComp.TCustomWinSocket.RemoteHost 77
ScktComp.TCustomWinSocket.RemotePort 77
ScktComp.TCustomWinSocket.SendBuf 68
ScktComp.TCustomWinSocket.SendStream 68
ScktComp.TCustomWinSocket.SendStreamThenDrop 69
ScktComp.TCustomWinSocket.SendText 69
ScktComp.TCustomWinSocket.SetAsyncStyles 70
ScktComp.TCustomWinSocket.SocketHandle 78
ScktComp.TCustomWinSocket.Unlock 70
ScktComp.TCustomWinSocket.Write 70
ScktComp.TErrorEvent 136
ScktComp.TErrorEvent enumeration 136
ScktComp.TGetSocketEvent 139
ScktComp.TGetSocketEvent type 139
ScktComp.TGetThreadEvent 140
ScktComp.TGetThreadEvent type 140
ScktComp.TLookupState 137
ScktComp.TLookupState enumeration 137
ScktComp.TServerAcceptThread 80
ScktComp.TServerAcceptThread.Create 80
ScktComp.TServerAcceptThread.Execute 81
ScktComp.TServerAcceptThread.ServerSocket 85
ScktComp.TServerClientThread 85
ScktComp.TServerClientThread.ClientExecute 88
ScktComp.TServerClientThread.ClientSocket 91
ScktComp.TServerClientThread.Create 86
ScktComp.TServerClientThread.Data 92
ScktComp.TServerClientThread.Destroy 87
ScktComp.TServerClientThread.DoTerminate 88
ScktComp.TServerClientThread.EndConnect 89
ScktComp.TServerClientThread.Error 89
ScktComp.TServerClientThread.Event 90
ScktComp.TServerClientThread.Execute 90
ScktComp.TServerClientThread.HandleException 90
ScktComp.TServerClientThread.KeepInCache 92
ScktComp.TServerClientThread.ReActivate 91
ScktComp.TServerClientThread.ServerSocket 93
ScktComp.TServerClientThread.StartConnect 91
ScktComp.TServerClientWinSocket 93
ScktComp.TServerClientWinSocket.Create 95
ScktComp.TServerClientWinSocket.Destroy 96
ScktComp.TServerClientWinSocket.ServerWinSocket 96
ScktComp.TServerSocket 97
ScktComp.TServerSocket.Active 100
ScktComp.TServerSocket.Create 99
ScktComp.TServerSocket.OnAccept 103
ScktComp.TServerSocket.OnClientConnect 104
ScktComp.TServerSocket.OnClientDisconnect 105
ScktComp.TServerSocket.OnClientError 105
ScktComp.TServerSocket.OnClientRead 106
ScktComp.TServerSocket.OnClientWrite 106

ScktComp.TServerSocket.OnGetSocket 107
ScktComp.TServerSocket.OnGetThread 107
ScktComp.TServerSocket.OnListen 108
ScktComp.TServerSocket.OnThreadEnd 108
ScktComp.TServerSocket.OnThreadStart 109
ScktComp.TServerSocket.Port 100
ScktComp.TServerSocket.ServerType 101
ScktComp.TServerSocket.Service 101
ScktComp.TServerSocket.Socket 102
ScktComp.TServerSocket.ThreadCacheSize 102
ScktComp.TServerType 137
ScktComp.TServerType enumeration 137
ScktComp.TServerWinSocket 109
ScktComp.TServerWinSocket.Accept 113
ScktComp.TServerWinSocket.ActiveConnections 118
ScktComp.TServerWinSocket.ActiveThreads 119
ScktComp.TServerWinSocket.ClientConnect 114
ScktComp.TServerWinSocket.ClientDisconnect 114
ScktComp.TServerWinSocket.ClientErrorEvent 114
ScktComp.TServerWinSocket.ClientRead 114
ScktComp.TServerWinSocket.ClientWrite 115
ScktComp.TServerWinSocket.Connections 119
ScktComp.TServerWinSocket.Create 112
ScktComp.TServerWinSocket.Destroy 112
ScktComp.TServerWinSocket.Disconnect 115
ScktComp.TServerWinSocket.DoCreateThread 115
ScktComp.TServerWinSocket.GetClientSocket 116
ScktComp.TServerWinSocket.GetClientThread 116
ScktComp.TServerWinSocket.GetServerThread 116
ScktComp.TServerWinSocket.IdleThreads 120
ScktComp.TServerWinSocket.Listen 116
ScktComp.TServerWinSocket.OnClientConnect 122
ScktComp.TServerWinSocket.OnClientDisconnect 123
ScktComp.TServerWinSocket.OnClientError 124
ScktComp.TServerWinSocket.OnClientRead 124
ScktComp.TServerWinSocket.OnClientWrite 125
ScktComp.TServerWinSocket.OnGetSocket 126
ScktComp.TServerWinSocket.OnGetThread 126
ScktComp.TServerWinSocket.OnThreadEnd 127
ScktComp.TServerWinSocket.OnThreadStart 127
ScktComp.TServerWinSocket.ServerType 120
ScktComp.TServerWinSocket.SetServerType 117
ScktComp.TServerWinSocket.SetThreadCacheSize 117
ScktComp.TServerWinSocket.ThreadCacheSize 121
ScktComp.TServerWinSocket.ThreadEnd 118
ScktComp.TServerWinSocket.ThreadStart 118
ScktComp.TSocketErrorEvent 140
ScktComp.TSocketErrorEvent type 140
ScktComp.TSocketErrorProc 141
ScktComp.TSocketErrorProc type 141
ScktComp.TSocketEvent 137
ScktComp.TSocketEvent enumeration 137
ScktComp.TSocketEventEvent 141
ScktComp.TSocketEventEvent type 141
ScktComp.TSocketNotifyEvent 141
ScktComp.TSocketNotifyEvent type 141
ScktComp.TThreadNotifyEvent 142
ScktComp.TThreadNotifyEvent type 142
ScktComp.TWinSocketStream 127
ScktComp.TWinSocketStream.Create 129
ScktComp.TWinSocketStream.Destroy 129
ScktComp.TWinSocketStream.Read 130
ScktComp.TWinSocketStream.Seek 130
ScktComp.TWinSocketStream.TimeOut 132
ScktComp.TWinSocketStream.WaitForData 131
ScktComp.TWinSocketStream.Write 131
scLineDown enumeration member 2911
scLineUp enumeration member 2911
SConnect 145
SConnect namespace 145
Classes 147
Constants 243
Functions 237
Interfaces 221
Structs, Records, Enums 238
Types 239
Variables 241
SConnect.asCreateObject 243

SConnect.asCreateObject constant 243
SConnect.asError 243
SConnect.asError constant 243
SConnect.asFreeObject 244
SConnect.asFreeObject constant 244
SConnect.asGetAppServers 244
SConnect.asGetAppServers constant 244
SConnect.asGetGUID 244
SConnect.asGetGUID constant 244
SConnect.asGetID 245
SConnect.asGetID constant 245
SConnect.asGetServers 245
SConnect.asGetServers constant 245
SConnect.asInvoke 245
SConnect.asInvoke constant 245
SConnect.asMask 246
SConnect.asMask constant 246
SConnect.asSoapCommand 246
SConnect.asSoapCommand constant 246
SConnect.CallSig 246
SConnect.CallSig constant 246
SConnect.EInterpreterError 147
SConnect.ESocketConnectionError 148
SConnect.GetPacketInterceptorList 237
SConnect.GetPacketInterceptorList function 237
SConnect.IDataBlock 221
SConnect.IDataBlock.BytesReserved 227
SConnect.IDataBlock.Clear 222
SConnect.IDataBlock.GetBytesReserved 223
SConnect.IDataBlock.GetMemory 223
SConnect.IDataBlock.GetSignature 223
SConnect.IDataBlock.GetSize 224
SConnect.IDataBlock.GetStream 224
SConnect.IDataBlock.IgnoreStream 224
SConnect.IDataBlock.InitData 225
SConnect.IDataBlock.Memory 227
SConnect.IDataBlock.Read 225
SConnect.IDataBlock.SetSignature 226
SConnect.IDataBlock.SetSize 226
SConnect.IDataBlock.Signature 228
SConnect.IDataBlock.Size 228
SConnect.IDataBlock.Stream 229
SConnect.IDataBlock.Write 226
SConnect.IDataIntercept 229
SConnect.IDataIntercept.DataIn 230
SConnect.IDataIntercept.DataOut 231
SConnect.ISendDataBlock 231
SConnect.ISendDataBlock.Send 232
SConnect.ITransport 233
SConnect.ITransport.Connected 236
SConnect.ITransport.GetConnected 234
SConnect.ITransport.GetWaitEvent 234
SConnect.ITransport.Receive 235
SConnect.ITransport.Send 235
SConnect.ITransport.SetConnected 235
SConnect.LoadWinSock2 237
SConnect.LoadWinSock2 function 237
SConnect.PIntArray 239
SConnect.PIntArray type 239
SConnect.PVariantArray 239
SConnect.PVariantArray type 239
SConnect.ResultSig 247
SConnect.ResultSig constant 247
SConnect.TCustomDataBlockInterpreter 148
SConnect.TCustomDataBlockInterpreter.AddDispatch 150
SConnect.TCustomDataBlockInterpreter.CallCreateObject 150
SConnect.TCustomDataBlockInterpreter.CallFreeObject 150
SConnect.TCustomDataBlockInterpreter.CallGetIDsOfNames 150
SConnect.TCustomDataBlockInterpreter.CallGetServerList 151
SConnect.TCustomDataBlockInterpreter.CallInvoke 151
SConnect.TCustomDataBlockInterpreter.CanCreateObject 151
SConnect.TCustomDataBlockInterpreter.CreateObject 151
SConnect.TCustomDataBlockInterpreter.InternalCreateObject 151
SConnect.TCustomDataBlockInterpreter.InterpretData 152
SConnect.TCustomDataBlockInterpreter.LockObject 152
SConnect.TCustomDataBlockInterpreter.ReleaseObject 152
SConnect.TCustomDataBlockInterpreter.RemoveDispatch 152

SConnect.TCustomDataBlockInterpreter.StoreObject 153
SConnect.TCustomDataBlockInterpreter.UnlockObject 153
SConnect.TDataBlock 153
SConnect.TDataBlock.BytesReserved 161
SConnect.TDataBlock.Clear 156
SConnect.TDataBlock.Create 155
SConnect.TDataBlock.Destroy 155
SConnect.TDataBlock.GetBytesReserved 157
SConnect.TDataBlock.GetMemory 157
SConnect.TDataBlock.GetSignature 157
SConnect.TDataBlock.GetSize 158
SConnect.TDataBlock.GetStream 158
SConnect.TDataBlock.IgnoreStream 158
SConnect.TDataBlock.InitData 158
SConnect.TDataBlock.Memory 161
SConnect.TDataBlock.Read 159
SConnect.TDataBlock.SetSignature 159
SConnect.TDataBlock.SetSize 160
SConnect.TDataBlock.Signature 162
SConnect.TDataBlock.Size 162
SConnect.TDataBlock.Stream 162
SConnect.TDataBlock.Write 160
SConnect.TDataBlockInterpreter 163
SConnect.TDataBlockInterpreter.AddDispatch 165
SConnect.TDataBlockInterpreter.CallCreateObject 166
SConnect.TDataBlockInterpreter.CallFreeObject 166
SConnect.TDataBlockInterpreter.CallGetIDsOfNames 166
SConnect.TDataBlockInterpreter.CallGetServerList 166
SConnect.TDataBlockInterpreter.CallInvoke 167
SConnect.TDataBlockInterpreter.CanCreateObject 167
SConnect.TDataBlockInterpreter.Create 164
SConnect.TDataBlockInterpreter.CreateObject 167
SConnect.TDataBlockInterpreter.Destroy 164
SConnect.TDataBlockInterpreter.DoCreateObject 167
SConnect.TDataBlockInterpreter.DoCustomAction 167
SConnect.TDataBlockInterpreter.DoFreeObject 168
SConnect.TDataBlockInterpreter.DoGetAppServerList 168
SConnect.TDataBlockInterpreter.DoGetIDsOfNames 168
SConnect.TDataBlockInterpreter.DoGetServerList 168
SConnect.TDataBlockInterpreter.DoInvoke 169
SConnect.TDataBlockInterpreter.InternalCreateObject 169
SConnect.TDataBlockInterpreter.InterpretData 169
SConnect.TDataBlockInterpreter.LockObject 169
SConnect.TDataBlockInterpreter.ReleaseObject 170
SConnect.TDataBlockInterpreter.RemoveDispatch 170
SConnect.TDataBlockInterpreter.StoreObject 170
SConnect.TDataBlockInterpreter.UnlockObject 170
SConnect.TDataDispatch 171
SConnect.TDataDispatch.Create 172
SConnect.TDataDispatch.Destroy 173
SConnect.TDataDispatch.DispatchIndex 174
SConnect.TDataDispatch.GetIDsOfNames 173
SConnect.TDataDispatch.GetTypeInfo 174
SConnect.TDataDispatch.GetTypeInfoCount 174
SConnect.TDataDispatch.Invoke 174
SConnect.THREAD_EXCEPTION 247
SConnect.THREAD_EXCEPTION constant 247
SConnect.THREAD_RECEIVEDSTREAM 247
SConnect.THREAD_RECEIVEDSTREAM constant 247
SConnect.THREAD_REPLACETRANSPORT 248
SConnect.THREAD_REPLACETRANSPORT constant 248
SConnect.THREAD_SENDNOTIFY 248
SConnect.THREAD_SENDNOTIFY constant 248
SConnect.THREAD_SENDSTREAM 248
SConnect.THREAD_SENDSTREAM constant 248
SConnect.TIntArray 240
SConnect.TIntArray type 240
SConnect.TPacketInterceptFactory 175
SConnect.TPacketInterceptFactory.UpdateRegistry 176
SConnect.TSocketConnection 176
SConnect.TSocketConnection.Address 180
SConnect.TSocketConnection.Create 178
SConnect.TSocketConnection.CreateTransport 179
SConnect.TSocketConnection.DoConnect 179
SConnect.TSocketConnection.Host 181
SConnect.TSocketConnection.InterceptGUID 181
SConnect.TSocketConnection.InterceptName 182
SConnect.TSocketConnection.ObjectBroker 183

SConnect.TSocketConnection.Port 183
SConnect.TSocketConnection.SupportCallbacks 184
SConnect.TSocketTransport 185
SConnect.TSocketTransport.Address 190
SConnect.TSocketTransport.Create 186
SConnect.TSocketTransport.Destroy 187
SConnect.TSocketTransport.GetConnected 188
SConnect.TSocketTransport.GetWaitEvent 188
SConnect.TSocketTransport.Host 190
SConnect.TSocketTransport.InterceptGUID 190
SConnect.TSocketTransport.Port 190
SConnect.TSocketTransport.Receive 188
SConnect.TSocketTransport.Send 189
SConnect.TSocketTransport.SetConnected 189
SConnect.TSocketTransport.Socket 191
SConnect.TStreamedConnection 191
SConnect.TStreamedConnection._AddRef 194
SConnect.TStreamedConnection._Release 194
SConnect.TStreamedConnection.Create 193
SConnect.TStreamedConnection.CreateTransport 195
SConnect.TStreamedConnection.Destroy 193
SConnect.TStreamedConnection.DoConnect 195
SConnect.TStreamedConnection.DoDisconnect 196
SConnect.TStreamedConnection.DoError 196
SConnect.TStreamedConnection.GetInterceptorList 196
SConnect.TStreamedConnection.GetInterpreter 197
SConnect.TStreamedConnection.GetServerList 197
SConnect.TStreamedConnection.Handle 201
SConnect.TStreamedConnection.InterceptGUID 201
SConnect.TStreamedConnection.InterceptName 202
SConnect.TStreamedConnection.InternalClose 198
SConnect.TStreamedConnection.InternalOpen 198
SConnect.TStreamedConnection.Interpreter 203
SConnect.TStreamedConnection.QueryInterface 199
SConnect.TStreamedConnection.Send 199
SConnect.TStreamedConnection.SupportCallbacks 203
SConnect.TStreamedConnection.ThreadException 199
SConnect.TStreamedConnection.ThreadReceivedStream 200
SConnect.TStreamedConnection.WndProc 200
SConnect.TTransportThread 204
SConnect.TTransportThread.Create 204
SConnect.TTransportThread.Destroy 204
SConnect.TTransportThread.Execute 205
SConnect.TTransportThread.Semaphore 209
SConnect.TVarFlag 238
SConnect.TVarFlag enumeration 238
SConnect.TVarFlags 240
SConnect.TVarFlags type 240
SConnect.TVariantArray 240
SConnect.TVariantArray type 240
SConnect.TWebConnection 209
SConnect.TWebConnection.Agent 217
SConnect.TWebConnection.Create 212
SConnect.TWebConnection.CreateTransport 213
SConnect.TWebConnection.Destroy 212
SConnect.TWebConnection.DoConnect 213
SConnect.TWebConnection.GetInterpreter 214
SConnect.TWebConnection.GetWaitEvent 214
SConnect.TWebConnection.ObjectBroker 217
SConnect.TWebConnection.Password 218
SConnect.TWebConnection.Proxy 218
SConnect.TWebConnection.ProxyByPass 219
SConnect.TWebConnection.Receive 214
SConnect.TWebConnection.Send 215
SConnect.TWebConnection.SetConnected 215
SConnect.TWebConnection.SupportCallbacks 219
SConnect.TWebConnection.Transport_GetConnected 216
SConnect.TWebConnection.Transport_SetConnected 216
SConnect.TWebConnection.URL 220
SConnect.TWebConnection.UserName 220
SConnect.WSACloseEvent 241
SConnect.WSACloseEvent variable 241
SConnect.WSACreateEvent 241
SConnect.WSACreateEvent variable 241
SConnect.WSAEventSelect 242
SConnect.WSAEventSelect variable 242
SConnect.WSAResetEvent 242
SConnect.WSAResetEvent variable 242

scPageDown enumeration member 2911
scPageUp enumeration member 2911
scPosition enumeration member 2911
ScreenTips 249
ScreenTips namespace 249
Classes 251
Constants 323
Interfaces 315
Structs, Records, Enums 317
Types 320
ScreenTips.cDefaultBorderColor 323
ScreenTips.cDefaultBorderColor constant 323
ScreenTips.cDefaultEndColor 323
ScreenTips.cDefaultEndColor constant 323
ScreenTips.cDefaultStartColor 324
ScreenTips.cDefaultStartColor constant 324
ScreenTips.cScreenTipHeaderSpacing 324
ScreenTips.cScreenTipHeaderSpacing constant 324
ScreenTips.cScreenTipMinimumHeight 324
ScreenTips.cScreenTipMinimumHeight constant 324
ScreenTips.cScreenTipTextAndImageWidth 325
ScreenTips.cScreenTipTextAndImageWidth constant 325
ScreenTips.cScreenTipTextOnlyWidth 325
ScreenTips.cScreenTipTextOnlyWidth constant 325
ScreenTips.EAllocatedActionsException 251
ScreenTips.IScreenTipItem 315
ScreenTips.IScreenTipItem.GetScreenTipItem 316
ScreenTips.TAllocatedActions 251
ScreenTips.TAllocatedActions.AddAction 253
ScreenTips.TAllocatedActions.Clear 253
ScreenTips.TAllocatedActions.Count 254
ScreenTips.TAllocatedActions.Create 252
ScreenTips.TAllocatedActions.Delete 253
ScreenTips.TAllocatedActions.Destroy 252
ScreenTips.TAllocatedActions.Exists 253
ScreenTips.TAllocatedActions.Find 254
ScreenTips.TAllocatedActions.GetItem 254
ScreenTips.TAssociateItem 255
ScreenTips.TAssociateItem.Associate 256
ScreenTips.TAssociateItem.Create 255
ScreenTips.TAssociateItem.Destroy 255
ScreenTips.TAssociateItem.PopupTip 256
ScreenTips.TAssociateItemList 256
ScreenTips.TAssociateItemList.Add 258
ScreenTips.TAssociateItemList.Create 257
ScreenTips.TAssociateItemList.Destroy 257
ScreenTips.TAssociateItemList.Find 258
ScreenTips.TAssociateItemList.Get 258
ScreenTips.TAssociateItemList.Remove 258
ScreenTips.TCreateScreenTipEvent 320
ScreenTips.TCreateScreenTipEvent type 320
ScreenTips.TCustomScreenTipsManager 259
ScreenTips.TCustomScreenTipsManager.AddPopup 262
ScreenTips.TCustomScreenTipsManager.BorderStyle 267
ScreenTips.TCustomScreenTipsManager.ClearScreenTips 262
ScreenTips.TCustomScreenTipsManager.Create 260
ScreenTips.TCustomScreenTipsManager.CreateScreenTip 262
ScreenTips.TCustomScreenTipsManager.Destroy 261
ScreenTips.TCustomScreenTipsManager.DisplayOffset 267
ScreenTips.TCustomScreenTipsManager.DoCreateScreenTip 263
ScreenTips.TCustomScreenTipsManager.DoDuplicateCreateError 263
ScreenTips.TCustomScreenTipsManager.DoGetScreenTipClass 263
ScreenTips.TCustomScreenTipsManager.DoScreenTipsLoaded 263
ScreenTips.TCustomScreenTipsManager.DoShow 263
ScreenTips.TCustomScreenTipsManager.DoShowScreenTip 264
ScreenTips.TCustomScreenTipsManager.Font 267
ScreenTips.TCustomScreenTipsManager.Footer 267
ScreenTips.TCustomScreenTipsManager.FooterImage 268
ScreenTips.TCustomScreenTipsManager.GenerateScreenTips 264
ScreenTips.TCustomScreenTipsManager.GetScreenTip 264
ScreenTips.TCustomScreenTipsManager.GradientEndColor 268
ScreenTips.TCustomScreenTipsManager.GradientStartColor 268

ScreenTips.TCustomScreenTipsManager.LinkedActionLists 268
ScreenTips.TCustomScreenTipsManager.Loaded 264
ScreenTips.TCustomScreenTipsManager.Notification 265
ScreenTips.TCustomScreenTipsManager.OnCreateScreenTip 269
ScreenTips.TCustomScreenTipsManager.OnDuplicateCreateError 269
ScreenTips.TCustomScreenTipsManager.OnGetScreenTipWinClass 270
ScreenTips.TCustomScreenTipsManager.OnScreenTipsLoaded 270
ScreenTips.TCustomScreenTipsManager.OnShowScreenTip 270
ScreenTips.TCustomScreenTipsManager.Options 268
ScreenTips.TCustomScreenTipsManager.RemovePopup 265
ScreenTips.TCustomScreenTipsManager.RemoveScreenTip 266
ScreenTips.TCustomScreenTipsManager.ScreenTips 269
ScreenTips.TCustomScreenTipsManager.UpdateActionInfo 266
ScreenTips.TCustomScreenTipsManager.UpdateOptions 266
ScreenTips.TCustomScreenTipsPopup 270
ScreenTips.TCustomScreenTipsPopup.Associate 275
ScreenTips.TCustomScreenTipsPopup.Create 271
ScreenTips.TCustomScreenTipsPopup.Destroy 272
ScreenTips.TCustomScreenTipsPopup.GetScreenTipItem 273
ScreenTips.TCustomScreenTipsPopup.Glyph 275
ScreenTips.TCustomScreenTipsPopup.LinkPopupToManager 273
ScreenTips.TCustomScreenTipsPopup.Notification 273
ScreenTips.TCustomScreenTipsPopup.Paint 274
ScreenTips.TCustomScreenTipsPopup.PopupType 275
ScreenTips.TCustomScreenTipsPopup.RemoveLinkToManager 274
ScreenTips.TCustomScreenTipsPopup.ScreenTip 276
ScreenTips.TCustomScreenTipsPopup.ScreenTipManager 276
ScreenTips.TCustomScreenTipsPopup.ShowHint 276
ScreenTips.TCustomScreenTipsPopup.Transparent 278
ScreenTips.TCustomScreenTipsPopup.UpdateAssociateToManager 274
ScreenTips.TDescriptionType 317
ScreenTips.TDescriptionType enumeration 317
ScreenTips.TDisplayOffset 278
ScreenTips.TDisplayOffset.Control 279
ScreenTips.TDisplayOffset.Create 278
ScreenTips.TDisplayOffset.X 279
ScreenTips.TDisplayOffset.Y 279
ScreenTips.TDuplicateCreateErrorEvent 320
ScreenTips.TDuplicateCreateErrorEvent type 320
ScreenTips.TImageAlign 317
ScreenTips.TImageAlign enumeration 317
ScreenTips.TItemImageAlign 318
ScreenTips.TItemImageAlign enumeration 318
ScreenTips.TScreenTipBorderStyle 280
ScreenTips.TScreenTipBorderStyle.Color 281
ScreenTips.TScreenTipBorderStyle.CornerSize 281
ScreenTips.TScreenTipBorderStyle.Create 280
ScreenTips.TScreenTipBorderStyle.Style 281
ScreenTips.TScreenTipItem 282
ScreenTips.TScreenTipItem.Action 286
ScreenTips.TScreenTipItem.Assign 284
ScreenTips.TScreenTipItem.Create 283
ScreenTips.TScreenTipItem.Description 286
ScreenTips.TScreenTipItem.Destroy 283
ScreenTips.TScreenTipItem.DisabledDescription 286
ScreenTips.TScreenTipItem.DisabledHeader 286
ScreenTips.TScreenTipItem.GetDisplayName 284
ScreenTips.TScreenTipItem.Header 286
ScreenTips.TScreenTipItem.Height 287
ScreenTips.TScreenTipItem.Image 287
ScreenTips.TScreenTipItem.IsEnabled 285
ScreenTips.TScreenTipItem.Manager 287
ScreenTips.TScreenTipItem.SetManager 285
ScreenTips.TScreenTipItem.ShowFooter 287
ScreenTips.TScreenTipItem.ShowImage 288
ScreenTips.TScreenTipItem.Width 288
ScreenTips.TScreenTipOption 318
ScreenTips.TScreenTipOption enumeration 318
ScreenTips.TScreenTipOptions 321
ScreenTips.TScreenTipOptions type 321

ScreenTips.TScreenTipPopupType 319
ScreenTips.TScreenTipPopupType enumeration 319
ScreenTips.TScreenTipsCollection 288
ScreenTips.TScreenTipsCollection.Add 289
ScreenTips.TScreenTipsCollection.Create 289
ScreenTips.TScreenTipsCollection.GetOwner 290
ScreenTips.TScreenTipsCollection.Insert 290
ScreenTips.TScreenTipsCollection.Items 291
ScreenTips.TScreenTipsManager 291
ScreenTips.TScreenTipsManager.BorderStyle 293
ScreenTips.TScreenTipsManager.DisplayOffset 293
ScreenTips.TScreenTipsManager.Font 293
ScreenTips.TScreenTipsManager.Footer 294
ScreenTips.TScreenTipsManager.FooterImage 294
ScreenTips.TScreenTipsManager.GradientEndColor 294
ScreenTips.TScreenTipsManager.GradientStartColor 294
ScreenTips.TScreenTipsManager.LinkedActionLists 294
ScreenTips.TScreenTipsManager.OnCreateScreenTip 295
ScreenTips.TScreenTipsManager.OnDuplicateCreateError 296
ScreenTips.TScreenTipsManager.OnGetScreenTipWinClass 296
ScreenTips.TScreenTipsManager.OnScreenTipsLoaded 296
ScreenTips.TScreenTipsManager.OnShowScreenTip 296
ScreenTips.TScreenTipsManager.Options 295
ScreenTips.TScreenTipsManager.ScreenTips 295
ScreenTips.TScreenTipsPopup 297
ScreenTips.TScreenTipsPopup.Align 298
ScreenTips.TScreenTipsPopup.Anchors 301
ScreenTips.TScreenTipsPopup.Associate 302
ScreenTips.TScreenTipsPopup.Glyph 302
ScreenTips.TScreenTipsPopup.PopupType 302
ScreenTips.TScreenTipsPopup.ScreenTip 303
ScreenTips.TScreenTipsPopup.ScreenTipManager 303
ScreenTips.TScreenTipsPopup.Transparent 303
ScreenTips.TScreenTipsPopup.Visible 303
ScreenTips.TScreenTipsWindow 305
ScreenTips.TScreenTipsWindow.ActivateToolTip 307
ScreenTips.TScreenTipsWindow.ActiveItem 311
ScreenTips.TScreenTipsWindow.Body 312
ScreenTips.TScreenTipsWindow.CMMouseenter 307
ScreenTips.TScreenTipsWindow.CMMouseLeave 308
ScreenTips.TScreenTipsWindow.CornerSize 312
ScreenTips.TScreenTipsWindow.Create 306
ScreenTips.TScreenTipsWindow.CreateParams 308
ScreenTips.TScreenTipsWindow.CreateRoundedStyle 308
ScreenTips.TScreenTipsWindow.CreateTipBorderStyle 309
ScreenTips.TScreenTipsWindow.CreateWnd 309
ScreenTips.TScreenTipsWindow.Destroy 306
ScreenTips.TScreenTipsWindow.DrawBackground 309
ScreenTips.TScreenTipsWindow.DrawBorder 310
ScreenTips.TScreenTipsWindow.DrawFooter 310
ScreenTips.TScreenTipsWindow.DrawHeader 310
ScreenTips.TScreenTipsWindow.Footer 312
ScreenTips.TScreenTipsWindow.FooterImage 312
ScreenTips.TScreenTipsWindow.GradientEndColor 313
ScreenTips.TScreenTipsWindow.GradientStartColor 313
ScreenTips.TScreenTipsWindow.HideHint 310
ScreenTips.TScreenTipsWindow.HintControl 313
ScreenTips.TScreenTipsWindow.Image 313
ScreenTips.TScreenTipsWindow.Paint 310
ScreenTips.TScreenTipsWindow.TipBorderStyle 313
ScreenTips.TScreenTipsWindow.WMNCHitTest 311
ScreenTips.TScreenTipsWindowClass 321
ScreenTips.TScreenTipsWindowClass type 321
ScreenTips.TScreenTipWinClass 321
ScreenTips.TScreenTipWinClass type 321
ScreenTips.TShowScreenTipEvent 322
ScreenTips.TShowScreenTipEvent type 322
ScreenTips.TTipBorderStyle 319
ScreenTips.TTipBorderStyle enumeration 319
scTop enumeration member 2911
scTrack enumeration member 2911
seAccept enumeration member 137
seConnect enumeration member 137
seConnecting enumeration member 137
seDisconnect enumeration member 137
seListen enumeration member 137
seLookup enumeration member 137
seRead enumeration member 137

SessColn 327
SessColn namespace 327
Classes 328
Structs, Records, Enums 348
Types 349
Variables 350
SessColn.ESessionsException 328
SessColn.Sessions 350
SessColn.Sessions variable 350
SessColn.TAbstractSessionIDGen 328
SessColn.TAbstractSessionIDGen.NextID 329
SessColn.TAbstractSessionStatusChange 329
SessColn.TAbstractSessionStatusChange.SessionStarted 330
SessColn.TAbstractSessionStatusChange.SessionTerminated 330
SessColn.TAbstractSessionStatusChange.SessionTimeout 331
SessColn.TSessionIDGen 331
SessColn.TSessionIDGen.Create 331
SessColn.TSessionIDGen.NextID 332
SessColn.TSessionItem 332
SessColn.TSessionItem.Assign 334
SessColn.TSessionItem.Create 333
SessColn.TSessionItem.Destroy 333
SessColn.TSessionItem.ExpirationTime 335
SessColn.TSessionItem.Expired 335
SessColn.TSessionItem.ID 336
SessColn.TSessionItem.IsActive 335
SessColn.TSessionItem.Items 336
SessColn.TSessionItem.StartTime 336
SessColn.TSessionItem.StatusFlags 336
SessColn.TSessionItem.Terminated 337
SessColn.TSessionItem.TimeoutMinutes 337
SessColn.TSessionItem.TouchTime 337
SessColn.TSessions 337
SessColn.TSessions.ActiveCount 347
SessColn.TSessions.CheckStatusChange 340
SessColn.TSessions.ConnectActiveSession 340
SessColn.TSessions.ConnectSession 340
SessColn.TSessions.Create 338
SessColn.TSessions.CreateSessionIDGen 341
SessColn.TSessions.Destroy 339
SessColn.TSessions.DisconnectSession 341
SessColn.TSessions.FileName 347
SessColn.TSessions.GetExpirationTime 341
SessColn.TSessions.GetExpired 341
SessColn.TSessions.GetIDCount 341
SessColn.TSessions.GetIDs 342
SessColn.TSessions.GetIsActive 342
SessColn.TSessions.GetItemCount 342
SessColn.TSessions.GetItems 342
SessColn.TSessions.GetSession 343
SessColn.TSessions.GetSessionIDGen 343
SessColn.TSessions.GetStartTime 343
SessColn.TSessions.GetTerminated 343
SessColn.TSessions.GetTimeoutMinutes 343
SessColn.TSessions.GetTouchTime 344
SessColn.TSessions.ItemValue 344
SessColn.TSessions.LoadFromFile 344
SessColn.TSessions.LoadFromStream 344
SessColn.TSessions.MaxSessions 347
SessColn.TSessions.NotifyStatusChange 345
SessColn.TSessions.SaveToFile 345
SessColn.TSessions.SaveToStream 345
SessColn.TSessions.SessionExists 345
SessColn.TSessions.SetItemValue 345
SessColn.TSessions.SetTimeoutMinutes 346
SessColn.TSessions.StartSession 346
SessColn.TSessions.TerminateSession 346
SessColn.TSessions.TouchSession 346
SessColn.TSessionStatusFlag 348
SessColn.TSessionStatusFlag enumeration 348
SessColn.TSessionStatusFlags 349
SessColn.TSessionStatusFlags type 349
seWrite enumeration member 137
sffCurrency enumeration member 3193, 3399, 3480
sffExponent enumeration member 3193, 3399, 3480
sffFixed enumeration member 3193, 3399, 3480
sffGeneral enumeration member 3193, 3399, 3480
sffNumber enumeration member 3193, 3399, 3480

ShadowWnd 351
ShadowWnd namespace 351
Classes 352
Structs, Records, Enums 358
ShadowWnd.TControlSide 358
ShadowWnd.TControlSide enumeration 358
ShadowWnd.TShadowWindow 352
ShadowWnd.TShadowWindow.Control 357
ShadowWnd.TShadowWindow.Create 353
ShadowWnd.TShadowWindow.CreateParams 354
ShadowWnd.TShadowWindow.CreateShadow 353
ShadowWnd.TShadowWindow.Destroy 353
ShadowWnd.TShadowWindow.Notification 355
ShadowWnd.TShadowWindow.Paint 355
ShadowWnd.TShadowWindow.SetBounds 356
ShadowWnd.TShadowWindow.Side 357
ShadowWnd.TShadowWindow.WMEraseBkgnd 356
ShareMem 359
ShareMem namespace 359
Functions 360
ShareMem.DumpBlocks 360
ShareMem.DumpBlocks function 360
ShareMem.GetAllocMemCount 360
ShareMem.GetAllocMemCount function 360
ShareMem.GetAllocMemSize 361
ShareMem.GetAllocMemSize function 361
ShareMem.GetHeapStatus 361
ShareMem.GetHeapStatus function 361
ShareMem.HeapAddRef 361
ShareMem.HeapAddRef function 361
ShareMem.HeapRelease 362
ShareMem.HeapRelease function 362
ShareMem.SysAllocMem 362
ShareMem.SysAllocMem function 362
ShareMem.SysFreeMem 362
ShareMem.SysFreeMem function 362
ShareMem.SysGetMem 363
ShareMem.SysGetMem function 363
ShareMem.SysReallocMem 363
ShareMem.SysReallocMem function 363
ShareMem.SysRegisterExpectedMemoryLeak 363
ShareMem.SysRegisterExpectedMemoryLeak function 363
ShareMem.SysUnregisterExpectedMemoryLeak 364
ShareMem.SysUnregisterExpectedMemoryLeak function 364
SHDocVw 365
SHDocVw namespace 365
Classes 371
Constants 780
Interfaces 571
Types 746
SHDocVw._SearchAssistantEvents 571
SHDocVw._SearchAssistantEvents.OnNewSearch 572
SHDocVw._SearchAssistantEvents.OnNextMenuSelect 572
SHDocVw.BrowserNavConstants 748
SHDocVw.BrowserNavConstants type 748
SHDocVw.CGID_IWebBrowser 782
SHDocVw.CGID_IWebBrowser constant 782
SHDocVw.CLASS_CScriptErrorList 783
SHDocVw.CLASS_CScriptErrorList constant 783
SHDocVw.CLASS_InternetExplorer 783
SHDocVw.CLASS_InternetExplorer constant 783
SHDocVw.CLASS_SearchAssistantOC 783
SHDocVw.CLASS_SearchAssistantOC constant 783
SHDocVw.CLASS_ShellBrowserWindow 784
SHDocVw.CLASS_ShellBrowserWindow constant 784
SHDocVw.CLASS_ShellNameSpace 784
SHDocVw.CLASS_ShellNameSpace constant 784
SHDocVw.CLASS_ShellUIHelper 784
SHDocVw.CLASS_ShellUIHelper constant 784
SHDocVw.CLASS_ShellWindows 785
SHDocVw.CLASS_ShellWindows constant 785
SHDocVw.CLASS_WebBrowser 785
SHDocVw.CLASS_WebBrowser constant 785
SHDocVw.CLASS_WebBrowser_V1 785
SHDocVw.CLASS_WebBrowser_V1 constant 785
SHDocVw.CoCScriptErrorList 371
SHDocVw.CoCScriptErrorList.Create 372
SHDocVw.CoCScriptErrorList.CreateRemote 372

SHDocVw.CoInternetExplorer 372
SHDocVw.CoInternetExplorer.Create 373
SHDocVw.CoInternetExplorer.CreateRemote 373
SHDocVw.CommandStateChangeConstants 748
SHDocVw.CommandStateChangeConstants type 748
SHDocVw.CoSearchAssistantOC 373
SHDocVw.CoSearchAssistantOC.Create 374
SHDocVw.CoSearchAssistantOC.CreateRemote 374
SHDocVw.CoShellBrowserWindow 375
SHDocVw.CoShellBrowserWindow.Create 375
SHDocVw.CoShellBrowserWindow.CreateRemote 376
SHDocVw.CoShellNameSpace 376
SHDocVw.CoShellNameSpace.Create 377
SHDocVw.CoShellNameSpace.CreateRemote 377
SHDocVw.CoShellUIHelper 377
SHDocVw.CoShellUIHelper.Create 378
SHDocVw.CoShellUIHelper.CreateRemote 378
SHDocVw.CoShellWindows 378
SHDocVw.CoShellWindows.Create 379
SHDocVw.CoShellWindows.CreateRemote 379
SHDocVw.CSC_NAVIGATEBACK 786
SHDocVw.CSC_NAVIGATEBACK constant 786
SHDocVw.CSC_NAVIGATEFORWARD 786
SHDocVw.CSC_NAVIGATEFORWARD constant 786
SHDocVw.CSC_UPDATECOMMANDS 786
SHDocVw.CSC_UPDATECOMMANDS constant 786
SHDocVw.CScriptErrorList 748
SHDocVw.CScriptErrorList type 748
SHDocVw.DIID__SearchAssistantEvents 787
SHDocVw.DIID__SearchAssistantEvents constant 787
SHDocVw.DIID_DShellNameSpaceEvents 787
SHDocVw.DIID_DShellNameSpaceEvents constant 787
SHDocVw.DIID_DShellWindowsEvents 787
SHDocVw.DIID_DShellWindowsEvents constant 787
SHDocVw.DIID_DWebBrowserEvents 788
SHDocVw.DIID_DWebBrowserEvents constant 788
SHDocVw.DIID_DWebBrowserEvents2 788
SHDocVw.DIID_DWebBrowserEvents2 constant 788
SHDocVw.DShellNameSpaceEvents 573
SHDocVw.DShellNameSpaceEvents.DoubleClick 573
SHDocVw.DShellNameSpaceEvents.FavoritesSelectionChange 574
SHDocVw.DShellNameSpaceEvents.Initialized 574
SHDocVw.DShellNameSpaceEvents.SelectionChange 574
SHDocVw.DShellWindowsEvents 574
SHDocVw.DShellWindowsEvents.WindowRegistered 575
SHDocVw.DShellWindowsEvents.WindowRevoked 575
SHDocVw.dtlOcxPage 788
SHDocVw.dtlOcxPage constant 788
SHDocVw.dtlServerPage 789
SHDocVw.dtlServerPage constant 789
SHDocVw.DWebBrowserEvents 576
SHDocVw.DWebBrowserEvents.BeforeNavigate 577
SHDocVw.DWebBrowserEvents.CommandStateChange 577
SHDocVw.DWebBrowserEvents.DownloadBegin 577
SHDocVw.DWebBrowserEvents.DownloadComplete 578
SHDocVw.DWebBrowserEvents.FrameBeforeNavigate 578
SHDocVw.DWebBrowserEvents.FrameNavigateComplete 578
SHDocVw.DWebBrowserEvents.FrameNewWindow 578
SHDocVw.DWebBrowserEvents.NavigateComplete 579
SHDocVw.DWebBrowserEvents.NewWindow 579
SHDocVw.DWebBrowserEvents.ProgressChange 579
SHDocVw.DWebBrowserEvents.PropertyChange 579
SHDocVw.DWebBrowserEvents.Quit 579
SHDocVw.DWebBrowserEvents.StatusTextChange 580
SHDocVw.DWebBrowserEvents.TitleChange 580
SHDocVw.DWebBrowserEvents.WindowActivate 580
SHDocVw.DWebBrowserEvents.WindowMove 580
SHDocVw.DWebBrowserEvents.WindowResize 581
SHDocVw.DWebBrowserEvents2 581
SHDocVw.DWebBrowserEvents2.BeforeNavigate2 583
SHDocVw.DWebBrowserEvents2.ClientToHostWindow 583
SHDocVw.DWebBrowserEvents2.CommandStateChange 583
SHDocVw.DWebBrowserEvents2.DocumentComplete 583
SHDocVw.DWebBrowserEvents2.DownloadBegin 584
SHDocVw.DWebBrowserEvents2.DownloadComplete 584
SHDocVw.DWebBrowserEvents2.FileDownload 584
SHDocVw.DWebBrowserEvents2.NavigateComplete2 584
SHDocVw.DWebBrowserEvents2.NavigateError 584

SHDocVw.DWebBrowserEvents2.NewWindow2 585
SHDocVw.DWebBrowserEvents2.OnFullScreen 585
SHDocVw.DWebBrowserEvents2.OnMenuBar 585
SHDocVw.DWebBrowserEvents2.OnQuit 585
SHDocVw.DWebBrowserEvents2.OnStatusBar 586
SHDocVw.DWebBrowserEvents2.OnTheaterMode 586
SHDocVw.DWebBrowserEvents2.OnToolBar 586
SHDocVw.DWebBrowserEvents2.OnVisible 586
SHDocVw.DWebBrowserEvents2.PrintTemplateInstantiation 586
SHDocVw.DWebBrowserEvents2.PrintTemplateTeardown 587
SHDocVw.DWebBrowserEvents2.PrivacyImpactedStateChange 587
SHDocVw.DWebBrowserEvents2.ProgressChange 587
SHDocVw.DWebBrowserEvents2.PropertyChange 587
SHDocVw.DWebBrowserEvents2.SetSecureLockIcon 588
SHDocVw.DWebBrowserEvents2.StatusTextChange 588
SHDocVw.DWebBrowserEvents2.TitleChange 588
SHDocVw.DWebBrowserEvents2.UpdatePageStatus 588
SHDocVw.DWebBrowserEvents2.WindowClosing 588
SHDocVw.DWebBrowserEvents2.WindowSetHeight 589
SHDocVw.DWebBrowserEvents2.WindowSetLeft 589
SHDocVw.DWebBrowserEvents2.WindowSetResizable 589
SHDocVw.DWebBrowserEvents2.WindowSetTop 589
SHDocVw.DWebBrowserEvents2.WindowSetWidth 590
SHDocVw.HTMLID_FIND 789
SHDocVw.HTMLID_FIND constant 789
SHDocVw.HTMLID_OPTIONS 789
SHDocVw.HTMLID_OPTIONS constant 789
SHDocVw.HTMLID_VIEWSOURCE 790
SHDocVw.HTMLID_VIEWSOURCE constant 790
SHDocVw.IID_IScriptErrorList 790
SHDocVw.IID_IScriptErrorList constant 790
SHDocVw.IID_ISearch 790
SHDocVw.IID_ISearch constant 790
SHDocVw.IID_ISearchAssistantOC 791
SHDocVw.IID_ISearchAssistantOC constant 791
SHDocVw.IID_ISearchAssistantOC2 791
SHDocVw.IID_ISearchAssistantOC2 constant 791
SHDocVw.IID_ISearchAssistantOC3 791
SHDocVw.IID_ISearchAssistantOC3 constant 791
SHDocVw.IID_ISearches 792
SHDocVw.IID_ISearches constant 792
SHDocVw.IID_IShellFavoritesNameSpace 792
SHDocVw.IID_IShellFavoritesNameSpace constant 792
SHDocVw.IID_IShellNameSpace 792
SHDocVw.IID_IShellNameSpace constant 792
SHDocVw.IID_IShellUIHelper 793
SHDocVw.IID_IShellUIHelper constant 793
SHDocVw.IID_IShellWindows 793
SHDocVw.IID_IShellWindows constant 793
SHDocVw.IID_IWebBrowser 793
SHDocVw.IID_IWebBrowser constant 793
SHDocVw.IID_IWebBrowser2 794
SHDocVw.IID_IWebBrowser2 constant 794
SHDocVw.IID_IWebBrowserApp 794
SHDocVw.IID_IWebBrowserApp constant 794
SHDocVw.InternetExplorer 749
SHDocVw.InternetExplorer type 749
SHDocVw.IScriptErrorList 590
SHDocVw.IScriptErrorList.advanceError 591
SHDocVw.IScriptErrorList.canAdvanceError 591
SHDocVw.IScriptErrorList.canRetreatError 591
SHDocVw.IScriptErrorList.getAlwaysShowLockState 592
SHDocVw.IScriptErrorList.getDetailsPaneOpen 592
SHDocVw.IScriptErrorList.getErrorChar 592
SHDocVw.IScriptErrorList.getErrorCode 592
SHDocVw.IScriptErrorList.getErrorLine 593
SHDocVw.IScriptErrorList.getErrorMsg 593
SHDocVw.IScriptErrorList.getErrorUrl 593
SHDocVw.IScriptErrorList.getPerErrorDisplay 593
SHDocVw.IScriptErrorList.retreatError 593
SHDocVw.IScriptErrorList.setDetailsPaneOpen 594
SHDocVw.IScriptErrorList.setPerErrorDisplay 594
SHDocVw.IScriptErrorListDisp 594
SHDocVw.IScriptErrorListDisp.advanceError 595
SHDocVw.IScriptErrorListDisp.canAdvanceError 596
SHDocVw.IScriptErrorListDisp.canRetreatError 596

SHDocVw.IScriptErrorListDisp.getAlwaysShowLockState 596
SHDocVw.IScriptErrorListDisp.getDetailsPaneOpen 596
SHDocVw.IScriptErrorListDisp.getErrorChar 596
SHDocVw.IScriptErrorListDisp.getErrorCode 597
SHDocVw.IScriptErrorListDisp.getErrorLine 597
SHDocVw.IScriptErrorListDisp.getErrorMsg 597
SHDocVw.IScriptErrorListDisp.getErrorUrl 597
SHDocVw.IScriptErrorListDisp.getPerErrorDisplay 598
SHDocVw.IScriptErrorListDisp.retreatError 598
SHDocVw.IScriptErrorListDisp.setDetailsPaneOpen 598
SHDocVw.IScriptErrorListDisp.setPerErrorDisplay 598
SHDocVw.ISearch 598
SHDocVw.ISearch.Get_Id 599
SHDocVw.ISearch.Get_Title 600
SHDocVw.ISearch.Get_URL 600
SHDocVw.ISearch.Id 600
SHDocVw.ISearch.Title 600
SHDocVw.ISearch.URL 601
SHDocVw.ISearchAssistantOC 601
SHDocVw.ISearchAssistantOC.AddNextMenuItem 603
SHDocVw.ISearchAssistantOC.ASProvider 609
SHDocVw.ISearchAssistantOC.ASSetting 609
SHDocVw.ISearchAssistantOC.EncodeString 603
SHDocVw.ISearchAssistantOC.EventHandled 610
SHDocVw.ISearchAssistantOC.FindComputer 603
SHDocVw.ISearchAssistantOC.FindFilesOrFolders 604
SHDocVw.ISearchAssistantOC.FindOnWeb 604
SHDocVw.ISearchAssistantOC.FindPeople 604
SHDocVw.ISearchAssistantOC.FindPrinter 604
SHDocVw.ISearchAssistantOC.Get_ASProvider 604
SHDocVw.ISearchAssistantOC.Get_ASSetting 605
SHDocVw.ISearchAssistantOC.Get_InWebFolder 605
SHDocVw.ISearchAssistantOC.Get_SearchAssistantDefault 605
SHDocVw.ISearchAssistantOC.Get_Searches 605
SHDocVw.ISearchAssistantOC.Get_ShellFeaturesEnabled 606
SHDocVw.ISearchAssistantOC.Get_Version 606
SHDocVw.ISearchAssistantOC.GetProperty 606
SHDocVw.ISearchAssistantOC.GetSearchAssistantURL 606
SHDocVw.ISearchAssistantOC.InWebFolder 610
SHDocVw.ISearchAssistantOC.IsRestricted 606
SHDocVw.ISearchAssistantOC.NavigateToDefaultSearch 607
SHDocVw.ISearchAssistantOC.NETDetectNextNavigate 607
SHDocVw.ISearchAssistantOC.NotifySearchSettingsChanged 607
SHDocVw.ISearchAssistantOC.PutFindText 607
SHDocVw.ISearchAssistantOC.PutProperty 608
SHDocVw.ISearchAssistantOC.ResetNextMenu 608
SHDocVw.ISearchAssistantOC.SearchAssistantDefault 610
SHDocVw.ISearchAssistantOC.Searches 610
SHDocVw.ISearchAssistantOC.Set_ASProvider 608
SHDocVw.ISearchAssistantOC.Set_ASSetting 608
SHDocVw.ISearchAssistantOC.Set_EventHandled 608
SHDocVw.ISearchAssistantOC.SetDefaultSearchUrl 609
SHDocVw.ISearchAssistantOC.ShellFeaturesEnabled 611
SHDocVw.ISearchAssistantOC.Version 611
SHDocVw.ISearchAssistantOC2 611
SHDocVw.ISearchAssistantOC2.Get_ShowFindPrinter 613
SHDocVw.ISearchAssistantOC2.ShowFindPrinter 613
SHDocVw.ISearchAssistantOC2Disp 613
SHDocVw.ISearchAssistantOC2Disp.AddNextMenuItem 615
SHDocVw.ISearchAssistantOC2Disp.ASProvider 619
SHDocVw.ISearchAssistantOC2Disp.ASSetting 619
SHDocVw.ISearchAssistantOC2Disp.EncodeString 615
SHDocVw.ISearchAssistantOC2Disp.EventHandled 619
SHDocVw.ISearchAssistantOC2Disp.FindComputer 615
SHDocVw.ISearchAssistantOC2Disp.FindFilesOrFolders 615
SHDocVw.ISearchAssistantOC2Disp.FindOnWeb 616
SHDocVw.ISearchAssistantOC2Disp.FindPeople 616
SHDocVw.ISearchAssistantOC2Disp.FindPrinter 616
SHDocVw.ISearchAssistantOC2Disp.GetProperty 616
SHDocVw.ISearchAssistantOC2Disp.GetSearchAssistantURL 617
SHDocVw.ISearchAssistantOC2Disp.InWebFolder 620
SHDocVw.ISearchAssistantOC2Disp.IsRestricted 617
SHDocVw.ISearchAssistantOC2Disp.NavigateToDefaultSearch 617
SHDocVw.ISearchAssistantOC2Disp.NETDetectNextNavigate 617
SHDocVw.ISearchAssistantOC2Disp.NotifySearchSettingsChanged

617
SHDocVw.ISearchAssistantOC2Disp.PutFindText 618
SHDocVw.ISearchAssistantOC2Disp.PutProperty 618
SHDocVw.ISearchAssistantOC2Disp.ResetNextMenu 618
SHDocVw.ISearchAssistantOC2Disp.SearchAssistantDefault 620
SHDocVw.ISearchAssistantOC2Disp.Searches 620
SHDocVw.ISearchAssistantOC2Disp.SetDefaultSearchUrl 618
SHDocVw.ISearchAssistantOC2Disp.ShellFeaturesEnabled 620
SHDocVw.ISearchAssistantOC2Disp.ShowFindPrinter 620
SHDocVw.ISearchAssistantOC2Disp.Version 621
SHDocVw.ISearchAssistantOC3 621
SHDocVw.ISearchAssistantOC3.Get_SearchCompanionAvailable 623
SHDocVw.ISearchAssistantOC3.Get_UseSearchCompanion 623
SHDocVw.ISearchAssistantOC3.SearchCompanionAvailable 623
SHDocVw.ISearchAssistantOC3.Set_UseSearchCompanion 623
SHDocVw.ISearchAssistantOC3.UseSearchCompanion 624
SHDocVw.ISearchAssistantOC3Disp 624
SHDocVw.ISearchAssistantOC3Disp.AddNextMenuItem 625
SHDocVw.ISearchAssistantOC3Disp.ASProvider 630
SHDocVw.ISearchAssistantOC3Disp.ASSetting 630
SHDocVw.ISearchAssistantOC3Disp.EncodeString 626
SHDocVw.ISearchAssistantOC3Disp.EventHandled 630
SHDocVw.ISearchAssistantOC3Disp.FindComputer 626
SHDocVw.ISearchAssistantOC3Disp.FindFilesOrFolders 626
SHDocVw.ISearchAssistantOC3Disp.FindOnWeb 626
SHDocVw.ISearchAssistantOC3Disp.FindPeople 627
SHDocVw.ISearchAssistantOC3Disp.FindPrinter 627
SHDocVw.ISearchAssistantOC3Disp.GetProperty 627
SHDocVw.ISearchAssistantOC3Disp.GetSearchAssistantURL 627
SHDocVw.ISearchAssistantOC3Disp.InWebFolder 630
SHDocVw.ISearchAssistantOC3Disp.IsRestricted 627
SHDocVw.ISearchAssistantOC3Disp.NavigateToDefaultSearch 628
SHDocVw.ISearchAssistantOC3Disp.NETDetectNextNavigate 628
SHDocVw.ISearchAssistantOC3Disp.NotifySearchSettingsChanged 628
SHDocVw.ISearchAssistantOC3Disp.PutFindText 628
SHDocVw.ISearchAssistantOC3Disp.PutProperty 629
SHDocVw.ISearchAssistantOC3Disp.ResetNextMenu 629
SHDocVw.ISearchAssistantOC3Disp.SearchAssistantDefault 630
SHDocVw.ISearchAssistantOC3Disp.SearchCompanionAvailable 631
SHDocVw.ISearchAssistantOC3Disp.Searches 631
SHDocVw.ISearchAssistantOC3Disp.SetDefaultSearchUrl 629
SHDocVw.ISearchAssistantOC3Disp.ShellFeaturesEnabled 631
SHDocVw.ISearchAssistantOC3Disp.ShowFindPrinter 631
SHDocVw.ISearchAssistantOC3Disp.UseSearchCompanion 632
SHDocVw.ISearchAssistantOC3Disp.Version 632
SHDocVw.ISearchAssistantOCDisp 632
SHDocVw.ISearchAssistantOCDisp.AddNextMenuItem 634
SHDocVw.ISearchAssistantOCDisp.ASProvider 638
SHDocVw.ISearchAssistantOCDisp.ASSetting 638
SHDocVw.ISearchAssistantOCDisp.EncodeString 634
SHDocVw.ISearchAssistantOCDisp.EventHandled 638
SHDocVw.ISearchAssistantOCDisp.FindComputer 634
SHDocVw.ISearchAssistantOCDisp.FindFilesOrFolders 634
SHDocVw.ISearchAssistantOCDisp.FindOnWeb 634
SHDocVw.ISearchAssistantOCDisp.FindPeople 635
SHDocVw.ISearchAssistantOCDisp.FindPrinter 635
SHDocVw.ISearchAssistantOCDisp.GetProperty 635
SHDocVw.ISearchAssistantOCDisp.GetSearchAssistantURL 635
SHDocVw.ISearchAssistantOCDisp.InWebFolder 638
SHDocVw.ISearchAssistantOCDisp.IsRestricted 636
SHDocVw.ISearchAssistantOCDisp.NavigateToDefaultSearch 636
SHDocVw.ISearchAssistantOCDisp.NETDetectNextNavigate 636
SHDocVw.ISearchAssistantOCDisp.NotifySearchSettingsChanged 636
SHDocVw.ISearchAssistantOCDisp.PutFindText 636
SHDocVw.ISearchAssistantOCDisp.PutProperty 637

SHDocVw.ISearchAssistantOCDisp.ResetNextMenu 637
SHDocVw.ISearchAssistantOCDisp.SearchAssistantDefault 638
SHDocVw.ISearchAssistantOCDisp.Searches 639
SHDocVw.ISearchAssistantOCDisp.SetDefaultSearchUrl 637
SHDocVw.ISearchAssistantOCDisp.ShellFeaturesEnabled 639
SHDocVw.ISearchAssistantOCDisp.Version 639
SHDocVw.ISearchDisp 639
SHDocVw.ISearchDisp.Id 640
SHDocVw.ISearchDisp.Title 640
SHDocVw.ISearchDisp.URL 641
SHDocVw.ISearches 641
SHDocVw.ISearches._NewEnum 642
SHDocVw.ISearches.Count 643
SHDocVw.ISearches.Default 643
SHDocVw.ISearches.Get_Count 642
SHDocVw.ISearches.Get_Default 642
SHDocVw.ISearches.Item 642
SHDocVw.ISearchesDisp 643
SHDocVw.ISearchesDisp._NewEnum 644
SHDocVw.ISearchesDisp.Count 645
SHDocVw.ISearchesDisp.Default 645
SHDocVw.ISearchesDisp.Item 644
SHDocVw.IShellFavoritesNameSpace 645
SHDocVw.IShellFavoritesNameSpace.CreateSubscriptionForSelection
647
SHDocVw.IShellFavoritesNameSpace.DeleteSubscriptionForSelection
647
SHDocVw.IShellFavoritesNameSpace.Export 647
SHDocVw.IShellFavoritesNameSpace.Get_SubscriptionsEnabled
647
SHDocVw.IShellFavoritesNameSpace.Import 647
SHDocVw.IShellFavoritesNameSpace.InvokeContextMenuCommand
648
SHDocVw.IShellFavoritesNameSpace.MoveSelectionDown 648
SHDocVw.IShellFavoritesNameSpace.MoveSelectionTo 648
SHDocVw.IShellFavoritesNameSpace.MoveSelectionUp 648
SHDocVw.IShellFavoritesNameSpace.NewFolder 649
SHDocVw.IShellFavoritesNameSpace.ResetSort 649
SHDocVw.IShellFavoritesNameSpace.SetRoot 649
SHDocVw.IShellFavoritesNameSpace.SubscriptionsEnabled
650
SHDocVw.IShellFavoritesNameSpace.Synchronize 649
SHDocVw.IShellFavoritesNameSpaceDisp 650
SHDocVw.IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection
651
SHDocVw.IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection
651
SHDocVw.IShellFavoritesNameSpaceDisp.Export 652
SHDocVw.IShellFavoritesNameSpaceDisp.Import 652
SHDocVw.IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand
652
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionDown
652
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionTo
653
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionUp
653
SHDocVw.IShellFavoritesNameSpaceDisp.NewFolder 653
SHDocVw.IShellFavoritesNameSpaceDisp.ResetSort 653
SHDocVw.IShellFavoritesNameSpaceDisp.SetRoot 653
SHDocVw.IShellFavoritesNameSpaceDisp.SubscriptionsEnabled
654
SHDocVw.IShellFavoritesNameSpaceDisp.Synchronize 654
SHDocVw.IShellNameSpace 654
SHDocVw.IShellNameSpace.Columns 661
SHDocVw.IShellNameSpace.CountViewTypes 662
SHDocVw.IShellNameSpace.Depth 662
SHDocVw.IShellNameSpace.EnumOptions 662
SHDocVw.IShellNameSpace.Expand 656
SHDocVw.IShellNameSpace.Flags 662
SHDocVw.IShellNameSpace.Get_Columns 657
SHDocVw.IShellNameSpace.Get_CountViewTypes 657
SHDocVw.IShellNameSpace.Get_Depth 657
SHDocVw.IShellNameSpace.Get_EnumOptions 657
SHDocVw.IShellNameSpace.Get_Flags 657
SHDocVw.IShellNameSpace.Get_Mode 658
SHDocVw.IShellNameSpace.Get_Root 658
SHDocVw.IShellNameSpace.Get_SelectedItem 658

SHDocVw.IShellNameSpace.Get_TVFlags 658
SHDocVw.IShellNameSpace.Mode 662
SHDocVw.IShellNameSpace.Root 663
SHDocVw.IShellNameSpace.SelectedItem 663
SHDocVw.IShellNameSpace.SelectedItems 659
SHDocVw.IShellNameSpace.Set_Columns 659
SHDocVw.IShellNameSpace.Set_Depth 659
SHDocVw.IShellNameSpace.Set_EnumOptions 659
SHDocVw.IShellNameSpace.Set_Flags 659
SHDocVw.IShellNameSpace.Set_Mode 660
SHDocVw.IShellNameSpace.Set_Root 660
SHDocVw.IShellNameSpace.Set_SelectedItem 660
SHDocVw.IShellNameSpace.Set_TVFlags 660
SHDocVw.IShellNameSpace.SetViewType 661
SHDocVw.IShellNameSpace.TVFlags 663
SHDocVw.IShellNameSpace.UnselectAll 661
SHDocVw.IShellNameSpaceDisp 663
SHDocVw.IShellNameSpaceDisp.Columns 669
SHDocVw.IShellNameSpaceDisp.CountViewTypes 669
SHDocVw.IShellNameSpaceDisp.CreateSubscriptionForSelection 665
SHDocVw.IShellNameSpaceDisp.DeleteSubscriptionForSelection 665
SHDocVw.IShellNameSpaceDisp.Depth 669
SHDocVw.IShellNameSpaceDisp.EnumOptions 669
SHDocVw.IShellNameSpaceDisp.Expand 665
SHDocVw.IShellNameSpaceDisp.Export 665
SHDocVw.IShellNameSpaceDisp.Flags 669
SHDocVw.IShellNameSpaceDisp.Import 666
SHDocVw.IShellNameSpaceDisp.InvokeContextMenuCommand 666
SHDocVw.IShellNameSpaceDisp.Mode 670
SHDocVw.IShellNameSpaceDisp.MoveSelectionDown 666
SHDocVw.IShellNameSpaceDisp.MoveSelectionTo 666
SHDocVw.IShellNameSpaceDisp.MoveSelectionUp 667
SHDocVw.IShellNameSpaceDisp.NewFolder 667
SHDocVw.IShellNameSpaceDisp.ResetSort 667
SHDocVw.IShellNameSpaceDisp.Root 670
SHDocVw.IShellNameSpaceDisp.SelectedItem 670
SHDocVw.IShellNameSpaceDisp.SelectedItems 667
SHDocVw.IShellNameSpaceDisp.SetRoot 667
SHDocVw.IShellNameSpaceDisp.SetViewType 668
SHDocVw.IShellNameSpaceDisp.SubscriptionsEnabled 670
SHDocVw.IShellNameSpaceDisp.Synchronize 668
SHDocVw.IShellNameSpaceDisp.TVFlags 671
SHDocVw.IShellNameSpaceDisp.UnselectAll 668
SHDocVw.IShellUIHelper 671
SHDocVw.IShellUIHelper.AddChannel 672
SHDocVw.IShellUIHelper.AddDesktopComponent 672
SHDocVw.IShellUIHelper.AddFavorite 672
SHDocVw.IShellUIHelper.AutoCompleteAttach 673
SHDocVw.IShellUIHelper.AutoCompleteSaveForm 673
SHDocVw.IShellUIHelper.AutoScan 673
SHDocVw.IShellUIHelper.ImportExportFavorites 673
SHDocVw.IShellUIHelper.IsSubscribed 674
SHDocVw.IShellUIHelper.NavigateAndFind 674
SHDocVw.IShellUIHelper.RefreshOfflineDesktop 674
SHDocVw.IShellUIHelper.ResetFirstBootMode 674
SHDocVw.IShellUIHelper.ResetSafeMode 674
SHDocVw.IShellUIHelper.ShowBrowserUI 675
SHDocVw.IShellUIHelperDisp 675
SHDocVw.IShellUIHelperDisp.AddChannel 676
SHDocVw.IShellUIHelperDisp.AddDesktopComponent 676
SHDocVw.IShellUIHelperDisp.AddFavorite 677
SHDocVw.IShellUIHelperDisp.AutoCompleteAttach 677
SHDocVw.IShellUIHelperDisp.AutoCompleteSaveForm 677
SHDocVw.IShellUIHelperDisp.AutoScan 677
SHDocVw.IShellUIHelperDisp.ImportExportFavorites 677
SHDocVw.IShellUIHelperDisp.IsSubscribed 678
SHDocVw.IShellUIHelperDisp.NavigateAndFind 678
SHDocVw.IShellUIHelperDisp.RefreshOfflineDesktop 678
SHDocVw.IShellUIHelperDisp.ResetFirstBootMode 678
SHDocVw.IShellUIHelperDisp.ResetSafeMode 679
SHDocVw.IShellUIHelperDisp.ShowBrowserUI 679
SHDocVw.IShellWindows 679
SHDocVw.IShellWindows._NewEnum 680
SHDocVw.IShellWindows.Count 683

SHDocVw.IShellWindows.FindWindowSW 680
SHDocVw.IShellWindows.Get_Count 681
SHDocVw.IShellWindows.Item 681
SHDocVw.IShellWindows.OnActivated 681
SHDocVw.IShellWindows.OnCreated 681
SHDocVw.IShellWindows.OnNavigate 682
SHDocVw.IShellWindows.ProcessAttachDetach 682
SHDocVw.IShellWindows.Register 682
SHDocVw.IShellWindows.RegisterPending 682
SHDocVw.IShellWindows.Revoke 682
SHDocVw.IShellWindowsDisp 683
SHDocVw.IShellWindowsDisp._NewEnum 684
SHDocVw.IShellWindowsDisp.Count 687
SHDocVw.IShellWindowsDisp.FindWindowSW 684
SHDocVw.IShellWindowsDisp.Item 685
SHDocVw.IShellWindowsDisp.OnActivated 685
SHDocVw.IShellWindowsDisp.OnCreated 685
SHDocVw.IShellWindowsDisp.OnNavigate 685
SHDocVw.IShellWindowsDisp.ProcessAttachDetach 686
SHDocVw.IShellWindowsDisp.Register 686
SHDocVw.IShellWindowsDisp.RegisterPending 686
SHDocVw.IShellWindowsDisp.Revoke 686
SHDocVw.IWebBrowser 687
SHDocVw.IWebBrowser.Application 695
SHDocVw.IWebBrowser.Busy 695
SHDocVw.IWebBrowser.Container 695
SHDocVw.IWebBrowser.Document 695
SHDocVw.IWebBrowser.Get_Application 689
SHDocVw.IWebBrowser.Get_Busy 689
SHDocVw.IWebBrowser.Get_Container 689
SHDocVw.IWebBrowser.Get_Document 689
SHDocVw.IWebBrowser.Get_Height 690
SHDocVw.IWebBrowser.Get_Left 690
SHDocVw.IWebBrowser.Get_LocationName 690
SHDocVw.IWebBrowser.Get_LocationURL 690
SHDocVw.IWebBrowser.Get_Parent 691
SHDocVw.IWebBrowser.Get_Top 691
SHDocVw.IWebBrowser.Get_TopLevelContainer 691
SHDocVw.IWebBrowser.Get_type_ 691
SHDocVw.IWebBrowser.Get_Width 691
SHDocVw.IWebBrowser.GoBack 692
SHDocVw.IWebBrowser.GoForward 692
SHDocVw.IWebBrowser.GoHome 692
SHDocVw.IWebBrowser.GoSearch 692
SHDocVw.IWebBrowser.Height 696
SHDocVw.IWebBrowser.Left 696
SHDocVw.IWebBrowser.LocationName 696
SHDocVw.IWebBrowser.LocationURL 696
SHDocVw.IWebBrowser.Navigate 693
SHDocVw.IWebBrowser.Parent 696
SHDocVw.IWebBrowser.Refresh 693
SHDocVw.IWebBrowser.Refresh2 693
SHDocVw.IWebBrowser.Set_Height 693
SHDocVw.IWebBrowser.Set_Left 693
SHDocVw.IWebBrowser.Set_Top 694
SHDocVw.IWebBrowser.Set_Width 694
SHDocVw.IWebBrowser.Stop 694
SHDocVw.IWebBrowser.Top 697
SHDocVw.IWebBrowser.TopLevelContainer 697
SHDocVw.IWebBrowser.type_ 697
SHDocVw.IWebBrowser.Width 697
SHDocVw.IWebBrowser2 698
SHDocVw.IWebBrowser2.AddressBar 705
SHDocVw.IWebBrowser2.ExecWB 700
SHDocVw.IWebBrowser2.Get_AddressBar 701
SHDocVw.IWebBrowser2.Get_Offline 701
SHDocVw.IWebBrowser2.Get_ReadyState 701
SHDocVw.IWebBrowser2.Get_RegisterAsBrowser 701
SHDocVw.IWebBrowser2.Get_RegisterAsDropTarget 702
SHDocVw.IWebBrowser2.Get_Resizable 702
SHDocVw.IWebBrowser2.Get_Silent 702
SHDocVw.IWebBrowser2.Get_TheaterMode 702
SHDocVw.IWebBrowser2.Navigate2 702
SHDocVw.IWebBrowser2.Offline 705
SHDocVw.IWebBrowser2.QueryStatusWB 703
SHDocVw.IWebBrowser2.ReadyState 705
SHDocVw.IWebBrowser2.RegisterAsBrowser 706
SHDocVw.IWebBrowser2.RegisterAsDropTarget 706

SHDocVw.IWebBrowser2.Resizable 706
SHDocVw.IWebBrowser2.Set_AddressBar 703
SHDocVw.IWebBrowser2.Set_Offline 703
SHDocVw.IWebBrowser2.Set_RegisterAsBrowser 703
SHDocVw.IWebBrowser2.Set_RegisterAsDropTarget 704
SHDocVw.IWebBrowser2.Set_Resizable 704
SHDocVw.IWebBrowser2.Set_Silent 704
SHDocVw.IWebBrowser2.Set_TheaterMode 704
SHDocVw.IWebBrowser2.ShowBrowserBar 704
SHDocVw.IWebBrowser2.Silent 706
SHDocVw.IWebBrowser2.TheaterMode 707
SHDocVw.IWebBrowser2Disp 707
SHDocVw.IWebBrowser2Disp.AddressBar 713
SHDocVw.IWebBrowser2Disp.Application 713
SHDocVw.IWebBrowser2Disp.Busy 713
SHDocVw.IWebBrowser2Disp.ClientToWindow 709
SHDocVw.IWebBrowser2Disp.Container 714
SHDocVw.IWebBrowser2Disp.Document 714
SHDocVw.IWebBrowser2Disp.ExecWB 709
SHDocVw.IWebBrowser2Disp.FullName 714
SHDocVw.IWebBrowser2Disp.FullScreen 714
SHDocVw.IWebBrowser2Disp.GetProperty 709
SHDocVw.IWebBrowser2Disp.GoBack 709
SHDocVw.IWebBrowser2Disp.GoForward 710
SHDocVw.IWebBrowser2Disp.GoHome 710
SHDocVw.IWebBrowser2Disp.GoSearch 710
SHDocVw.IWebBrowser2Disp.Height 714
SHDocVw.IWebBrowser2Disp.HWnd 715
SHDocVw.IWebBrowser2Disp.Left 715
SHDocVw.IWebBrowser2Disp.LocationName 715
SHDocVw.IWebBrowser2Disp.LocationURL 715
SHDocVw.IWebBrowser2Disp.MenuBar 716
SHDocVw.IWebBrowser2Disp.Name 716
SHDocVw.IWebBrowser2Disp.Navigate 710
SHDocVw.IWebBrowser2Disp.Navigate2 710
SHDocVw.IWebBrowser2Disp.Offline 716
SHDocVw.IWebBrowser2Disp.Parent 716
SHDocVw.IWebBrowser2Disp.Path 716
SHDocVw.IWebBrowser2Disp.PutProperty 711
SHDocVw.IWebBrowser2Disp.QueryStatusWB 711
SHDocVw.IWebBrowser2Disp.Quit 711
SHDocVw.IWebBrowser2Disp.ReadyState 717
SHDocVw.IWebBrowser2Disp.Refresh 711
SHDocVw.IWebBrowser2Disp.Refresh2 712
SHDocVw.IWebBrowser2Disp.RegisterAsBrowser 717
SHDocVw.IWebBrowser2Disp.RegisterAsDropTarget 717
SHDocVw.IWebBrowser2Disp.Resizable 717
SHDocVw.IWebBrowser2Disp.ShowBrowserBar 712
SHDocVw.IWebBrowser2Disp.Silent 718
SHDocVw.IWebBrowser2Disp.StatusBar 718
SHDocVw.IWebBrowser2Disp.StatusText 718
SHDocVw.IWebBrowser2Disp.Stop 712
SHDocVw.IWebBrowser2Disp.TheaterMode 718
SHDocVw.IWebBrowser2Disp.ToolBar 718
SHDocVw.IWebBrowser2Disp.Top 719
SHDocVw.IWebBrowser2Disp.TopLevelContainer 719
SHDocVw.IWebBrowser2Disp.type_ 719
SHDocVw.IWebBrowser2Disp.Visible 719
SHDocVw.IWebBrowser2Disp.Width 720
SHDocVw.IWebBrowserApp 720
SHDocVw.IWebBrowserApp.ClientToWindow 722
SHDocVw.IWebBrowserApp.FullName 727
SHDocVw.IWebBrowserApp.FullScreen 727
SHDocVw.IWebBrowserApp.Get_FullName 722
SHDocVw.IWebBrowserApp.Get_FullScreen 723
SHDocVw.IWebBrowserApp.Get_HWnd 723
SHDocVw.IWebBrowserApp.Get_MenuBar 723
SHDocVw.IWebBrowserApp.Get_Name 723
SHDocVw.IWebBrowserApp.Get_Path 724
SHDocVw.IWebBrowserApp.Get_StatusBar 724
SHDocVw.IWebBrowserApp.Get_StatusText 724
SHDocVw.IWebBrowserApp.Get_ToolBar 724
SHDocVw.IWebBrowserApp.Get_Visible 724
SHDocVw.IWebBrowserApp.GetProperty 725
SHDocVw.IWebBrowserApp.HWnd 727
SHDocVw.IWebBrowserApp.MenuBar 728
SHDocVw.IWebBrowserApp.Name 728
SHDocVw.IWebBrowserApp.Path 728

SHDocVw.IWebBrowserApp.PutProperty 725
SHDocVw.IWebBrowserApp.Quit 725
SHDocVw.IWebBrowserApp.Set_FullScreen 725
SHDocVw.IWebBrowserApp.Set_MenuBar 726
SHDocVw.IWebBrowserApp.Set_StatusBar 726
SHDocVw.IWebBrowserApp.Set_StatusText 726
SHDocVw.IWebBrowserApp.Set_ToolBar 726
SHDocVw.IWebBrowserApp.Set_Visible 726
SHDocVw.IWebBrowserApp.StatusBar 728
SHDocVw.IWebBrowserApp.StatusText 729
SHDocVw.IWebBrowserApp.ToolBar 729
SHDocVw.IWebBrowserApp.Visible 729
SHDocVw.IWebBrowserAppDisp 729
SHDocVw.IWebBrowserAppDisp.Application 734
SHDocVw.IWebBrowserAppDisp.Busy 734
SHDocVw.IWebBrowserAppDisp.ClientToWindow 731
SHDocVw.IWebBrowserAppDisp.Container 734
SHDocVw.IWebBrowserAppDisp.Document 735
SHDocVw.IWebBrowserAppDisp.FullName 735
SHDocVw.IWebBrowserAppDisp.FullScreen 735
SHDocVw.IWebBrowserAppDisp.GetProperty 731
SHDocVw.IWebBrowserAppDisp.GoBack 731
SHDocVw.IWebBrowserAppDisp.GoForward 731
SHDocVw.IWebBrowserAppDisp.GoHome 732
SHDocVw.IWebBrowserAppDisp.GoSearch 732
SHDocVw.IWebBrowserAppDisp.Height 735
SHDocVw.IWebBrowserAppDisp.HWnd 736
SHDocVw.IWebBrowserAppDisp.Left 736
SHDocVw.IWebBrowserAppDisp.LocationName 736
SHDocVw.IWebBrowserAppDisp.LocationURL 736
SHDocVw.IWebBrowserAppDisp.MenuBar 736
SHDocVw.IWebBrowserAppDisp.Name 737
SHDocVw.IWebBrowserAppDisp.Navigate 732
SHDocVw.IWebBrowserAppDisp.Parent 737
SHDocVw.IWebBrowserAppDisp.Path 737
SHDocVw.IWebBrowserAppDisp.PutProperty 732
SHDocVw.IWebBrowserAppDisp.Quit 733
SHDocVw.IWebBrowserAppDisp.Refresh 733
SHDocVw.IWebBrowserAppDisp.Refresh2 733
SHDocVw.IWebBrowserAppDisp.StatusBar 737
SHDocVw.IWebBrowserAppDisp.StatusText 738
SHDocVw.IWebBrowserAppDisp.Stop 733
SHDocVw.IWebBrowserAppDisp.ToolBar 738
SHDocVw.IWebBrowserAppDisp.Top 738
SHDocVw.IWebBrowserAppDisp.TopLevelContainer 738
SHDocVw.IWebBrowserAppDisp.type_ 738
SHDocVw.IWebBrowserAppDisp.Visible 739
SHDocVw.IWebBrowserAppDisp.Width 739
SHDocVw.IWebBrowserDisp 739
SHDocVw.IWebBrowserDisp.Application 743
SHDocVw.IWebBrowserDisp.Busy 743
SHDocVw.IWebBrowserDisp.Container 743
SHDocVw.IWebBrowserDisp.Document 743
SHDocVw.IWebBrowserDisp.GoBack 740
SHDocVw.IWebBrowserDisp.GoForward 741
SHDocVw.IWebBrowserDisp.GoHome 741
SHDocVw.IWebBrowserDisp.GoSearch 741
SHDocVw.IWebBrowserDisp.Height 743
SHDocVw.IWebBrowserDisp.Left 744
SHDocVw.IWebBrowserDisp.LocationName 744
SHDocVw.IWebBrowserDisp.LocationURL 744
SHDocVw.IWebBrowserDisp.Navigate 741
SHDocVw.IWebBrowserDisp.Parent 744
SHDocVw.IWebBrowserDisp.Refresh 741
SHDocVw.IWebBrowserDisp.Refresh2 742
SHDocVw.IWebBrowserDisp.Stop 742
SHDocVw.IWebBrowserDisp.Top 745
SHDocVw.IWebBrowserDisp.TopLevelContainer 745
SHDocVw.IWebBrowserDisp.type_ 745
SHDocVw.IWebBrowserDisp.Width 745
SHDocVw.LIBID_SHDocVw 794
SHDocVw.LIBID_SHDocVw constant 794
SHDocVw.navAllowAutosearch 795
SHDocVw.navAllowAutosearch constant 795
SHDocVw.navBrowserBar 795
SHDocVw.navBrowserBar constant 795
SHDocVw.navNoHistory 795
SHDocVw.navNoHistory constant 795

SHDocVw.navNoReadFromCache 796
SHDocVw.navNoReadFromCache constant 796
SHDocVw.navNoWriteToCache 796
SHDocVw.navNoWriteToCache constant 796
SHDocVw.navOpenInNewWindow 796
SHDocVw.navOpenInNewWindow constant 796
SHDocVw.OLECMDEXECOPT 749
SHDocVw.OLECMDEXECOPT type 749
SHDocVw.OLECMDEXECOPT_DODEFAULT 797
SHDocVw.OLECMDEXECOPT_DODEFAULT constant 797
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER 797
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER constant 797
SHDocVw.OLECMDEXECOPT_PROMPTUSER 797
SHDocVw.OLECMDEXECOPT_PROMPTUSER constant 797
SHDocVw.OLECMDEXECOPT_SHOWHELP 798
SHDocVw.OLECMDEXECOPT_SHOWHELP constant 798
SHDocVw.OLECMDF 750
SHDocVw.OLECMDF type 750
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU 798
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU constant 798
SHDocVw.OLECMDF_ENABLED 798
SHDocVw.OLECMDF_ENABLED constant 798
SHDocVw.OLECMDF_INVISIBLE 799
SHDocVw.OLECMDF_INVISIBLE constant 799
SHDocVw.OLECMDF_LATCHED 799
SHDocVw.OLECMDF_LATCHED constant 799
SHDocVw.OLECMDF_NINCHED 799
SHDocVw.OLECMDF_NINCHED constant 799
SHDocVw.OLECMDF_SUPPORTED 800
SHDocVw.OLECMDF_SUPPORTED constant 800
SHDocVw.OLECMDID 750
SHDocVw.OLECMDID type 750
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS 800
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS constant 800
SHDocVw.OLECMDID_CLEARSELECTION 800
SHDocVw.OLECMDID_CLEARSELECTION constant 800
SHDocVw.OLECMDID_CLOSE 801
SHDocVw.OLECMDID_CLOSE constant 801
SHDocVw.OLECMDID_COPY 801
SHDocVw.OLECMDID_COPY constant 801
SHDocVw.OLECMDID_CUT 801
SHDocVw.OLECMDID_CUT constant 801
SHDocVw.OLECMDID_DELETE 802
SHDocVw.OLECMDID_DELETE constant 802
SHDocVw.OLECMDID_DONTDOWNLOADCSS 802
SHDocVw.OLECMDID_DONTDOWNLOADCSS constant 802
SHDocVw.OLECMDID_ENABLE_INTERACTION 802
SHDocVw.OLECMDID_ENABLE_INTERACTION constant 802
SHDocVw.OLECMDID_FIND 803
SHDocVw.OLECMDID_FIND constant 803
SHDocVw.OLECMDID_GETPRINTTEMPLATE 803
SHDocVw.OLECMDID_GETPRINTTEMPLATE constant 803
SHDocVw.OLECMDID_GETZOOMRANGE 803
SHDocVw.OLECMDID_GETZOOMRANGE constant 803
SHDocVw.OLECMDID_HIDETOOLBARS 804
SHDocVw.OLECMDID_HIDETOOLBARS constant 804
SHDocVw.OLECMDID_HTTPEQUIV 804
SHDocVw.OLECMDID_HTTPEQUIV constant 804
SHDocVw.OLECMDID_HTTPEQUIV_DONE 804
SHDocVw.OLECMDID_HTTPEQUIV_DONE constant 804
SHDocVw.OLECMDID_NEW 805
SHDocVw.OLECMDID_NEW constant 805
SHDocVw.OLECMDID_ONTOOLBARACTIVATED 805
SHDocVw.OLECMDID_ONTOOLBARACTIVATED constant 805
SHDocVw.OLECMDID_ONUNLOAD 805
SHDocVw.OLECMDID_ONUNLOAD constant 805
SHDocVw.OLECMDID_OPEN 806
SHDocVw.OLECMDID_OPEN constant 806
SHDocVw.OLECMDID_PAGESETUP 806
SHDocVw.OLECMDID_PAGESETUP constant 806
SHDocVw.OLECMDID_PASTE 806
SHDocVw.OLECMDID_PASTE constant 806
SHDocVw.OLECMDID_PASTESPECIAL 807
SHDocVw.OLECMDID_PASTESPECIAL constant 807
SHDocVw.OLECMDID_PREREFRESH 807
SHDocVw.OLECMDID_PREREFRESH constant 807
SHDocVw.OLECMDID_PRINT 807
SHDocVw.OLECMDID_PRINT constant 807

SHDocVw.OLECMDID_PRINT2 808
SHDocVw.OLECMDID_PRINT2 constant 808
SHDocVw.OLECMDID_PRINTPREVIEW 808
SHDocVw.OLECMDID_PRINTPREVIEW constant 808
SHDocVw.OLECMDID_PRINTPREVIEW2 808
SHDocVw.OLECMDID_PRINTPREVIEW2 constant 808
SHDocVw.OLECMDID_PROPERTIES 809
SHDocVw.OLECMDID_PROPERTIES constant 809
SHDocVw.OLECMDID_PROPERTYBAG2 809
SHDocVw.OLECMDID_PROPERTYBAG2 constant 809
SHDocVw.OLECMDID_REDO 809
SHDocVw.OLECMDID_REDO constant 809
SHDocVw.OLECMDID_REFRESH 810
SHDocVw.OLECMDID_REFRESH constant 810
SHDocVw.OLECMDID_SAVE 810
SHDocVw.OLECMDID_SAVE constant 810
SHDocVw.OLECMDID_SAVEAS 810
SHDocVw.OLECMDID_SAVEAS constant 810
SHDocVw.OLECMDID_SAVECOPYAS 811
SHDocVw.OLECMDID_SAVECOPYAS constant 811
SHDocVw.OLECMDID_SELECTALL 811
SHDocVw.OLECMDID_SELECTALL constant 811
SHDocVw.OLECMDID_SETDOWNLOADSTATE 811
SHDocVw.OLECMDID_SETDOWNLOADSTATE constant 811
SHDocVw.OLECMDID_SETPRINTTEMPLATE 812
SHDocVw.OLECMDID_SETPRINTTEMPLATE constant 812
SHDocVw.OLECMDID_SETPROGRESSMAX 812
SHDocVw.OLECMDID_SETPROGRESSMAX constant 812
SHDocVw.OLECMDID_SETPROGRESSPOS 812
SHDocVw.OLECMDID_SETPROGRESSPOS constant 812
SHDocVw.OLECMDID_SETPROGRESSTEXT 813
SHDocVw.OLECMDID_SETPROGRESSTEXT constant 813
SHDocVw.OLECMDID_SETTITLE 813
SHDocVw.OLECMDID_SETTITLE constant 813
SHDocVw.OLECMDID_SHOWFIND 813
SHDocVw.OLECMDID_SHOWFIND constant 813
SHDocVw.OLECMDID_SHOWMESSAGE 814
SHDocVw.OLECMDID_SHOWMESSAGE constant 814
SHDocVw.OLECMDID_SHOWPAGESETUP 814
SHDocVw.OLECMDID_SHOWPAGESETUP constant 814
SHDocVw.OLECMDID_SHOWPRINT 814
SHDocVw.OLECMDID_SHOWPRINT constant 814
SHDocVw.OLECMDID_SHOWSCRIPTERROR 815
SHDocVw.OLECMDID_SHOWSCRIPTERROR constant 815
SHDocVw.OLECMDID_SPELL 815
SHDocVw.OLECMDID_SPELL constant 815
SHDocVw.OLECMDID_STOP 815
SHDocVw.OLECMDID_STOP constant 815
SHDocVw.OLECMDID_STOPDOWNLOAD 816
SHDocVw.OLECMDID_STOPDOWNLOAD constant 816
SHDocVw.OLECMDID_UNDO 816
SHDocVw.OLECMDID_UNDO constant 816
SHDocVw.OLECMDID_UPDATECOMMANDS 816
SHDocVw.OLECMDID_UPDATECOMMANDS constant 816
SHDocVw.OLECMDID_UPDATEPAGESTATUS 817
SHDocVw.OLECMDID_UPDATEPAGESTATUS constant 817
SHDocVw.OLECMDID_ZOOM 817
SHDocVw.OLECMDID_ZOOM constant 817
SHDocVw.POleVariant1 751
SHDocVw.POleVariant1 type 751
SHDocVw.READYSTATE_COMPLETE 817
SHDocVw.READYSTATE_COMPLETE constant 817
SHDocVw.READYSTATE_INTERACTIVE 818
SHDocVw.READYSTATE_INTERACTIVE constant 818
SHDocVw.READYSTATE_LOADED 818
SHDocVw.READYSTATE_LOADED constant 818
SHDocVw.READYSTATE_LOADING 818
SHDocVw.READYSTATE_LOADING constant 818
SHDocVw.READYSTATE_UNINITIALIZED 819
SHDocVw.READYSTATE_UNINITIALIZED constant 819
SHDocVw.REFRESH_COMPLETELY 819
SHDocVw.REFRESH_COMPLETELY constant 819
SHDocVw.REFRESH_CONTINUE 819
SHDocVw.REFRESH_CONTINUE constant 819
SHDocVw.REFRESH_IFEXPIRED 820
SHDocVw.REFRESH_IFEXPIRED constant 820
SHDocVw.REFRESH_NORMAL 820
SHDocVw.REFRESH_NORMAL constant 820

SHDocVw.RefreshConstants 751
SHDocVw.RefreshConstants type 751
SHDocVw.SearchAssistantOC 751
SHDocVw.SearchAssistantOC type 751
SHDocVw.SecureLockIconConstants 752
SHDocVw.SecureLockIconConstants type 752
SHDocVw.secureLockIconMixed 820
SHDocVw.secureLockIconMixed constant 820
SHDocVw.secureLockIconSecure128Bit 821
SHDocVw.secureLockIconSecure128Bit constant 821
SHDocVw.secureLockIconSecure40Bit 821
SHDocVw.secureLockIconSecure40Bit constant 821
SHDocVw.secureLockIconSecure56Bit 821
SHDocVw.secureLockIconSecure56Bit constant 821
SHDocVw.secureLockIconSecureFortezza 822
SHDocVw.secureLockIconSecureFortezza constant 822
SHDocVw.secureLockIconSecureUnknownBits 822
SHDocVw.secureLockIconSecureUnknownBits constant 822
SHDocVw.secureLockIconUnsecure 822
SHDocVw.secureLockIconUnsecure constant 822
SHDocVw.SHDocVwMajorVersion 823
SHDocVw.SHDocVwMajorVersion constant 823
SHDocVw.SHDocVwMinorVersion 823
SHDocVw.SHDocVwMinorVersion constant 823
SHDocVw.ShellBrowserWindow 752
SHDocVw.ShellBrowserWindow type 752
SHDocVw.ShellNameSpace 752
SHDocVw.ShellNameSpace type 752
SHDocVw.ShellUIHelper 753
SHDocVw.ShellUIHelper type 753
SHDocVw.ShellWindowFindWindowOptions 753
SHDocVw.ShellWindowFindWindowOptions type 753
SHDocVw.ShellWindows 753
SHDocVw.ShellWindows type 753
SHDocVw.ShellWindowTypeConstants 754
SHDocVw.ShellWindowTypeConstants type 754
SHDocVw.SWC_3RDPARTY 823
SHDocVw.SWC_3RDPARTY constant 823
SHDocVw.SWC_BROWSER 824
SHDocVw.SWC_BROWSER constant 824
SHDocVw.SWC_CALLBACK 824
SHDocVw.SWC_CALLBACK constant 824
SHDocVw.SWC_EXPLORER 824
SHDocVw.SWC_EXPLORER constant 824
SHDocVw.SWFO_COOKIEPASSED 825
SHDocVw.SWFO_COOKIEPASSED constant 825
SHDocVw.SWFO_INCLUDEPENDING 825
SHDocVw.SWFO_INCLUDEPENDING constant 825
SHDocVw.SWFO_NEEDDISPATCH 825
SHDocVw.SWFO_NEEDDISPATCH constant 825
SHDocVw.tagREADYSTATE 754
SHDocVw.tagREADYSTATE type 754
SHDocVw.TInternetExplorer 379
SHDocVw.TInternetExplorer.AddressBar 407
SHDocVw.TInternetExplorer.Application 407
SHDocVw.TInternetExplorer.Busy 407
SHDocVw.TInternetExplorer.ClientToWindow 386
SHDocVw.TInternetExplorer.Connect 386
SHDocVw.TInternetExplorer.ConnectTo 386
SHDocVw.TInternetExplorer.Container 408
SHDocVw.TInternetExplorer.Create 383
SHDocVw.TInternetExplorer.DefaultInterface 408
SHDocVw.TInternetExplorer.Destroy 383
SHDocVw.TInternetExplorer.Disconnect 387
SHDocVw.TInternetExplorer.Document 408
SHDocVw.TInternetExplorer.ExecWB 387, 388
SHDocVw.TInternetExplorer.FullName 408
SHDocVw.TInternetExplorer.FullScreen 408
SHDocVw.TInternetExplorer.Get_AddressBar 388
SHDocVw.TInternetExplorer.Get_Application 388
SHDocVw.TInternetExplorer.Get_Busy 388
SHDocVw.TInternetExplorer.Get_Container 389
SHDocVw.TInternetExplorer.Get_Document 389
SHDocVw.TInternetExplorer.Get_FullName 389
SHDocVw.TInternetExplorer.Get_FullScreen 389
SHDocVw.TInternetExplorer.Get_Height 389
SHDocVw.TInternetExplorer.Get_HWnd 390
SHDocVw.TInternetExplorer.Get_Left 390

SHDocVw.TInternetExplorer.Get_LocationName 390
SHDocVw.TInternetExplorer.Get_LocationURL 390
SHDocVw.TInternetExplorer.Get_MenuBar 391
SHDocVw.TInternetExplorer.Get_Name 391
SHDocVw.TInternetExplorer.Get_Offline 391
SHDocVw.TInternetExplorer.Get_Parent 391
SHDocVw.TInternetExplorer.Get_Path 391
SHDocVw.TInternetExplorer.Get_ReadyState 392
SHDocVw.TInternetExplorer.Get_RegisterAsBrowser 392
SHDocVw.TInternetExplorer.Get_RegisterAsDropTarget 392
SHDocVw.TInternetExplorer.Get_Resizable 392
SHDocVw.TInternetExplorer.Get_Silent 393
SHDocVw.TInternetExplorer.Get_StatusBar 393
SHDocVw.TInternetExplorer.Get_StatusText 393
SHDocVw.TInternetExplorer.Get_TheaterMode 393
SHDocVw.TInternetExplorer.Get_ToolBar 393
SHDocVw.TInternetExplorer.Get_Top 394
SHDocVw.TInternetExplorer.Get_TopLevelContainer 394
SHDocVw.TInternetExplorer.Get_type_ 394
SHDocVw.TInternetExplorer.Get_Visible 394
SHDocVw.TInternetExplorer.Get_Width 395
SHDocVw.TInternetExplorer.GetProperty 395
SHDocVw.TInternetExplorer.GoBack 395
SHDocVw.TInternetExplorer.GoForward 395
SHDocVw.TInternetExplorer.GoHome 395
SHDocVw.TInternetExplorer.GoSearch 396
SHDocVw.TInternetExplorer.Height 409
SHDocVw.TInternetExplorer.HWnd 409
SHDocVw.TInternetExplorer.InitServerData 396
SHDocVw.TInternetExplorer.InvokeEvent 396
SHDocVw.TInternetExplorer.Left 409
SHDocVw.TInternetExplorer.LocationName 409
SHDocVw.TInternetExplorer.LocationURL 410
SHDocVw.TInternetExplorer.MenuBar 410
SHDocVw.TInternetExplorer.Name 410
SHDocVw.TInternetExplorer.Navigate 397, 398
SHDocVw.TInternetExplorer.Navigate2 399, 400
SHDocVw.TInternetExplorer.Offline 412
SHDocVw.TInternetExplorer.OnBeforeNavigate2 417
SHDocVw.TInternetExplorer.OnClientToHostWindow 417
SHDocVw.TInternetExplorer.OnCommandStateChange 417
SHDocVw.TInternetExplorer.OnDocumentComplete 417
SHDocVw.TInternetExplorer.OnDownloadBegin 417
SHDocVw.TInternetExplorer.OnDownloadComplete 418
SHDocVw.TInternetExplorer.OnFileDownload 418
SHDocVw.TInternetExplorer.OnFullScreen 418
SHDocVw.TInternetExplorer.OnMenuBar 418
SHDocVw.TInternetExplorer.OnNavigateComplete2 419
SHDocVw.TInternetExplorer.OnNavigateError 419
SHDocVw.TInternetExplorer.OnNewWindow2 419
SHDocVw.TInternetExplorer.OnPrintTemplateInstantiation 419
SHDocVw.TInternetExplorer.OnPrintTemplateTeardown 419
SHDocVw.TInternetExplorer.OnPrivacyImpactedStateChange 420
SHDocVw.TInternetExplorer.OnProgressChange 420
SHDocVw.TInternetExplorer.OnPropertyChange 420
SHDocVw.TInternetExplorer.OnQuit 420
SHDocVw.TInternetExplorer.OnSetSecureLockIcon 421
SHDocVw.TInternetExplorer.OnStatusBar 421
SHDocVw.TInternetExplorer.OnStatusTextChange 421
SHDocVw.TInternetExplorer.OnTheaterMode 421
SHDocVw.TInternetExplorer.OnTitleChange 421
SHDocVw.TInternetExplorer.OnToolBar 422
SHDocVw.TInternetExplorer.OnUpdatePageStatus 422
SHDocVw.TInternetExplorer.OnVisible 422
SHDocVw.TInternetExplorer.OnWindowClosing 422
SHDocVw.TInternetExplorer.OnWindowSetHeight 423
SHDocVw.TInternetExplorer.OnWindowSetLeft 423
SHDocVw.TInternetExplorer.OnWindowSetResizable 423
SHDocVw.TInternetExplorer.OnWindowSetTop 423
SHDocVw.TInternetExplorer.OnWindowSetWidth 423
SHDocVw.TInternetExplorer.Parent 412
SHDocVw.TInternetExplorer.Path 413
SHDocVw.TInternetExplorer.PutProperty 400
SHDocVw.TInternetExplorer.QueryStatusWB 400
SHDocVw.TInternetExplorer.Quit 400
SHDocVw.TInternetExplorer.ReadyState 413
SHDocVw.TInternetExplorer.Refresh 400
SHDocVw.TInternetExplorer.Refresh2 401

SHDocVw.TInternetExplorer.RegisterAsBrowser 413
SHDocVw.TInternetExplorer.RegisterAsDropTarget 413
SHDocVw.TInternetExplorer.Resizable 413
SHDocVw.TInternetExplorer.Set_AddressBar 401
SHDocVw.TInternetExplorer.Set_FullScreen 402
SHDocVw.TInternetExplorer.Set_Height 402
SHDocVw.TInternetExplorer.Set_Left 402
SHDocVw.TInternetExplorer.Set_MenuBar 402
SHDocVw.TInternetExplorer.Set_Offline 402
SHDocVw.TInternetExplorer.Set_RegisterAsBrowser 403
SHDocVw.TInternetExplorer.Set_RegisterAsDropTarget 403
SHDocVw.TInternetExplorer.Set_Resizable 403
SHDocVw.TInternetExplorer.Set_Silent 403
SHDocVw.TInternetExplorer.Set_StatusBar 404
SHDocVw.TInternetExplorer.Set_StatusText 404
SHDocVw.TInternetExplorer.Set_TheaterMode 404
SHDocVw.TInternetExplorer.Set_ToolBar 404
SHDocVw.TInternetExplorer.Set_Top 404
SHDocVw.TInternetExplorer.Set_Visible 405
SHDocVw.TInternetExplorer.Set_Width 405
SHDocVw.TInternetExplorer.ShowBrowserBar 405, 406
SHDocVw.TInternetExplorer.Silent 414
SHDocVw.TInternetExplorer.StatusBar 414
SHDocVw.TInternetExplorer.StatusText 414
SHDocVw.TInternetExplorer.Stop 406
SHDocVw.TInternetExplorer.TheaterMode 414
SHDocVw.TInternetExplorer.ToolBar 415
SHDocVw.TInternetExplorer.Top 415
SHDocVw.TInternetExplorer.TopLevelContainer 415
SHDocVw.TInternetExplorer.type_ 415
SHDocVw.TInternetExplorer.Visible 415
SHDocVw.TInternetExplorer.Width 416
SHDocVw.TInternetExplorerBeforeNavigate2 754
SHDocVw.TInternetExplorerBeforeNavigate2 type 754
SHDocVw.TInternetExplorerClientToHostWindow 755
SHDocVw.TInternetExplorerClientToHostWindow type 755
SHDocVw.TInternetExplorerCommandStateChange 755
SHDocVw.TInternetExplorerCommandStateChange type 755
SHDocVw.TInternetExplorerDocumentComplete 755
SHDocVw.TInternetExplorerDocumentComplete type 755
SHDocVw.TInternetExplorerFileDownload 756
SHDocVw.TInternetExplorerFileDownload type 756
SHDocVw.TInternetExplorerNavigateComplete2 756
SHDocVw.TInternetExplorerNavigateComplete2 type 756
SHDocVw.TInternetExplorerNavigateError 756
SHDocVw.TInternetExplorerNavigateError type 756
SHDocVw.TInternetExplorerNewWindow2 757
SHDocVw.TInternetExplorerNewWindow2 type 757
SHDocVw.TInternetExplorerOnFullScreen 757
SHDocVw.TInternetExplorerOnFullScreen type 757
SHDocVw.TInternetExplorerOnMenuBar 757
SHDocVw.TInternetExplorerOnMenuBar type 757
SHDocVw.TInternetExplorerOnStatusBar 758
SHDocVw.TInternetExplorerOnStatusBar type 758
SHDocVw.TInternetExplorerOnTheaterMode 758
SHDocVw.TInternetExplorerOnTheaterMode type 758
SHDocVw.TInternetExplorerOnToolBar 758
SHDocVw.TInternetExplorerOnToolBar type 758
SHDocVw.TInternetExplorerOnVisible 759
SHDocVw.TInternetExplorerOnVisible type 759
SHDocVw.TInternetExplorerPrintTemplateInstantiation 759
SHDocVw.TInternetExplorerPrintTemplateInstantiation type 759
SHDocVw.TInternetExplorerPrintTemplateTeardown 759
SHDocVw.TInternetExplorerPrintTemplateTeardown type 759
SHDocVw.TInternetExplorerPrivacyImpactedStateChange 760
SHDocVw.TInternetExplorerPrivacyImpactedStateChange type 760
SHDocVw.TInternetExplorerProgressChange 760
SHDocVw.TInternetExplorerProgressChange type 760
SHDocVw.TInternetExplorerPropertyChange 760
SHDocVw.TInternetExplorerPropertyChange type 760
SHDocVw.TInternetExplorerSetSecureLockIcon 761
SHDocVw.TInternetExplorerSetSecureLockIcon type 761
SHDocVw.TInternetExplorerStatusTextChange 761
SHDocVw.TInternetExplorerStatusTextChange type 761
SHDocVw.TInternetExplorerTitleChange 761
SHDocVw.TInternetExplorerTitleChange type 761
SHDocVw.TInternetExplorerUpdatePageStatus 762
SHDocVw.TInternetExplorerUpdatePageStatus type 762

SHDocVw.TInternetExplorerWindowClosing 762
SHDocVw.TInternetExplorerWindowClosing type 762
SHDocVw.TInternetExplorerWindowSetHeight 762
SHDocVw.TInternetExplorerWindowSetHeight type 762
SHDocVw.TInternetExplorerWindowSetLeft 763
SHDocVw.TInternetExplorerWindowSetLeft type 763
SHDocVw.TInternetExplorerWindowSetResizable 763
SHDocVw.TInternetExplorerWindowSetResizable type 763
SHDocVw.TInternetExplorerWindowSetTop 763
SHDocVw.TInternetExplorerWindowSetTop type 763
SHDocVw.TInternetExplorerWindowSetWidth 764
SHDocVw.TInternetExplorerWindowSetWidth type 764
SHDocVw.TShellNameSpace 424
SHDocVw.TShellNameSpace.Columns 438
SHDocVw.TShellNameSpace.Connect 428
SHDocVw.TShellNameSpace.ConnectTo 428
SHDocVw.TShellNameSpace.CountViewTypes 438
SHDocVw.TShellNameSpace.Create 425
SHDocVw.TShellNameSpace.CreateSubscriptionForSelection 429
SHDocVw.TShellNameSpace.DefaultInterface 438
SHDocVw.TShellNameSpace.DeleteSubscriptionForSelection 429
SHDocVw.TShellNameSpace.Depth 438
SHDocVw.TShellNameSpace.Destroy 426
SHDocVw.TShellNameSpace.Disconnect 429
SHDocVw.TShellNameSpace.EnumOptions 439
SHDocVw.TShellNameSpace.Expand 430
SHDocVw.TShellNameSpace.Export 430
SHDocVw.TShellNameSpace.Flags 439
SHDocVw.TShellNameSpace.Get_Columns 430
SHDocVw.TShellNameSpace.Get_CountViewTypes 430
SHDocVw.TShellNameSpace.Get_Depth 430
SHDocVw.TShellNameSpace.Get_EnumOptions 431
SHDocVw.TShellNameSpace.Get_Flags 431
SHDocVw.TShellNameSpace.Get_Mode 431
SHDocVw.TShellNameSpace.Get_Root 431
SHDocVw.TShellNameSpace.Get_SelectedItem 432
SHDocVw.TShellNameSpace.Get_SubscriptionsEnabled 432
SHDocVw.TShellNameSpace.Get_TVFlags 432
SHDocVw.TShellNameSpace.Import 432
SHDocVw.TShellNameSpace.InitServerData 432
SHDocVw.TShellNameSpace.InvokeContextMenuCommand 433
SHDocVw.TShellNameSpace.InvokeEvent 433
SHDocVw.TShellNameSpace.Mode 439
SHDocVw.TShellNameSpace.MoveSelectionDown 433
SHDocVw.TShellNameSpace.MoveSelectionTo 434
SHDocVw.TShellNameSpace.MoveSelectionUp 434
SHDocVw.TShellNameSpace.NewFolder 434
SHDocVw.TShellNameSpace.OnDoubleClick 440
SHDocVw.TShellNameSpace.OnFavoritesSelectionChange 441
SHDocVw.TShellNameSpace.OnInitialized 441
SHDocVw.TShellNameSpace.OnSelectionChange 441
SHDocVw.TShellNameSpace.ResetSort 434
SHDocVw.TShellNameSpace.Root 439
SHDocVw.TShellNameSpace.SelectedItem 440
SHDocVw.TShellNameSpace.SelectedItems 435
SHDocVw.TShellNameSpace.Set_Columns 435
SHDocVw.TShellNameSpace.Set_Depth 435
SHDocVw.TShellNameSpace.Set_EnumOptions 435
SHDocVw.TShellNameSpace.Set_Flags 435
SHDocVw.TShellNameSpace.Set_Mode 436
SHDocVw.TShellNameSpace.Set_Root 436
SHDocVw.TShellNameSpace.Set_SelectedItem 436
SHDocVw.TShellNameSpace.Set_TVFlags 436
SHDocVw.TShellNameSpace.SetRoot 437
SHDocVw.TShellNameSpace.SetViewType 437
SHDocVw.TShellNameSpace.SubscriptionsEnabled 440
SHDocVw.TShellNameSpace.Synchronize 437
SHDocVw.TShellNameSpace.TVFlags 440
SHDocVw.TShellNameSpace.UnselectAll 437
SHDocVw.TShellNameSpaceFavoritesSelectionChange 764
SHDocVw.TShellNameSpaceFavoritesSelectionChange type 764
SHDocVw.TShellUIHelper 441
SHDocVw.TShellUIHelper.AddChannel 444
SHDocVw.TShellUIHelper.AddDesktopComponent 445, 446
SHDocVw.TShellUIHelper.AddFavorite 446, 447
SHDocVw.TShellUIHelper.AutoCompleteAttach 447

SHDocVw.TShellUIHelper.AutoCompleteSaveForm 448
SHDocVw.TShellUIHelper.AutoScan 448, 449
SHDocVw.TShellUIHelper.Connect 449
SHDocVw.TShellUIHelper.ConnectTo 449
SHDocVw.TShellUIHelper.Create 442
SHDocVw.TShellUIHelper.DefaultInterface 452
SHDocVw.TShellUIHelper.Destroy 443
SHDocVw.TShellUIHelper.Disconnect 450
SHDocVw.TShellUIHelper.ImportExportFavorites 450
SHDocVw.TShellUIHelper.InitServerData 450
SHDocVw.TShellUIHelper.IsSubscribed 451
SHDocVw.TShellUIHelper.NavigateAndFind 451
SHDocVw.TShellUIHelper.RefreshOfflineDesktop 451
SHDocVw.TShellUIHelper.ResetFirstBootMode 451
SHDocVw.TShellUIHelper.ResetSafeMode 452
SHDocVw.TShellUIHelper.ShowBrowserUI 452
SHDocVw.TShellWindows 452
SHDocVw.TShellWindows._NewEnum 456
SHDocVw.TShellWindows.Connect 456
SHDocVw.TShellWindows.ConnectTo 456
SHDocVw.TShellWindows.Count 461
SHDocVw.TShellWindows.Create 453
SHDocVw.TShellWindows.DefaultInterface 461
SHDocVw.TShellWindows.Destroy 454
SHDocVw.TShellWindows.Disconnect 457
SHDocVw.TShellWindows.FindWindowSW 457
SHDocVw.TShellWindows.Get_Count 457
SHDocVw.TShellWindows.InitServerData 457
SHDocVw.TShellWindows.InvokeEvent 458
SHDocVw.TShellWindows.Item 458, 459
SHDocVw.TShellWindows.OnActivated 459
SHDocVw.TShellWindows.OnCreated 459
SHDocVw.TShellWindows.OnNavigate 459
SHDocVw.TShellWindows.OnWindowRegistered 461
SHDocVw.TShellWindows.OnWindowRevoked 462
SHDocVw.TShellWindows.ProcessAttachDetach 460
SHDocVw.TShellWindows.Register 460
SHDocVw.TShellWindows.RegisterPending 460
SHDocVw.TShellWindows.Revoke 460
SHDocVw.TShellWindowsWindowRegistered 764
SHDocVw.TShellWindowsWindowRegistered type 764
SHDocVw.TShellWindowsWindowRevoked 765
SHDocVw.TShellWindowsWindowRevoked type 765
SHDocVw.TWebBrowser 462
SHDocVw.TWebBrowser.AddressBar 478
SHDocVw.TWebBrowser.Align 478
SHDocVw.TWebBrowser.Anchors 481
SHDocVw.TWebBrowser.Application 482
SHDocVw.TWebBrowser.Busy 482
SHDocVw.TWebBrowser.ClientToWindow 465
SHDocVw.TWebBrowser.Container 483
SHDocVw.TWebBrowser.ControlInterface 483
SHDocVw.TWebBrowser.CreateControl 466
SHDocVw.TWebBrowser.DefaultInterface 484
SHDocVw.TWebBrowser.Document 484
SHDocVw.TWebBrowser.DragCursor 484
SHDocVw.TWebBrowser.DragMode 485
SHDocVw.TWebBrowser.ExecWB 466
SHDocVw.TWebBrowser.FullName 486
SHDocVw.TWebBrowser.FullScreen 487
SHDocVw.TWebBrowser.Get_Application 467
SHDocVw.TWebBrowser.Get_Container 467
SHDocVw.TWebBrowser.Get_Document 467
SHDocVw.TWebBrowser.Get_Parent 467
SHDocVw.TWebBrowser.GetProperty 468
SHDocVw.TWebBrowser.GoBack 468
SHDocVw.TWebBrowser.GoForward 468
SHDocVw.TWebBrowser.GoHome 469
SHDocVw.TWebBrowser.GoSearch 469
SHDocVw.TWebBrowser.HWnd 487
SHDocVw.TWebBrowser.InitControlData 470
SHDocVw.TWebBrowser.LocationName 487
SHDocVw.TWebBrowser.LocationURL 488
SHDocVw.TWebBrowser.MenuBar 488
SHDocVw.TWebBrowser.Name 488
SHDocVw.TWebBrowser.Navigate 470
SHDocVw.TWebBrowser.Navigate2 472
SHDocVw.TWebBrowser.Offline 489

SHDocVw.TWebBrowser.OnBeforeNavigate2 503
SHDocVw.TWebBrowser.OnClientToHostWindow 503
SHDocVw.TWebBrowser.OnCommandStateChange 503
SHDocVw.TWebBrowser.OnDocumentComplete 504
SHDocVw.TWebBrowser.OnDownloadBegin 504
SHDocVw.TWebBrowser.OnDownloadComplete 505
SHDocVw.TWebBrowser.OnDragDrop 505
SHDocVw.TWebBrowser.OnDragOver 508
SHDocVw.TWebBrowser.OnEndDrag 512
SHDocVw.TWebBrowser.OnEnter 513
SHDocVw.TWebBrowser.OnExit 515
SHDocVw.TWebBrowser.OnFileDownload 518
SHDocVw.TWebBrowser.OnFullScreen 518
SHDocVw.TWebBrowser.OnMenuBar 518
SHDocVw.TWebBrowser.OnNavigateComplete2 519
SHDocVw.TWebBrowser.OnNavigateError 519
SHDocVw.TWebBrowser.OnNewWindow2 519
SHDocVw.TWebBrowser.OnPrintTemplateInstantiation 520
SHDocVw.TWebBrowser.OnPrintTemplateTeardown 520
SHDocVw.TWebBrowser.OnPrivacyImpactedStateChange 520
SHDocVw.TWebBrowser.OnProgressChange 520
SHDocVw.TWebBrowser.OnPropertyChange 521
SHDocVw.TWebBrowser.OnQuit 521
SHDocVw.TWebBrowser.OnSetSecureLockIcon 521
SHDocVw.TWebBrowser.OnStartDrag 522
SHDocVw.TWebBrowser.OnStatusBar 522
SHDocVw.TWebBrowser.OnStatusTextChange 523
SHDocVw.TWebBrowser.OnTheaterMode 523
SHDocVw.TWebBrowser.OnTitleChange 523
SHDocVw.TWebBrowser.OnToolBar 524
SHDocVw.TWebBrowser.OnUpdatePageStatus 524
SHDocVw.TWebBrowser.OnVisible 524
SHDocVw.TWebBrowser.OnWindowClosing 524
SHDocVw.TWebBrowser.OnWindowSetHeight 525
SHDocVw.TWebBrowser.OnWindowSetLeft 525
SHDocVw.TWebBrowser.OnWindowSetResizable 525
SHDocVw.TWebBrowser.OnWindowSetTop 525
SHDocVw.TWebBrowser.OnWindowSetWidth 525
SHDocVw.TWebBrowser.Parent 489
SHDocVw.TWebBrowser.ParentShowHint 490
SHDocVw.TWebBrowser.Path 490
SHDocVw.TWebBrowser.PopupMenu 491
SHDocVw.TWebBrowser.PutProperty 473
SHDocVw.TWebBrowser.QueryStatusWB 474
SHDocVw.TWebBrowser.Quit 474
SHDocVw.TWebBrowser.ReadyState 496
SHDocVw.TWebBrowser.Refresh 474
SHDocVw.TWebBrowser.Refresh2 475
SHDocVw.TWebBrowser.RegisterAsBrowser 496
SHDocVw.TWebBrowser.RegisterAsDropTarget 496
SHDocVw.TWebBrowser.Resizable 497
SHDocVw.TWebBrowser.ShowBrowserBar 476
SHDocVw.TWebBrowser.ShowHint 497
SHDocVw.TWebBrowser.Silent 499
SHDocVw.TWebBrowser.StatusBar 499
SHDocVw.TWebBrowser.StatusText 499
SHDocVw.TWebBrowser.Stop 476
SHDocVw.TWebBrowser.TabOrder 499
SHDocVw.TWebBrowser.TabStop 500
SHDocVw.TWebBrowser.TheaterMode 500
SHDocVw.TWebBrowser.ToolBar 501
SHDocVw.TWebBrowser.TopLevelContainer 501
SHDocVw.TWebBrowser.type_ 501
SHDocVw.TWebBrowser.Visible 502
SHDocVw.TWebBrowser_V1 526
SHDocVw.TWebBrowser_V1.Align 534
SHDocVw.TWebBrowser_V1.Anchors 537
SHDocVw.TWebBrowser_V1.Application 538
SHDocVw.TWebBrowser_V1.Busy 538
SHDocVw.TWebBrowser_V1.Container 538
SHDocVw.TWebBrowser_V1.ControlInterface 538
SHDocVw.TWebBrowser_V1.CreateControl 528
SHDocVw.TWebBrowser_V1.DefaultInterface 538
SHDocVw.TWebBrowser_V1.Document 539
SHDocVw.TWebBrowser_V1.DragCursor 539
SHDocVw.TWebBrowser_V1.DragMode 539
SHDocVw.TWebBrowser_V1.Get_Application 528
SHDocVw.TWebBrowser_V1.Get_Container 528

SHDocVw.TWebBrowser_V1.Get_Document 529
SHDocVw.TWebBrowser_V1.Get_Parent 529
SHDocVw.TWebBrowser_V1.GoBack 529
SHDocVw.TWebBrowser_V1.GoForward 529
SHDocVw.TWebBrowser_V1.GoHome 529
SHDocVw.TWebBrowser_V1.GoSearch 530
SHDocVw.TWebBrowser_V1.InitControlData 530
SHDocVw.TWebBrowser_V1.LocationName 541
SHDocVw.TWebBrowser_V1.LocationURL 541
SHDocVw.TWebBrowser_V1.Navigate 530, 531
SHDocVw.TWebBrowser_V1.OnBeforeNavigate 554
SHDocVw.TWebBrowser_V1.OnCommandStateChange 554
SHDocVw.TWebBrowser_V1.OnDownloadBegin 554
SHDocVw.TWebBrowser_V1.OnDownloadComplete 554
SHDocVw.TWebBrowser_V1.OnDragDrop 554
SHDocVw.TWebBrowser_V1.OnDragOver 557
SHDocVw.TWebBrowser_V1.OnEndDrag 561
SHDocVw.TWebBrowser_V1.OnEnter 562
SHDocVw.TWebBrowser_V1.OnExit 564
SHDocVw.TWebBrowser_V1.OnFrameBeforeNavigate 567
SHDocVw.TWebBrowser_V1.OnFrameNavigateComplete 567
SHDocVw.TWebBrowser_V1.OnFrameNewWindow 567
SHDocVw.TWebBrowser_V1.OnNavigateComplete 568
SHDocVw.TWebBrowser_V1.OnNewWindow 568
SHDocVw.TWebBrowser_V1.OnProgressChange 568
SHDocVw.TWebBrowser_V1.OnPropertyChange 568
SHDocVw.TWebBrowser_V1.OnQuit 568
SHDocVw.TWebBrowser_V1.OnStartDrag 569
SHDocVw.TWebBrowser_V1.OnStatusTextChange 569
SHDocVw.TWebBrowser_V1.OnTitleChange 570
SHDocVw.TWebBrowser_V1.OnWindowActivate 570
SHDocVw.TWebBrowser_V1.OnWindowMove 570
SHDocVw.TWebBrowser_V1.OnWindowResize 570
SHDocVw.TWebBrowser_V1.Parent 541
SHDocVw.TWebBrowser_V1.ParentShowHint 543
SHDocVw.TWebBrowser_V1.PopupMenu 543
SHDocVw.TWebBrowser_V1.Refresh 532
SHDocVw.TWebBrowser_V1.Refresh2 532
SHDocVw.TWebBrowser_V1.ShowHint 548
SHDocVw.TWebBrowser_V1.Stop 533
SHDocVw.TWebBrowser_V1.TabOrder 550
SHDocVw.TWebBrowser_V1.TabStop 551
SHDocVw.TWebBrowser_V1.TopLevelContainer 551
SHDocVw.TWebBrowser_V1.type_ 551
SHDocVw.TWebBrowser_V1.Visible 551
SHDocVw.TWebBrowser_V1BeforeNavigate 765
SHDocVw.TWebBrowser_V1BeforeNavigate type 765
SHDocVw.TWebBrowser_V1CommandStateChange 765
SHDocVw.TWebBrowser_V1CommandStateChange type 765
SHDocVw.TWebBrowser_V1FrameBeforeNavigate 766
SHDocVw.TWebBrowser_V1FrameBeforeNavigate type 766
SHDocVw.TWebBrowser_V1FrameNavigateComplete 766
SHDocVw.TWebBrowser_V1FrameNavigateComplete type 766
SHDocVw.TWebBrowser_V1FrameNewWindow 766
SHDocVw.TWebBrowser_V1FrameNewWindow type 766
SHDocVw.TWebBrowser_V1NavigateComplete 767
SHDocVw.TWebBrowser_V1NavigateComplete type 767
SHDocVw.TWebBrowser_V1NewWindow 767
SHDocVw.TWebBrowser_V1NewWindow type 767
SHDocVw.TWebBrowser_V1ProgressChange 768
SHDocVw.TWebBrowser_V1ProgressChange type 768
SHDocVw.TWebBrowser_V1PropertyChange 768
SHDocVw.TWebBrowser_V1PropertyChange type 768
SHDocVw.TWebBrowser_V1Quit 768
SHDocVw.TWebBrowser_V1Quit type 768
SHDocVw.TWebBrowser_V1StatusTextChange 769
SHDocVw.TWebBrowser_V1StatusTextChange type 769
SHDocVw.TWebBrowser_V1TitleChange 769
SHDocVw.TWebBrowser_V1TitleChange type 769
SHDocVw.TWebBrowserBeforeNavigate2 769
SHDocVw.TWebBrowserBeforeNavigate2 type 769
SHDocVw.TWebBrowserClientToHostWindow 770
SHDocVw.TWebBrowserClientToHostWindow type 770
SHDocVw.TWebBrowserCommandStateChange 770
SHDocVw.TWebBrowserCommandStateChange type 770
SHDocVw.TWebBrowserDocumentComplete 770
SHDocVw.TWebBrowserDocumentComplete type 770
SHDocVw.TWebBrowserFileDownload 771

SHDocVw.TWebBrowserFileDownload type 771
SHDocVw.TWebBrowserNavigateComplete2 771
SHDocVw.TWebBrowserNavigateComplete2 type 771
SHDocVw.TWebBrowserNavigateError 771
SHDocVw.TWebBrowserNavigateError type 771
SHDocVw.TWebBrowserNewWindow2 772
SHDocVw.TWebBrowserNewWindow2 type 772
SHDocVw.TWebBrowserOnFullScreen 772
SHDocVw.TWebBrowserOnFullScreen type 772
SHDocVw.TWebBrowserOnMenuBar 772
SHDocVw.TWebBrowserOnMenuBar type 772
SHDocVw.TWebBrowserOnStatusBar 773
SHDocVw.TWebBrowserOnStatusBar type 773
SHDocVw.TWebBrowserOnTheaterMode 773
SHDocVw.TWebBrowserOnTheaterMode type 773
SHDocVw.TWebBrowserOnToolBar 773
SHDocVw.TWebBrowserOnToolBar type 773
SHDocVw.TWebBrowserOnVisible 774
SHDocVw.TWebBrowserOnVisible type 774
SHDocVw.TWebBrowserPrintTemplateInstantiation 774
SHDocVw.TWebBrowserPrintTemplateInstantiation type 774
SHDocVw.TWebBrowserPrintTemplateTeardown 774
SHDocVw.TWebBrowserPrintTemplateTeardown type 774
SHDocVw.TWebBrowserPrivacyImpactedStateChange 775
SHDocVw.TWebBrowserPrivacyImpactedStateChange type 775
SHDocVw.TWebBrowserProgressChange 775
SHDocVw.TWebBrowserProgressChange type 775
SHDocVw.TWebBrowserPropertyChange 775
SHDocVw.TWebBrowserPropertyChange type 775
SHDocVw.TWebBrowserSetSecureLockIcon 776
SHDocVw.TWebBrowserSetSecureLockIcon type 776
SHDocVw.TWebBrowserStatusTextChange 776
SHDocVw.TWebBrowserStatusTextChange type 776
SHDocVw.TWebBrowserTitleChange 776
SHDocVw.TWebBrowserTitleChange type 776
SHDocVw.TWebBrowserUpdatePageStatus 777
SHDocVw.TWebBrowserUpdatePageStatus type 777
SHDocVw.TWebBrowserWindowClosing 777
SHDocVw.TWebBrowserWindowClosing type 777
SHDocVw.TWebBrowserWindowSetHeight 777
SHDocVw.TWebBrowserWindowSetHeight type 777
SHDocVw.TWebBrowserWindowSetLeft 778
SHDocVw.TWebBrowserWindowSetLeft type 778
SHDocVw.TWebBrowserWindowSetResizable 778
SHDocVw.TWebBrowserWindowSetResizable type 778
SHDocVw.TWebBrowserWindowSetTop 778
SHDocVw.TWebBrowserWindowSetTop type 778
SHDocVw.TWebBrowserWindowSetWidth 779
SHDocVw.TWebBrowserWindowSetWidth type 779
SHDocVw.WebBrowser 779
SHDocVw.WebBrowser type 779
SHDocVw.WebBrowser_V1 779
SHDocVw.WebBrowser_V1 type 779
ShellAnimations 827
ShellAnimations namespace 827
    Classes 828
ShellAnimations.TShellResources 828
SimpleDS 829
SimpleDS namespace 829
    Classes 830
SimpleDS.TInternalSQLDataSet 830
SimpleDS.TInternalSQLDataSet.CommandText 833
SimpleDS.TInternalSQLDataSet.CommandType 834
SimpleDS.TInternalSQLDataSet.DataSource 835
SimpleDS.TInternalSQLDataSet.GetMetadata 836
SimpleDS.TInternalSQLDataSet.MaxBlobSize 836
SimpleDS.TInternalSQLDataSet.ParamCheck 837
SimpleDS.TInternalSQLDataSet.Params 837
SimpleDS.TInternalSQLDataSet.SortFieldNames 838
SimpleDS.TSimpleDataSet 838
SimpleDS.TSimpleDataSet.Active 846
SimpleDS.TSimpleDataSet.AfterApplyUpdates 873
SimpleDS.TSimpleDataSet.AfterCancel 873
SimpleDS.TSimpleDataSet.AfterClose 875
SimpleDS.TSimpleDataSet.AfterDelete 877
SimpleDS.TSimpleDataSet.AfterEdit 879
SimpleDS.TSimpleDataSet.AfterExecute 880
SimpleDS.TSimpleDataSet.AfterGetParams 881

SimpleDS.TSimpleDataSet.AfterGetRecords 882
SimpleDS.TSimpleDataSet.AfterInsert 882
SimpleDS.TSimpleDataSet.AfterOpen 883
SimpleDS.TSimpleDataSet.AfterPost 884
SimpleDS.TSimpleDataSet.AfterRefresh 885
SimpleDS.TSimpleDataSet.AfterRowRequest 885
SimpleDS.TSimpleDataSet.AfterScroll 886
SimpleDS.TSimpleDataSet.Aggregates 848
SimpleDS.TSimpleDataSet.AggregatesActive 849
SimpleDS.TSimpleDataSet.AllocConnection 842
SimpleDS.TSimpleDataSet.AllocDataSet 842
SimpleDS.TSimpleDataSet.AllocProvider 843
SimpleDS.TSimpleDataSet.AutoCalcFields 849
SimpleDS.TSimpleDataSet.BeforeApplyUpdates 886
SimpleDS.TSimpleDataSet.BeforeCancel 887
SimpleDS.TSimpleDataSet.BeforeClose 888
SimpleDS.TSimpleDataSet.BeforeDelete 888
SimpleDS.TSimpleDataSet.BeforeEdit 889
SimpleDS.TSimpleDataSet.BeforeExecute 889
SimpleDS.TSimpleDataSet.BeforeGetParams 890
SimpleDS.TSimpleDataSet.BeforeGetRecords 891
SimpleDS.TSimpleDataSet.BeforeInsert 892
SimpleDS.TSimpleDataSet.BeforeOpen 894
SimpleDS.TSimpleDataSet.BeforePost 894
SimpleDS.TSimpleDataSet.BeforeRefresh 895
SimpleDS.TSimpleDataSet.BeforeRowRequest 895
SimpleDS.TSimpleDataSet.BeforeScroll 896
SimpleDS.TSimpleDataSet.Connection 850
SimpleDS.TSimpleDataSet.Constraints 850
SimpleDS.TSimpleDataSet.Create 841
SimpleDS.TSimpleDataSet.DataSet 851
SimpleDS.TSimpleDataSet.Destroy 842
SimpleDS.TSimpleDataSet.DisableStringTrim 851
SimpleDS.TSimpleDataSet.FetchOnDemand 852
SimpleDS.TSimpleDataSet.FieldDefs 853
SimpleDS.TSimpleDataSet.FileName 856
SimpleDS.TSimpleDataSet.Filter 857
SimpleDS.TSimpleDataSet.Filtered 859
SimpleDS.TSimpleDataSet.FilterOptions 859
SimpleDS.TSimpleDataSet.IndexDefs 860
SimpleDS.TSimpleDataSet.IndexFieldNames 861
SimpleDS.TSimpleDataSet.IndexName 862
SimpleDS.TSimpleDataSet.Loaded 843
SimpleDS.TSimpleDataSet.MasterFields 863
SimpleDS.TSimpleDataSet.MasterSource 863
SimpleDS.TSimpleDataSet.Notification 843
SimpleDS.TSimpleDataSet.ObjectView 864
SimpleDS.TSimpleDataSet.OnCalcFields 897
SimpleDS.TSimpleDataSet.OnDeleteError 897
SimpleDS.TSimpleDataSet.OnEditError 898
SimpleDS.TSimpleDataSet.OnFilterRecord 898
SimpleDS.TSimpleDataSet.OnNewRecord 899
SimpleDS.TSimpleDataSet.OnPostError 899
SimpleDS.TSimpleDataSet.OnReconcileError 900
SimpleDS.TSimpleDataSet.OpenCursor 844
SimpleDS.TSimpleDataSet.PacketRecords 865
SimpleDS.TSimpleDataSet.Params 866
SimpleDS.TSimpleDataSet.PSGetCommandText 844
SimpleDS.TSimpleDataSet.ReadOnly 871
SimpleDS.TSimpleDataSet.SetConnection 845
SimpleDS.TSimpleDataSet.StoreDefs 871
SiteComp 901
SiteComp namespace 901
Classes 905
Constants 1072
Functions 1061
Interfaces 919
Structs, Records, Enums 1062
Types 1067
Variables 1069
SiteComp.AdapterActionHTMLElementTypeNames 1072
SiteComp.AdapterActionHTMLElementTypeNames constant 1072
SiteComp.AdapterDisplayHTMLElementTypeNames 1072
SiteComp.AdapterDisplayHTMLElementTypeNames constant 1072
SiteComp.AdapterDisplayViewModeTypeNames 1073
SiteComp.AdapterDisplayViewModeTypeNames constant 1073
SiteComp.AdapterInputHTMLElementTypeNames 1073

SiteComp.AdapterInputHTMLElementTypeNames constant 1073
SiteComp.ExecuteScript 1069
SiteComp.ExecuteScript variable 1069
SiteComp.FullyQualifiedFieldName 1061
SiteComp.FullyQualifiedFieldName function 1061
SiteComp.FullyQualifiedName 1061
SiteComp.FullyQualifiedName function 1061
SiteComp.GetHTMLSampleImage 1069
SiteComp.GetHTMLSampleImage variable 1069
SiteComp.IActionImageProducer 920
SiteComp.IActionImageProducer.GetAdapterEventImage 922
SiteComp.IActionImageProducer.GetDisplayStyle 922
SiteComp.IAdapterAccess 923
SiteComp.IAdapterAccess.HasExecuteAccess 924
SiteComp.IAdapterAccess.HasModifyAccess 924
SiteComp.IAdapterAccess.HasViewAccess 924
SiteComp.IAdapterActionAccess 924
SiteComp.IAdapterActionAccess.HasExecuteAccess 925
SiteComp.IAdapterActionAttributes 925
SiteComp.IAdapterActionAttributes.GetVisible 926
SiteComp.IAdapterActionAttributes.Visible 926
SiteComp.IAdapterDispatcher 927
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue 928
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsHREF 928
SiteComp.IAdapterDispatcher.GetAdapterItemRequestParams 929
SiteComp.IAdapterDispatchParams 929
SiteComp.IAdapterDispatchParams.GetHandler 931
SiteComp.IAdapterDispatchParams.GetObjectIdentifier 931
SiteComp.IAdapterDispatchParams.GetParams 931
SiteComp.IAdapterDispatchParams.GetRequestIdentifier 931
SiteComp.IAdapterDispatchParams.Handler 932
SiteComp.IAdapterDispatchParams.ObjectIdentifier 932
SiteComp.IAdapterDispatchParams.Params 933
SiteComp.IAdapterDispatchParams.RequestIdentifier 933
SiteComp.IAdapterError 934
SiteComp.IAdapterError.ErrorText 936
SiteComp.IAdapterError.Field 936
SiteComp.IAdapterError.GetErrorText 934
SiteComp.IAdapterError.GetField 935
SiteComp.IAdapterError.GetID 935
SiteComp.IAdapterError.ID 936
SiteComp.IAdapterErrors 937
SiteComp.IAdapterErrors.ClearErrors 938
SiteComp.IAdapterErrors.DefineLabel 938
SiteComp.IAdapterErrors.ErrorCount 940
SiteComp.IAdapterErrors.Errors 940
SiteComp.IAdapterErrors.GetError 938
SiteComp.IAdapterErrors.GetErrorCount 939
SiteComp.IAdapterErrors.GetObjectErrors 939
SiteComp.IAdapterErrors.HasObjectErrors 939
SiteComp.IAdapterFieldAccess 940
SiteComp.IAdapterFieldAccess.HasModifyAccess 941
SiteComp.IAdapterFieldAccess.HasViewAccess 941
SiteComp.IAdapterFieldAttributes 941
SiteComp.IAdapterFieldAttributes.GetRequired 942
SiteComp.IAdapterFieldAttributes.GetVisible 942
SiteComp.IAdapterFieldAttributes.Required 943
SiteComp.IAdapterFieldAttributes.Visible 943
SiteComp.IAdapterHiddenField 943
SiteComp.IAdapterHiddenField.GetName 944
SiteComp.IAdapterHiddenField.GetValue 944
SiteComp.IAdapterHiddenField.Name 945
SiteComp.IAdapterHiddenField.Value 945
SiteComp.IAdapterHiddenFields 945
SiteComp.IAdapterHiddenFields.FieldCount 947
SiteComp.IAdapterHiddenFields.Fields 947
SiteComp.IAdapterHiddenFields.GetField 946
SiteComp.IAdapterHiddenFields.GetFieldCount 946
SiteComp.IAdapterItemRequestParams 947
SiteComp.IAdapterItemRequestParams.FailurePage 950
SiteComp.IAdapterItemRequestParams.GetFailurePage 948
SiteComp.IAdapterItemRequestParams.GetParamValues 949
SiteComp.IAdapterItemRequestParams.GetSuccessPage 949
SiteComp.IAdapterItemRequestParams.ParamValues 951

SiteComp.IAdapterItemRequestParams.SetFailurePage 949
SiteComp.IAdapterItemRequestParams.SetSuccessPage 950
SiteComp.IAdapterItemRequestParams.SuccessPage 951
SiteComp.IAdapterMode 951
SiteComp.IAdapterMode.AdapterMode 953
SiteComp.IAdapterMode.GetAdapterMode 952
SiteComp.IAdapterMode.GetNeedAdapterMode 952
SiteComp.IAdapterMode.NeedAdapterMode 953
SiteComp.IAdapterMode.RestoreDefaultAdapterMode 953
SiteComp.IAdapterMode.SetAdapterMode 953
SiteComp.IAdapterRequestHandler 954
SiteComp.IAdapterRequestHandler.CreateRequestContext 955
SiteComp.IAdapterRequestHandler.HandleRequest 955
SiteComp.IAsFormatted 955
SiteComp.IAsFormatted.AsFormatted 956
SiteComp.IDefaultPageFileName 956
SiteComp.IDefaultPageFileName.SetDefaultPageFileName 957
SiteComp.IDEFileManager 1070
SiteComp.IDEFileManager variable 1070
SiteComp.IFieldValuesAdapter 957
SiteComp.IFieldValuesAdapter.GetField 958
SiteComp.IFieldValuesAdapter.GetValueIndex 958
SiteComp.IFindIncludeFile 958
SiteComp.IFindIncludeFile.FindIncludeFile 959
SiteComp.IGetAdapterActions 959
SiteComp.IGetAdapterActions.GetAdapterActions 960
SiteComp.IGetAdapterDisplayCharacteristics 960
SiteComp.IGetAdapterDisplayCharacteristics.GetAdapterDisplayCharacteristics 961
SiteComp.IGetAdapterErrors 961
SiteComp.IGetAdapterErrors.GetAdapterErrors 962
SiteComp.IGetAdapterFields 962
SiteComp.IGetAdapterFields.GetAdapterFields 963
SiteComp.IGetAdapterHiddenFields 963
SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenFields 964
SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenRecordFields 964
SiteComp.IGetAdapterImage 964
SiteComp.IGetAdapterImage.GetAdapterImage 965
SiteComp.IGetAdapterItemRequestParams 965
SiteComp.IGetAdapterItemRequestParams.GetAdapterItemRequestParams 966
SiteComp.IGetAdapterRequestDefaultResponse 966
SiteComp.IGetAdapterRequestDefaultResponse.FailurePage 968
SiteComp.IGetAdapterRequestDefaultResponse.GetFailurePage 967
SiteComp.IGetAdapterRequestDefaultResponse.GetSuccessPage 967
SiteComp.IGetAdapterRequestDefaultResponse.SuccessPage 968
SiteComp.IGetAdapterValuesList 968
SiteComp.IGetAdapterValuesList.GetAdapterValuesList 969
SiteComp.IGetDefaultAction 969
SiteComp.IGetDefaultAction.DefaultAction 970
SiteComp.IGetDefaultAction.GetDefaultAction 970
SiteComp.IGetDesigntimeWarnings 971
SiteComp.IGetDesigntimeWarnings.GetDesigntimeWarnings 971
SiteComp.IGetFieldValuesAdapter 972
SiteComp.IGetFieldValuesAdapter.GetFieldValuesAdapter 972
SiteComp.IGetHTMLStyle 973
SiteComp.IGetHTMLStyle.GetDisplayStyle 974
SiteComp.IGetHTMLStyle.GetInputStyle 974
SiteComp.IGetHTMLStyle.GetViewMode 975
SiteComp.IGetProducerComponent 976
SiteComp.IGetProducerComponent.GetProducerComponent 976
SiteComp.IGetScriptObject 977
SiteComp.IGetScriptObject.GetScriptObject 977
SiteComp.IGetScriptObjectClassKey 978
SiteComp.IGetScriptObjectClassKey.GetScriptObjectClassKey 978
SiteComp.IGetWebAppComponents 979
SiteComp.IGetWebAppComponents.GetAdapterDispatcher 980
SiteComp.IGetWebAppComponents.GetLocateFileService 980
SiteComp.IGetWebAppComponents.GetPageDispatcher 980
SiteComp.IGetWebAppComponents.GetWebApplicationInfo 980

SiteComp.IGetWebAppComponents.GetWebEndUser 980
SiteComp.IGetWebAppComponents.GetWebUserList 981
SiteComp.IGetWebAppComponents.WebApplicationInfo 981
SiteComp.IGetWebAppComponents.WebEndUser 981
SiteComp.IIDEFileManager 982
SiteComp.IIDEFileManager.SearchFile 982
SiteComp.IImageProducer 983
SiteComp.IImageProducer.GetAdapterImage 984
SiteComp.IInternetEnvOptions 984
SiteComp.IInternetEnvOptions.DisableDebugger 986
SiteComp.IInternetEnvOptions.GetDisableDebugging 985
SiteComp.IInternetEnvOptions.GetHTMLExt 985
SiteComp.IInternetEnvOptions.GetSampleImageFile 985
SiteComp.IInternetEnvOptions.GetScriptTimeoutSecs 986
SiteComp.IInternetEnvOptions.HTMLExt 986
SiteComp.IInternetEnvOptions.SampleImageFile 986
SiteComp.IInternetEnvOptions.ScriptTimeoutSecs 987
SiteComp.IInternetExecuteScript 987
SiteComp.IInternetExecuteScript.CanExecuteScript 988
SiteComp.IInternetExecuteScript.ExecuteScript 988
SiteComp.IInternetFileTypes 988
SiteComp.IInternetFileTypes.FindFileExt 989
SiteComp.IInternetScriptDebugger 989
SiteComp.IInternetScriptDebugger.CanDebugScript 990
SiteComp.IInternetScriptDebugger.DebugScript 990
SiteComp.IIsAdapterActionList 990
SiteComp.IIsAdapterActionList.GetIsAdapterActionList 991
SiteComp.IIsAdapterActionList.IsAdapterActionList 991
SiteComp.IIterateIntfSupport 992
SiteComp.IIterateIntfSupport.GetCurrentIntf 992
SiteComp.IIterateObjectSupport 993
SiteComp.IIterateObjectSupport.GetCurrentObject 993
SiteComp.IIteratorIndex 994
SiteComp.IIteratorIndex.GetIteratorIndex 995
SiteComp.IIteratorIndex.InIterator 995
SiteComp.IIteratorIndex.IteratorIndex 995
SiteComp.IIteratorSupport 996
SiteComp.IIteratorSupport.EndIterator 997
SiteComp.IIteratorSupport.NextIteration 997
SiteComp.IIteratorSupport.StartIterator 998
SiteComp.InetFileTypes 1070
SiteComp.InetFileTypes variable 1070
SiteComp.INotifyAdapterChange 998
SiteComp.INotifyAdapterChange.NotifyAdapterChange 999
SiteComp.INotifyList 999
SiteComp.INotifyList.AddNotify 1000
SiteComp.INotifyList.RemoveNotify 1000
SiteComp.InternetEnvOptions 1070
SiteComp.InternetEnvOptions variable 1070
SiteComp.IPageDispatcher 1001
SiteComp.IPageDispatcher.DefaultPageName 1005
SiteComp.IPageDispatcher.DispatchPageName 1002
SiteComp.IPageDispatcher.GetCanView 1003
SiteComp.IPageDispatcher.GetDefaultPageName 1003
SiteComp.IPageDispatcher.GetLoginRequired 1003
SiteComp.IPageDispatcher.PageNameOfRequest 1004
SiteComp.IPageDispatcher.RedirectToPageName 1004
SiteComp.IPageResult 1006
SiteComp.IPageResult.DispatchPage 1006
SiteComp.IPageResult.IncludePage 1007
SiteComp.IPageResult.RedirectToPage 1007
SiteComp.ISessionsService 1007
SiteComp.ISessionsService.CheckStatusChange 1008
SiteComp.ISessionsService.Clear 1008
SiteComp.ISessionsService.NewSession 1009
SiteComp.ISessionsService.NotifyStatusChange 1009
SiteComp.ISessionsServiceConnectID 1010
SiteComp.ISessionsServiceConnectID.Connect 1010
SiteComp.ISessionsServiceConnectID.ConnectActive 1011
SiteComp.ISessionsServicePersist 1011
SiteComp.ISessionsServicePersist.LoadFromFile 1012
SiteComp.ISessionsServicePersist.LoadFromStream 1012
SiteComp.ISessionsServicePersist.SaveToFile 1012
SiteComp.ISessionsServicePersist.SaveToStream 1012
SiteComp.ISessionsServiceStoreID 1013
SiteComp.ISessionsServiceStoreID.ReadSessionID 1013
SiteComp.ISessionsServiceStoreID.WriteSessionID 1014
SiteComp.IValuesListAdapter 1014

SiteComp.IValuesListAdapter.GetListImage 1015
SiteComp.IValuesListAdapter.GetListImageOfValue 1016
SiteComp.IValuesListAdapter.GetListName 1016
SiteComp.IValuesListAdapter.GetListNameOfValue 1016
SiteComp.IValuesListAdapter.GetListValue 1017
SiteComp.IWebActionsList 1017
SiteComp.IWebActionsList.FindAction 1018
SiteComp.IWebActionsList.GetActionsList 1018
SiteComp.IWebActionsList.GetVisibleActions 1019
SiteComp.IWebActionsList.IsActionInUse 1019
SiteComp.IWebActionsList.VisibleActions 1019
SiteComp.IWebApplicationInfo 1019
SiteComp.IWebApplicationInfo.ApplicationAdapter 1021
SiteComp.IWebApplicationInfo.ApplicationTitle 1022
SiteComp.IWebApplicationInfo.GetApplicationAdapter 1021
SiteComp.IWebApplicationInfo.GetApplicationTitle 1021
SiteComp.IWebContainerSubComponent 1022
SiteComp.IWebDataFields 1023
SiteComp.IWebDataFields.FindField 1023
SiteComp.IWebDataFields.GetFieldsList 1024
SiteComp.IWebDataFields.GetVisibleFields 1024
SiteComp.IWebDataFields.IsFieldInUse 1024
SiteComp.IWebDataFields.VisibleFields 1025
SiteComp.IWebEnabled 1025
SiteComp.IWebEnabled.WebEnabled 1026
SiteComp.IWebFormComponentContainer 1026
SiteComp.IWebFormComponentContainer.Count 1027
SiteComp.IWebFormComponentContainer.GetCount 1027
SiteComp.IWebFormComponentContainer.GetItem 1027
SiteComp.IWebFormComponentContainer.Items 1028
SiteComp.IWebGetActionName 1028
SiteComp.IWebGetActionName.ActionName 1029
SiteComp.IWebGetActionName.GetActionName 1029
SiteComp.IWebGetDisplayLabel 1029
SiteComp.IWebGetDisplayLabel.GetDisplayLabel 1030
SiteComp.IWebGetDisplayWidth 1030
SiteComp.IWebGetDisplayWidth.GetDisplayWidth 1031
SiteComp.IWebGetFieldName 1031
SiteComp.IWebGetFieldName.FieldName 1032
SiteComp.IWebGetFieldName.GetFieldName 1032
SiteComp.IWebGetFieldValue 1032
SiteComp.IWebGetFieldValue.GetValue 1033
SiteComp.IWebGetFieldValue.Value 1034
SiteComp.IWebGetFieldValues 1034
SiteComp.IWebGetFieldValues.EndValueIterator 1035
SiteComp.IWebGetFieldValues.GetCurrentValue 1035
SiteComp.IWebGetFieldValues.HasValue 1035
SiteComp.IWebGetFieldValues.NextValueIteration 1036
SiteComp.IWebGetFieldValues.StartValueIterator 1036
SiteComp.IWebGetMaxLength 1037
SiteComp.IWebGetMaxLength.GetMaxLength 1037
SiteComp.IWebGetSessionIDs 1038
SiteComp.IWebGetSessionIDs.GetSessionIDs 1038
SiteComp.IWebImageHREF 1039
SiteComp.IWebImageHREF.WebImageHREF 1040
SiteComp.IWebInputName 1040
SiteComp.IWebInputName.GetInputName 1041
SiteComp.IWebInputName.InputName 1041
SiteComp.IWebIsDefaultAction 1041
SiteComp.IWebIsDefaultAction.IsDefaultAction 1042
SiteComp.IWebIsDefaultField 1042
SiteComp.IWebIsDefaultField.IsDefaultField 1043
SiteComp.IWebSession 1043
SiteComp.IWebSession.Terminate 1044
SiteComp.IWebSessionAttributes 1045
SiteComp.IWebSessionAttributes.ExpirationTime 1048
SiteComp.IWebSessionAttributes.Expired 1048
SiteComp.IWebSessionAttributes.GetExpirationTime 1046
SiteComp.IWebSessionAttributes.GetExpired 1046
SiteComp.IWebSessionAttributes.GetIsActive 1046
SiteComp.IWebSessionAttributes.GetStartTime 1047
SiteComp.IWebSessionAttributes.GetTerminated 1047
SiteComp.IWebSessionAttributes.GetTimeoutMinutes 1047
SiteComp.IWebSessionAttributes.GetTouchTime 1047
SiteComp.IWebSessionAttributes.IsActive 1048
SiteComp.IWebSessionAttributes.SetTimeoutMinutes 1047
SiteComp.IWebSessionAttributes.StartTime 1049
SiteComp.IWebSessionAttributes.Terminated 1049

SiteComp.IWebSessionAttributes.TimeoutMinutes 1049
SiteComp.IWebSessionAttributes.TouchTime 1049
SiteComp.IWebSessionID 1049
SiteComp.IWebSessionID.GetID 1050
SiteComp.IWebSessionID.ID 1051
SiteComp.IWebSessionIDs 1051
SiteComp.IWebSessionIDs.GetID 1052
SiteComp.IWebSessionIDs.GetIDCount 1052
SiteComp.IWebSessionIDs.IDCount 1053
SiteComp.IWebSessionIDs.IDs 1053
SiteComp.IWebSessionValues 1053
SiteComp.IWebSessionValues.GetItemCount 1054
SiteComp.IWebSessionValues.GetItems 1054
SiteComp.IWebSessionValues.GetValue 1055
SiteComp.IWebSessionValues.SetValue 1055
SiteComp.IWebSessionValues.Values 1055
SiteComp.IWebSetActionName 1055
SiteComp.IWebSetActionName.SetActionName 1056
SiteComp.IWebSetFieldName 1056
SiteComp.IWebSetFieldName.SetFieldName 1057
SiteComp.IWebUserList 1057
SiteComp.IWebUserList.CheckAccessRights 1058
SiteComp.IWebUserList.ValidateUser 1059
SiteComp.IWebValueIsEqual 1059
SiteComp.IWebValueIsEqual.IsEqual 1060
SiteComp.ScriptDebugger 1071
SiteComp.ScriptDebugger variable 1071
SiteComp.TAbstractDesigntimeWarnings 905
SiteComp.TAbstractDesigntimeWarnings.AddObject 906
SiteComp.TAbstractDesigntimeWarnings.AddString 906
SiteComp.TAbstractInetFileType 907
SiteComp.TAbstractInetFileType.FileExt 909
SiteComp.TAbstractInetFileType.FileExtCount 909
SiteComp.TAbstractInetFileType.GetExtCount 908
SiteComp.TAbstractInetFileType.GetFileExt 908
SiteComp.TAbstractInetFileType.GetTypeID 909
SiteComp.TAbstractInetFileType.TypeID 910
SiteComp.TAdapterActionHTMLElementType 1062
SiteComp.TAdapterActionHTMLElementType enumeration 1062
SiteComp.TAdapterDisplayCharacteristic 1062
SiteComp.TAdapterDisplayCharacteristic enumeration 1062
SiteComp.TAdapterDisplayCharacteristics 1067
SiteComp.TAdapterDisplayCharacteristics type 1067
SiteComp.TAdapterDisplayHTMLElementType 1063
SiteComp.TAdapterDisplayHTMLElementType enumeration 1063
SiteComp.TAdapterDisplayViewModeType 1063
SiteComp.TAdapterDisplayViewModeType enumeration 1063
SiteComp.TAdapterInputHTMLElementType 1064
SiteComp.TAdapterInputHTMLElementType enumeration 1064
SiteComp.TAdapterItemRequestParamOption 1065
SiteComp.TAdapterItemRequestParamOption enumeration 1065
SiteComp.TAdapterItemRequestParamOptions 1067
SiteComp.TAdapterItemRequestParamOptions type 1067
SiteComp.TDispatchPageFlag 1065
SiteComp.TDispatchPageFlag enumeration 1065
SiteComp.TDispatchPageFlags 1068
SiteComp.TDispatchPageFlags type 1068
SiteComp.TNotifyList 910
SiteComp.TNotifyList.AddNotify 912
SiteComp.TNotifyList.Count 914
SiteComp.TNotifyList.Create 911
SiteComp.TNotifyList.Destroy 912
SiteComp.TNotifyList.GetCount 913
SiteComp.TNotifyList.GetItem 913
SiteComp.TNotifyList.Items 914
SiteComp.TNotifyList.RemoveNotify 913
SiteComp.TQualifyOption 1065
SiteComp.TQualifyOption enumeration 1065
SiteComp.TScriptComponent 915
SiteComp.TScriptComponent.Create 917
SiteComp.TScriptComponent.GetComponent 917
SiteComp.TScriptComponent.Obj 917
SiteComp.TScriptObject 918
SiteComp.TSessionID 1068
SiteComp.TSessionID type 1068
SiteProd 1075
SiteProd namespace 1075

Classes 1076
SiteProd.TAdapterPageProducer 1076
SiteProd.TAdapterPageProducer.HTMLDoc 1079
SiteProd.TAdapterPageProducer.HTMLFile 1080
SiteProd.TAdapterPageProducer.OnAfterGetContent 1082
SiteProd.TAdapterPageProducer.OnBeforeGetContent 1082
SiteProd.TAdapterPageProducer.Styles 1080
SiteProd.TAdapterPageProducer.StylesFile 1081
SiteProd.TAdapterPageProducer.WebPageItems 1081
SiteProd.TBaseAdapterPageProducer 1083
SiteProd.TBaseAdapterPageProducer.CanAddClass 1086
SiteProd.TBaseAdapterPageProducer.ContentFromStream 1087
SiteProd.TBaseAdapterPageProducer.Create 1085
SiteProd.TBaseAdapterPageProducer.CreatePageElements 1087
SiteProd.TBaseAdapterPageProducer.DeclareStyles 1088
SiteProd.TBaseAdapterPageProducer.DeclareWarnings 1088
SiteProd.TBaseAdapterPageProducer.Destroy 1085
SiteProd.TBaseAdapterPageProducer.DoAfterGetContent 1088
SiteProd.TBaseAdapterPageProducer.DoBeforeGetContent 1089
SiteProd.TBaseAdapterPageProducer.DoTagEvent 1089
SiteProd.TBaseAdapterPageProducer.GeneratePageElements 1090
SiteProd.TBaseAdapterPageProducer.GetContentOptions 1090
SiteProd.TBaseAdapterPageProducer.GetDefaultTemplate 1090
SiteProd.TBaseAdapterPageProducer.GetRequiredTags 1091
SiteProd.TBaseAdapterPageProducer.GetScriptEngines 1091
SiteProd.TBaseAdapterPageProducer.GetTemplateFileType 1092
SiteProd.TBaseAdapterPageProducer.GetXMLData 1092
SiteProd.TBaseAdapterPageProducer.GetXSLData 1092
SiteProd.TBaseAdapterPageProducer.HandleScriptTag 1093
SiteProd.TBaseAdapterPageProducer.HasHTMLBrowserView 1093
SiteProd.TBaseAdapterPageProducer.HasScriptView 1094
SiteProd.TBaseAdapterPageProducer.HasXMLBrowserView 1094
SiteProd.TBaseAdapterPageProducer.HasXSLBrowserView 1094
SiteProd.TBaseAdapterPageProducer.ImplCanAddClass 1095
SiteProd.TBaseAdapterPageProducer.OnAfterGetContent 1098
SiteProd.TBaseAdapterPageProducer.OnBeforeGetContent 1099
SiteProd.TBaseAdapterPageProducer.PageElements 1097
SiteProd.TBaseAdapterPageProducer.SetStyles 1095
SiteProd.TBaseAdapterPageProducer.SetStylesFile 1096
SiteProd.TBaseAdapterPageProducer.SetWebContentOptions 1096
SiteProd.TBaseAdapterPageProducer.Styles 1097
SiteProd.TBaseAdapterPageProducer.StylesFile 1098
SiteProd.TCustomAdapterPageProducer 1099
SiteProd.TCustomAdapterPageProducer.Create 1101
SiteProd.TCustomAdapterPageProducer.Destroy 1102
SiteProd.TCustomAdapterPageProducer.GetTemplateFileName 1102
SiteProd.TCustomAdapterPageProducer.GetTemplateStream 1103
SiteProd.TCustomAdapterPageProducer.HTMLDoc 1105
SiteProd.TCustomAdapterPageProducer.HTMLFile 1106
SiteProd.TCustomAdapterPageProducer.SetHTMLDoc 1103
SiteProd.TCustomAdapterPageProducer.SetHTMLFile 1104
SiteProd.TCustomAdapterPageProducer.WebContent 1104
SiteProd.TSitePageElements 1107
SiteProd.TSitePageElements.ServerFunctionsContent 1108
SiteProd.TSitePageElements.ServerScriptContent 1109
SiteProd.TSitePageElements.ServerVariablesContent 1109
SiteProd.TSitePageElements.StylesContent 1109
SiteProd.TSitePageElements.WarningsContent 1109
SmallString template 3322
[] 3324
= 3324
about SmallString template 3322
AnsiString 3324
SmallString 3323
SmallString constructor 3322
SmallString Members 3322
SmallString operators 3323
SmallStringBase template 3324
about SmallStringBase template 3324
Data 3325

Len 3325
SmallStringBase data members 3325
SmallStringBase Members 3325
SOAPAttachIntf 1111
SOAPAttachIntf namespace 1111
Classes 1112
Interfaces 1116
Types 1125
SOAPAttachIntf.IMimeAttachmentHandler 1116
SOAPAttachIntf.IMimeAttachmentHandler.AddSoapHeader 1117
SOAPAttachIntf.IMimeAttachmentHandler.CreateMimeStream 1118
SOAPAttachIntf.IMimeAttachmentHandler.FinalizeStream 1118
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEBoundary 1119
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEStream 1119
SOAPAttachIntf.IMimeAttachmentHandler.GetOnGetAttachmentEvent 1120
SOAPAttachIntf.IMimeAttachmentHandler.GetOnSendAttachmentEvent 1120
SOAPAttachIntf.IMimeAttachmentHandler.MIMEBoundary 1122
SOAPAttachIntf.IMimeAttachmentHandler.OnGetAttachment 1123
SOAPAttachIntf.IMimeAttachmentHandler.OnSendAttachment 1123
SOAPAttachIntf.IMimeAttachmentHandler.ProcessMultiPartForm 1120
SOAPAttachIntf.IMimeAttachmentHandler.SetMIMEBoundary 1121
SOAPAttachIntf.IMimeAttachmentHandler.SetOnGetAttachmentEvent 1121
SOAPAttachIntf.IMimeAttachmentHandler.SetOnSendAttachmentEvent 1122
SOAPAttachIntf.TOnGetAttachmentEvent 1125
SOAPAttachIntf.TOnGetAttachmentEvent type 1125
SOAPAttachIntf.TOnSendAttachmentEvent 1125
SOAPAttachIntf.TOnSendAttachmentEvent type 1125
SOAPAttachIntf.TSoapDataList 1112
SOAPAttachIntf.TSoapDataList.Add 1113
SOAPAttachIntf.TSoapDataList.Assign 1113
SOAPAttachIntf.TSoapDataList.Clear 1114
SOAPAttachIntf.TSoapDataList.Insert 1114
SOAPAttachIntf.TSoapDataList.Remove 1114
SOAPConn 1127
SOAPConn namespace 1127
Classes 1128
SOAPConn.TSoapConnection 1128
SOAPConn.TSoapConnection.AfterConnect 1141
SOAPConn.TSoapConnection.AfterDisconnect 1141
SOAPConn.TSoapConnection.Agent 1136
SOAPConn.TSoapConnection.BeforeConnect 1142
SOAPConn.TSoapConnection.BeforeDisconnect 1142
SOAPConn.TSoapConnection.Connected 1136
SOAPConn.TSoapConnection.Create 1130
SOAPConn.TSoapConnection.Destroy 1131
SOAPConn.TSoapConnection.DoConnect 1131
SOAPConn.TSoapConnection.DoDisconnect 1132
SOAPConn.TSoapConnection.GetConnected 1132
SOAPConn.TSoapConnection.GetProviderNames 1132
SOAPConn.TSoapConnection.GetServer 1133
SOAPConn.TSoapConnection.GetServerList 1133
SOAPConn.TSoapConnection.GetSOAPServer 1134
SOAPConn.TSoapConnection.GetSOAPServerIID 1134
SOAPConn.TSoapConnection.Notification 1134
SOAPConn.TSoapConnection.OnAfterExecute 1143
SOAPConn.TSoapConnection.OnBeforeExecute 1143
SOAPConn.TSoapConnection.Password 1137
SOAPConn.TSoapConnection.Proxy 1137
SOAPConn.TSoapConnection.ProxyByPass 1137
SOAPConn.TSoapConnection.RIO 1138
SOAPConn.TSoapConnection.SetSOAPServerIID 1135
SOAPConn.TSoapConnection.SetUseSoapAdapter 1135
SOAPConn.TSoapConnection.SOAPServerIID 1138
SOAPConn.TSoapConnection.URL 1139
SOAPConn.TSoapConnection.UserName 1140
SOAPConn.TSoapConnection.UseSOAPAdapter 1140
SOAPDm 1145

SOAPDm namespace 1145
Classes 1146
SOAPDm.TSoapDataModule 1146
SOAPDm.TSoapDataModule._AddRef 1149
SOAPDm.TSoapDataModule._Release 1150
SOAPDm.TSoapDataModule.AfterConstruction 1150
SOAPDm.TSoapDataModule.ApplyUpdates 1151
SOAPDm.TSoapDataModule.AS_ApplyUpdates 1151
SOAPDm.TSoapDataModule.AS_DataRequest 1152
SOAPDm.TSoapDataModule.AS_Execute 1153
SOAPDm.TSoapDataModule.AS_GetParams 1154
SOAPDm.TSoapDataModule.AS_GetProviderNames 1155
SOAPDm.TSoapDataModule.AS_GetRecords 1155
SOAPDm.TSoapDataModule.AS_RowRequest 1156
SOAPDm.TSoapDataModule.BeforeDestruction 1157
SOAPDm.TSoapDataModule.Create 1148
SOAPDm.TSoapDataModule.DataRequest 1157
SOAPDm.TSoapDataModule.Destroy 1148
SOAPDm.TSoapDataModule.Execute 1158
SOAPDm.TSoapDataModule.GetParams 1158
SOAPDm.TSoapDataModule.GetProvider 1159
SOAPDm.TSoapDataModule.GetProviderNames 1159
SOAPDm.TSoapDataModule.GetRecords 1160
SOAPDm.TSoapDataModule.Lock 1161
SOAPDm.TSoapDataModule.NewInstance 1161
SOAPDm.TSoapDataModule.ProviderCount 1168
SOAPDm.TSoapDataModule.Providers 1168
SOAPDm.TSoapDataModule.RefCount 1168
SOAPDm.TSoapDataModule.RegisterProvider 1162
SOAPDm.TSoapDataModule.RowRequest 1162
SOAPDm.TSoapDataModule.SafeCallException 1162
SOAPDm.TSoapDataModule.SAS_ApplyUpdates 1163
SOAPDm.TSoapDataModule.SAS_DataRequest 1163
SOAPDm.TSoapDataModule.SAS_Execute 1164
SOAPDm.TSoapDataModule.SAS_GetParams 1165
SOAPDm.TSoapDataModule.SAS_GetProviderNames 1165
SOAPDm.TSoapDataModule.SAS_GetRecords 1166
SOAPDm.TSoapDataModule.SAS_RowRequest 1166
SOAPDm.TSoapDataModule.Unlock 1167
SOAPDm.TSoapDataModule.UnRegisterProvider 1167
SOAPDomConv 1171
SOAPDomConv namespace 1171
Classes 1172
Interfaces 1179
Structs, Records, Enums 1182
Types 1183
SOAPDomConv.IDOMHeaderProcessor 1179
SOAPDomConv.IDOMHeaderProcessor.CreateHeader 1180
SOAPDomConv.IDOMHeaderProcessor.ProcessHeader 1180
SOAPDomConv.TDOMHeaderProcessorArray 1183
SOAPDomConv.TDOMHeaderProcessorArray type 1183
SOAPDomConv.TDOMHeaderProcessorEntry 1182
SOAPDomConv.TDOMHeaderProcessorEntry record 1182
SOAPDomConv.TSOAPDOMProcessor 1172
SOAPDomConv.TSOAPDOMProcessor._AddRef 1174
SOAPDomConv.TSOAPDOMProcessor._Release 1175
SOAPDomConv.TSOAPDOMProcessor.AddHeaderProcessor 1175
SOAPDomConv.TSOAPDOMProcessor.AfterConstruction 1176
SOAPDomConv.TSOAPDOMProcessor.DefaultProcessHeader 1176
SOAPDomConv.TSOAPDOMProcessor.Destroy 1173
SOAPDomConv.TSOAPDOMProcessor.FHeaderProcessors 1173
SOAPDomConv.TSOAPDOMProcessor.FindHeaderProcessor 1177
SOAPDomConv.TSOAPDOMProcessor.MakeHeaderNodes 1177
SOAPDomConv.TSOAPDOMProcessor.NewInstance 1177
SOAPHTTPClient 1185
SOAPHTTPClient namespace 1185
Classes 1186
SOAPHTTPClient.THTTPRIO 1186
SOAPHTTPClient.THTTPRIO.Converter 1190
SOAPHTTPClient.THTTPRIO.Create 1187
SOAPHTTPClient.THTTPRIO.Destroy 1188
SOAPHTTPClient.THTTPRIO.HTTPWebNode 1190
SOAPHTTPClient.THTTPRIO.Notification 1188
SOAPHTTPClient.THTTPRIO.Port 1191
SOAPHTTPClient.THTTPRIO.QueryInterface 1189

SOAPHTTPClient.THTTPRIO.Service 1191
SOAPHTTPClient.THTTPRIO.URL 1192
SOAPHTTPClient.THTTPRIO.WSDLItems 1193
SOAPHTTPClient.THTTPRIO.WSDLLocation 1193
SOAPHTTPDisp 1195
SOAPHTTPDisp namespace 1195
Classes 1196
Interfaces 1200
SOAPHTTPDisp.IHTTPSoapDispatch 1200
SOAPHTTPDisp.IHTTPSoapDispatch.DispatchSOAP 1201
SOAPHTTPDisp.THTTPSoapDispatchNode 1196
SOAPHTTPDisp.THTTPSoapDispatchNode.Dispatcher 1199
SOAPHTTPDisp.THTTPSoapDispatchNode.DispatchSOAP 1197
SOAPHTTPDisp.THTTPSoapDispatchNode.FSoapDispatcher 1197
SOAPHTTPDisp.THTTPSoapDispatchNode.Notification 1198
SOAPHTTPPasInv 1203
SOAPHTTPPasInv namespace 1203
Classes 1204
SOAPHTTPPasInv.THTTPSoapCppInvoker 1204
SOAPHTTPPasInv.THTTPSoapPascalInvoker 1205
SOAPHTTPPasInv.THTTPSoapPascalInvoker.DispatchSOAP 1207
SOAPHTTPTrans 1209
SOAPHTTPTrans namespace 1209
Classes 1210
Interfaces 1234
Structs, Records, Enums 1236
Types 1237
SOAPHTTPTrans.ESOAPHTTPException 1210
SOAPHTTPTrans.ESOAPHTTPException.Create 1212
SOAPHTTPTrans.ESOAPHTTPException.CreateFmt 1212
SOAPHTTPTrans.ESOAPHTTPException.StatusCode 1213
SOAPHTTPTrans.IHTTPReqResp 1234
SOAPHTTPTrans.IHTTPReqResp.GetHTTPReqResp 1235
SOAPHTTPTrans.SOAPInvokeOptions 1236
SOAPHTTPTrans.SOAPInvokeOptions enumeration 1236
SOAPHTTPTrans.TBeforePostEvent 1237
SOAPHTTPTrans.TBeforePostEvent type 1237
SOAPHTTPTrans.THTTPReqResp 1214
SOAPHTTPTrans.THTTPReqResp._AddRef 1217
SOAPHTTPTrans.THTTPReqResp._Release 1218
SOAPHTTPTrans.THTTPReqResp.AfterConstruction 1218
SOAPHTTPTrans.THTTPReqResp.Agent 1226
SOAPHTTPTrans.THTTPReqResp.BeforeExecute 1219
SOAPHTTPTrans.THTTPReqResp.Check 1219
SOAPHTTPTrans.THTTPReqResp.CheckContentType 1220
SOAPHTTPTrans.THTTPReqResp.Connect 1220
SOAPHTTPTrans.THTTPReqResp.ConnectTimeout 1226
SOAPHTTPTrans.THTTPReqResp.Create 1216
SOAPHTTPTrans.THTTPReqResp.Destroy 1216
SOAPHTTPTrans.THTTPReqResp.Execute 1221
SOAPHTTPTrans.THTTPReqResp.Get 1222
SOAPHTTPTrans.THTTPReqResp.GetHTTPReqResp 1222
SOAPHTTPTrans.THTTPReqResp.GetMimeBoundary 1222
SOAPHTTPTrans.THTTPReqResp.HandleWinInetError 1223
SOAPHTTPTrans.THTTPReqResp.InvokeOptions 1226
SOAPHTTPTrans.THTTPReqResp.MaxSinglePostSize 1227
SOAPHTTPTrans.THTTPReqResp.NewInstance 1223
SOAPHTTPTrans.THTTPReqResp.OnBeforePost 1232
SOAPHTTPTrans.THTTPReqResp.OnPostingData 1232
SOAPHTTPTrans.THTTPReqResp.OnReceivingData 1233
SOAPHTTPTrans.THTTPReqResp.OnWinInetError 1233
SOAPHTTPTrans.THTTPReqResp.Password 1227
SOAPHTTPTrans.THTTPReqResp.Proxy 1227
SOAPHTTPTrans.THTTPReqResp.ProxyByPass 1228
SOAPHTTPTrans.THTTPReqResp.Receive 1223
SOAPHTTPTrans.THTTPReqResp.ReceiveTimeout 1228
SOAPHTTPTrans.THTTPReqResp.Send 1224
SOAPHTTPTrans.THTTPReqResp.SendGet 1224
SOAPHTTPTrans.THTTPReqResp.SendTimeout 1229
SOAPHTTPTrans.THTTPReqResp.SetMimeBoundary 1225
SOAPHTTPTrans.THTTPReqResp.SoapAction 1229
SOAPHTTPTrans.THTTPReqResp.UDDIBindingKey 1229
SOAPHTTPTrans.THTTPReqResp.UDDIOperator 1230
SOAPHTTPTrans.THTTPReqResp.URL 1230
SOAPHTTPTrans.THTTPReqResp.UserName 1231
SOAPHTTPTrans.THTTPReqResp.UseUTF8InHeader 1231

SOAPHTTPTrans.THTTPReqResp.WSDLView 1231
SOAPHTTPTrans.TPostingDataEvent 1237
SOAPHTTPTrans.TPostingDataEvent type 1237
SOAPHTTPTrans.TReceivingDataEvent 1238
SOAPHTTPTrans.TReceivingDataEvent type 1238
SOAPHTTPTrans.TSOAPInvokeOptions 1238
SOAPHTTPTrans.TSOAPInvokeOptions type 1238
SOAPHTTPTrans.TWinInetErrorEvent 1238
SOAPHTTPTrans.TWinInetErrorEvent type 1238
SOAPLinked 1239
SOAPLinked namespace 1239
Classes 1241
Interfaces 1268
Structs, Records, Enums 1272
Types 1273
SOAPLinked.ECustomException 1241
SOAPLinked.ECustomException.CatchException 1246
SOAPLinked.ECustomException.Create 1243
SOAPLinked.ECustomException.Destroy 1245
SOAPLinked.ECustomException.InstanceCount 1246
SOAPLinked.ECustomException.ThrowExcept 1246
SOAPLinked.EventKind 1272
SOAPLinked.EventKind enumeration 1272
SOAPLinked.IIntegerTest 1268
SOAPLinked.IIntegerTest.Add 1269
SOAPLinked.IIntegerTest.Doubler 1269
SOAPLinked.IObjectGetter 1269
SOAPLinked.IObjectGetter.GetAsObject 1270
SOAPLinked.IStringTest 1270
SOAPLinked.IStringTest.Append 1271
SOAPLinked.IStringTest.UpCase 1271
SOAPLinked.TCStringEvent 1273
SOAPLinked.TCStringEvent type 1273
SOAPLinked.TCustExCallback 1273
SOAPLinked.TCustExCallback type 1273
SOAPLinked.TFTCompT 1247
SOAPLinked.TFTCompT.CStringEvent 1248
SOAPLinked.TFTCompT.fireEvent 1248
SOAPLinked.TFTCompT.fireEventEx 1248
SOAPLinked.TFTCompT.StringEvent 1248
SOAPLinked.TFTCompT.VStringEvent 1249
SOAPLinked.TFTCompTI 1249
SOAPLinked.TFTCompTI.Add 1251
SOAPLinked.TFTCompTI.Append 1251
SOAPLinked.TFTCompTI.Doubler 1251
SOAPLinked.TFTCompTI.GetAsObject 1252
SOAPLinked.TFTCompTI.UpCase 1252
SOAPLinked.TLinkedRIO 1252
SOAPLinked.TLinkedRIO.Create 1253
SOAPLinked.TLinkedRIO.CreateFile 1254
SOAPLinked.TLinkedRIO.Destroy 1254
SOAPLinked.TLinkedRIO.GetResponseStream 1255
SOAPLinked.TLinkedRIO.QueryInterface 1255
SOAPLinked.TLinkedRIO.WebNode 1256
SOAPLinked.TLinkedWebNode 1257
SOAPLinked.TLinkedWebNode.BeforeExecute 1259
SOAPLinked.TLinkedWebNode.Create 1258
SOAPLinked.TLinkedWebNode.Destroy 1258
SOAPLinked.TLinkedWebNode.Execute 1260
SOAPLinked.TLinkedWebNode.GetMimeBoundary 1260
SOAPLinked.TLinkedWebNode.GetResponseStream 1261
SOAPLinked.TLinkedWebNode.Invoker 1262
SOAPLinked.TLinkedWebNode.MimeBoundary 1262
SOAPLinked.TLinkedWebNode.SetMimeBoundary 1261
SOAPLinked.TLogLinkedWebNode 1262
SOAPLinked.TLogLinkedWebNode.Create 1263
SOAPLinked.TLogLinkedWebNode.Destroy 1264
SOAPLinked.TLogLinkedWebNode.Execute 1264
SOAPLinked.TLogLinkedWebNode.GetResponseStream 1265
SOAPLinked.TLogLinkedWebNode.ReqFile 1265
SOAPLinked.TLogLinkedWebNode.RespFile 1265
SOAPLinked.TStringEvent 1274
SOAPLinked.TStringEvent type 1274
SOAPLinked.TVStringEvent 1274
SOAPLinked.TVStringEvent type 1274
SOAPLinked.TWinHandleUser 1266
SOAPLinked.TWinHandleUser.UseHandle1 1267
SOAPLinked.TWinHandleUser.UseHandle2 1267

SOAPMidas 1275
SOAPMidas namespace 1275
Functions 1281
Interfaces 1276
SOAPMidas.IAppServerSOAP 1276
SOAPMidas.IAppServerSOAP.SAS_ApplyUpdates 1277
SOAPMidas.IAppServerSOAP.SAS_DataRequest 1277
SOAPMidas.IAppServerSOAP.SAS_Execute 1278
SOAPMidas.IAppServerSOAP.SAS_GetParams 1279
SOAPMidas.IAppServerSOAP.SAS_GetProviderNames 1279
SOAPMidas.IAppServerSOAP.SAS_GetRecords 1279
SOAPMidas.IAppServerSOAP.SAS_RowRequest 1280
SOAPMidas.RegDefIAppServerInvClass 1281
SOAPMidas.RegDefIAppServerInvClass function 1281
SOAPPasInv 1283
SOAPPasInv namespace 1283
Classes 1284
Functions 1296
Interfaces 1293
Types 1297
SOAPPasInv.GetSOAPHeaders 1296
SOAPPasInv.GetSOAPHeaders function 1296
SOAPPasInv.IAdapterConverter 1293
SOAPPasInv.IAdapterConverter.GetDomConverter 1294
SOAPPasInv.IInvokerConverter 1294
SOAPPasInv.IInvokerConverter.GetInvoker 1295
SOAPPasInv.TAfterDispatchEvent 1297
SOAPPasInv.TAfterDispatchEvent type 1297
SOAPPasInv.TBeforeDispatchEvent 1297
SOAPPasInv.TBeforeDispatchEvent type 1297
SOAPPasInv.TBeforeDispatchEvent2 1298
SOAPPasInv.TBeforeDispatchEvent2 type 1298
SOAPPasInv.TOnExceptionEvent 1298
SOAPPasInv.TOnExceptionEvent type 1298
SOAPPasInv.TSoapPascalInvoker 1284
SOAPPasInv.TSoapPascalInvoker.AfterDispatchEvent 1290
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent 1291
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent2 1291
SOAPPasInv.TSoapPascalInvoker.Converter 1289
SOAPPasInv.TSoapPascalInvoker.Create 1285
SOAPPasInv.TSoapPascalInvoker.Destroy 1286
SOAPPasInv.TSoapPascalInvoker.FConverter 1286
SOAPPasInv.TSoapPascalInvoker.FDomConverter 1286
SOAPPasInv.TSoapPascalInvoker.FHeadersIn 1287
SOAPPasInv.TSoapPascalInvoker.FHeadersOut 1287
SOAPPasInv.TSoapPascalInvoker.FSOAPHeaders 1287
SOAPPasInv.TSoapPascalInvoker.GetDomConverter 1288
SOAPPasInv.TSoapPascalInvoker.GetInvoker 1288
SOAPPasInv.TSoapPascalInvoker.Invoke 1288
SOAPPasInv.TSoapPascalInvoker.Notification 1289
SOAPPasInv.TSoapPascalInvoker.OnExceptionEvent 1291
SOAPPasInv.TSoapPascalInvoker.OnGetAttachment 1292
SOAPPasInv.TSoapPascalInvoker.OnSendAttachment 1292
soAutoCheckAccessPointViaUDDI enumeration member 1236
Sockets 1299
Sockets namespace 1299
Classes 1301
Constants 1412
Structs, Records, Enums 1403
Types 1408
Sockets.CRLF 1412
Sockets.CRLF constant 1412
Sockets.ESocketError 1301
Sockets.TBaseSocket 1302
Sockets.TBaseSocket.Active 1314
Sockets.TBaseSocket.BlockMode 1315
Sockets.TBaseSocket.BytesReceived 1315
Sockets.TBaseSocket.BytesSent 1315
Sockets.TBaseSocket.Close 1305
Sockets.TBaseSocket.Create 1303
Sockets.TBaseSocket.Destroy 1304
Sockets.TBaseSocket.DoCreateHandle 1305
Sockets.TBaseSocket.DoDestroyHandle 1305
Sockets.TBaseSocket.DoHandleError 1305
Sockets.TBaseSocket.Domain 1316
Sockets.TBaseSocket.DoReceive 1306
Sockets.TBaseSocket.DoSend 1306
Sockets.TBaseSocket.ErrorCheck 1306

Sockets.TBaseSocket.Handle 1316
Sockets.TBaseSocket.Loaded 1306
Sockets.TBaseSocket.MapDomain 1307
Sockets.TBaseSocket.MapSockType 1307
Sockets.TBaseSocket.OnCreateHandle 1317
Sockets.TBaseSocket.OnDestroyHandle 1319
Sockets.TBaseSocket.OnError 1320
Sockets.TBaseSocket.OnReceive 1320
Sockets.TBaseSocket.OnSend 1321
Sockets.TBaseSocket.Open 1307
Sockets.TBaseSocket.PeekBuf 1308
Sockets.TBaseSocket.Protocol 1316
Sockets.TBaseSocket.ReceiveBuf 1308
Sockets.TBaseSocket.Receiveln 1310
Sockets.TBaseSocket.Select 1310
Sockets.TBaseSocket.SendBuf 1310
Sockets.TBaseSocket.Sendln 1311
Sockets.TBaseSocket.SendStream 1311
Sockets.TBaseSocket.SetBytesReceived 1312
Sockets.TBaseSocket.SetBytesSent 1313
Sockets.TBaseSocket.SockType 1317
Sockets.TBaseSocket.WaitForData 1313
Sockets.TClientSocketThread 1322
Sockets.TClientSocketThread.ClientSocket 1325
Sockets.TClientSocketThread.Create 1323
Sockets.TClientSocketThread.Destroy 1323
Sockets.TClientSocketThread.Execute 1324
Sockets.TClientSocketThread.ExecuteSyncProc 1324
Sockets.TClientSocketThread.ServerSocketThread 1326
Sockets.TClientSocketThread.SyncProc 1325
Sockets.TCustomIpClient 1326
Sockets.TCustomIpClient.Close 1329
Sockets.TCustomIpClient.Connect 1329
Sockets.TCustomIpClient.Connected 1331
Sockets.TCustomIpClient.Create 1328
Sockets.TCustomIpClient.Disconnect 1329
Sockets.TCustomIpClient.DoConnect 1330
Sockets.TCustomIpClient.DoDisconnect 1330
Sockets.TCustomIpClient.GetThreadObject 1330
Sockets.TCustomIpClient.OnConnect 1332
Sockets.TCustomIpClient.OnDisconnect 1332
Sockets.TCustomIpClient.Open 1330
Sockets.TCustomTcpServer 1332
Sockets.TCustomTcpServer.Accept 1336
Sockets.TCustomTcpServer.BlockMode 1338
Sockets.TCustomTcpServer.Close 1336
Sockets.TCustomTcpServer.Create 1334
Sockets.TCustomTcpServer.Destroy 1335
Sockets.TCustomTcpServer.DoAccept 1336
Sockets.TCustomTcpServer.Listen 1337
Sockets.TCustomTcpServer.Listening 1338
Sockets.TCustomTcpServer.OnAccept 1339
Sockets.TCustomTcpServer.OnGetThread 1340
Sockets.TCustomTcpServer.OnListening 1341
Sockets.TCustomTcpServer.Open 1337
Sockets.TCustomTcpServer.ServerSocketThread 1338
Sockets.TCustomTcpServer.WaitForConnection 1337
Sockets.TGetThreadEvent 1408
Sockets.TGetThreadEvent type 1408
Sockets.TIPHeader 1403
Sockets.TIPHeader record 1403
Sockets.TIpSocket 1341
Sockets.TIpSocket.Bind 1344
Sockets.TIpSocket.Create 1343
Sockets.TIpSocket.Domain 1348
Sockets.TIpSocket.GetSocketAddr 1344
Sockets.TIpSocket.LocalDomainName 1344
Sockets.TIpSocket.LocalHost 1348
Sockets.TIpSocket.LocalHostAddr 1345
Sockets.TIpSocket.LocalHostName 1345
Sockets.TIpSocket.LocalPort 1349
Sockets.TIpSocket.LookupHostAddr 1345
Sockets.TIpSocket.LookupHostName 1346
Sockets.TIpSocket.LookupPort 1346
Sockets.TIpSocket.LookupProtocol 1347
Sockets.TIpSocket.ReceiveFrom 1347
Sockets.TIpSocket.RemoteHost 1350
Sockets.TIpSocket.RemotePort 1352

Sockets.TIpSocket.SendTo 1347
Sockets.TRawSocket 1353
Sockets.TRawSocket.Create 1355
Sockets.TRawSocket.SockType 1355
Sockets.TServerSocketBlockMode 1404
Sockets.TServerSocketBlockMode enumeration 1404
Sockets.TServerSocketThread 1356
Sockets.TServerSocketThread.AddClientSocketThread 1358
Sockets.TServerSocketThread.ClearThreadPool 1358
Sockets.TServerSocketThread.Create 1357
Sockets.TServerSocketThread.CreateThread 1359
Sockets.TServerSocketThread.Destroy 1357
Sockets.TServerSocketThread.Execute 1359
Sockets.TServerSocketThread.FetchClientSocketThread 1359
Sockets.TServerSocketThread.OnGetThread 1361
Sockets.TServerSocketThread.RemoveClientSocketThread 1359
Sockets.TServerSocketThread.ServerSocket 1360
Sockets.TServerSocketThread.ThreadCacheSize 1360
Sockets.TServerSocketThread.ThreadPool 1361
Sockets.TSocketAcceptEvent 1408
Sockets.TSocketAcceptEvent type 1408
Sockets.TSocketBlockMode 1409
Sockets.TSocketBlockMode type 1409
Sockets.TSocketDataEvent 1409
Sockets.TSocketDataEvent type 1409
Sockets.TSocketDomain 1404
Sockets.TSocketDomain enumeration 1404
Sockets.TSocketErrorEvent 1409
Sockets.TSocketErrorEvent type 1409
Sockets.TSocketHost 1410
Sockets.TSocketHost type 1410
Sockets.TSocketNotifyEvent 1410
Sockets.TSocketNotifyEvent type 1410
Sockets.TSocketPort 1410
Sockets.TSocketPort type 1410
Sockets.TSocketProtocol 1411
Sockets.TSocketProtocol type 1411
Sockets.TSocketType 1406
Sockets.TSocketType enumeration 1406
Sockets.TTcpClient 1362
Sockets.TTcpClient.Active 1365
Sockets.TTcpClient.BlockMode 1366
Sockets.TTcpClient.Connected 1366
Sockets.TTcpClient.OnConnect 1370
Sockets.TTcpClient.OnCreateHandle 1370
Sockets.TTcpClient.OnDestroyHandle 1371
Sockets.TTcpClient.OnDisconnect 1373
Sockets.TTcpClient.OnError 1373
Sockets.TTcpClient.OnReceive 1373
Sockets.TTcpClient.OnSend 1374
Sockets.TTcpClient.RemoteHost 1367
Sockets.TTcpClient.RemotePort 1368
Sockets.TTcpServer 1375
Sockets.TTcpServer.Active 1379
Sockets.TTcpServer.BlockMode 1380
Sockets.TTcpServer.LocalHost 1380
Sockets.TTcpServer.LocalPort 1381
Sockets.TTcpServer.OnAccept 1382
Sockets.TTcpServer.OnCreateHandle 1384
Sockets.TTcpServer.OnDestroyHandle 1385
Sockets.TTcpServer.OnGetThread 1387
Sockets.TTcpServer.OnListening 1387
Sockets.TUdpSocket 1387
Sockets.TUdpSocket.Active 1390
Sockets.TUdpSocket.BlockMode 1391
Sockets.TUdpSocket.Create 1389
Sockets.TUdpSocket.LocalHost 1392
Sockets.TUdpSocket.LocalPort 1392
Sockets.TUdpSocket.OnConnect 1396
Sockets.TUdpSocket.OnCreateHandle 1397
Sockets.TUdpSocket.OnDestroyHandle 1398
Sockets.TUdpSocket.OnDisconnect 1399
Sockets.TUdpSocket.OnError 1400
Sockets.TUdpSocket.OnReceive 1400
Sockets.TUdpSocket.OnSend 1401
Sockets.TUdpSocket.RemoteHost 1394
Sockets.TUdpSocket.RemotePort 1395
soDown enumeration member 3000

soHorizontal enumeration member 4679
soIgnoreInvalidCerts enumeration member 1236
soLeftRight enumeration member 4574
soMatchCase enumeration member 3000
soNone enumeration member 4679
soNoSOAPActionHeader enumeration member 1236
soNoValueForEmptySOAPAction enumeration member 1236
soShowHeader enumeration member 318
soShowShortCut enumeration member 318
soUpDown enumeration member 4574
soVertical enumeration member 4679
soWholeWord enumeration member 3000
sPointer template 3326
& 3327
~sPointer 3326
-> 3328
about sPointer template 3326
sPointer 3327
sPointer constructor 3326
sPointer Members 3326
sPointer operators 3327
SqlExpr 1413
SqlExpr namespace 1413
Classes 1416
Constants 1644
Functions 1620
Structs, Records, Enums 1624
Types 1633
Variables 1643
SqlExpr.BlobTypeMap 1644
SqlExpr.BlobTypeMap constant 1644
SqlExpr.DataTypeMap 1645
SqlExpr.DataTypeMap constant 1645
SqlExpr.DefaultCursor 1645
SqlExpr.DefaultCursor constant 1645
SqlExpr.DefaultMaxBlobSize 1646
SqlExpr.DefaultMaxBlobSize constant 1646
SqlExpr.DefaultRowsetSize 1646
SqlExpr.DefaultRowsetSize constant 1646
SqlExpr.DllGetClassObject 1643
SqlExpr.DllGetClassObject variable 1643
SqlExpr.EConnectFlag 1624
SqlExpr.EConnectFlag enumeration 1624
SqlExpr.eSQLAutoIncr 1646
SqlExpr.eSQLAutoIncr constant 1646
SqlExpr.eSQLDefault 1647
SqlExpr.eSQLDefault constant 1647
SqlExpr.eSQLFunction 1647
SqlExpr.eSQLFunction constant 1647
SqlExpr.eSQLLocal 1647
SqlExpr.eSQLLocal constant 1647
SqlExpr.eSQLNonUnique 1648
SqlExpr.eSQLNonUnique constant 1648
SqlExpr.eSQLPackage 1648
SqlExpr.eSQLPackage constant 1648
SqlExpr.eSQLPrimaryKey 1648
SqlExpr.eSQLPrimaryKey constant 1648
SqlExpr.eSQLProcedure 1649
SqlExpr.eSQLProcedure constant 1649
SqlExpr.eSQLRowId 1649
SqlExpr.eSQLRowId constant 1649
SqlExpr.eSQLRowVersion 1649
SqlExpr.eSQLRowVersion constant 1649
SqlExpr.eSQLSynonym 1650
SqlExpr.eSQLSynonym constant 1650
SqlExpr.eSQLSysProcedure 1650
SqlExpr.eSQLSysProcedure constant 1650
SqlExpr.eSQLSystemTable 1650
SqlExpr.eSQLSystemTable constant 1650
SqlExpr.eSQLTable 1651
SqlExpr.eSQLTable constant 1651
SqlExpr.eSQLTempTable 1651
SqlExpr.eSQLTempTable constant 1651
SqlExpr.eSQLUnique 1651
SqlExpr.eSQLUnique constant 1651
SqlExpr.eSQLView 1652
SqlExpr.eSQLView constant 1652
SqlExpr.FldSubTypeMap 1652

SqlExpr.FldSubTypeMap constant 1652
SqlExpr.FldTypeMap 1652
SqlExpr.FldTypeMap constant 1652
SqlExpr.FreeProcParams 1620
SqlExpr.FreeProcParams function 1620
SqlExpr.GetConnectionNames 1620
SqlExpr.GetConnectionNames function 1620
SqlExpr.GetConnectionRegistryFile 1621
SqlExpr.GetConnectionRegistryFile function 1621
SqlExpr.GetDelegateDriverNames 1621
SqlExpr.GetDelegateDriverNames function 1621
SqlExpr.GetDriverNames 1621
SqlExpr.GetDriverNames function 1621
SqlExpr.GetDriverRegistryFile 1622
SqlExpr.GetDriverRegistryFile function 1622
SqlExpr.HourGlassCursor 1653
SqlExpr.HourGlassCursor constant 1653
SqlExpr.LoadParamListItems 1622
SqlExpr.LoadParamListItems function 1622
SqlExpr.PSPParamDesc 1633
SqlExpr.PSPParamDesc type 1633
SqlExpr.pSQLTRACEDesc 1634
SqlExpr.pSQLTRACEDesc type 1634
SqlExpr.pSQLTRACEDesc25 1634
SqlExpr.pSQLTRACEDesc25 type 1634
SqlExpr.pSQLTRACEDesc30 1634
SqlExpr.pSQLTRACEDesc30 type 1634
SqlExpr.RegisterDbXpressLib 1623
SqlExpr.RegisterDbXpressLib function 1623
SqlExpr.SAnd 1653
SqlExpr.SAnd constant 1653
SqlExpr.SOrderBy 1653
SqlExpr.SOrderBy constant 1653
SqlExpr.SParam 1654
SqlExpr.SParam constant 1654
SqlExpr.SPParamDesc 1625
SqlExpr.SPParamDesc record 1625
SqlExpr.SQLResult 1635
SqlExpr.SQLResult type 1635
SqlExpr.SQLSPParamDesc 1636
SqlExpr.SQLSPParamDesc type 1636
SqlExpr.SQLTRACEDesc 1636
SqlExpr.SQLTRACEDesc type 1636
SqlExpr.SQLTRACEDesc25 1626
SqlExpr.SQLTRACEDesc25 record 1626
SqlExpr.SQLTRACEDesc30 1626
SqlExpr.SQLTRACEDesc30 record 1626
SqlExpr.SSelect 1654
SqlExpr.SSelect constant 1654
SqlExpr.SSelectSpaces 1654
SqlExpr.SSelectSpaces constant 1654
SqlExpr.SSelectStar 1655
SqlExpr.SSelectStar constant 1655
SqlExpr.SSelectStarFrom 1655
SqlExpr.SSelectStarFrom constant 1655
SqlExpr.SUB_TYPE_MEMO 1655
SqlExpr.SUB_TYPE_MEMO constant 1655
SqlExpr.SWhere 1656
SqlExpr.SWhere constant 1656
SqlExpr.TConnectChangeEvent 1636
SqlExpr.TConnectChangeEvent type 1636
SqlExpr.TConnectionData 1416
SqlExpr.TConnectionData.AddProperties 1418
SqlExpr.TConnectionData.BeginUpdate 1419
SqlExpr.TConnectionData.ConnectionName 1420
SqlExpr.TConnectionData.Create 1417, 1418
SqlExpr.TConnectionData.DelegateConnection 1420
SqlExpr.TConnectionData.Destroy 1418
SqlExpr.TConnectionData.DriverName 1421
SqlExpr.TConnectionData.EndUpdate 1419
SqlExpr.TConnectionData.GetFullParams 1419
SqlExpr.TConnectionData.IsModified 1421
SqlExpr.TConnectionData.Properties 1421
SqlExpr.TConnectionData.RefreshProperties 1419
SqlExpr.TConnectionData.ReloadProperties 1420
SqlExpr.TConnectionData.UpdateProperties 1420
SqlExpr.TConnectionState 1627
SqlExpr.TConnectionState enumeration 1627

SqlExpr.TConnectionUserType 1627
SqlExpr.TConnectionUserType enumeration 1627
SqlExpr.TCustomSQLDataSet 1421
SqlExpr.TCustomSQLDataSet.Active 1453
SqlExpr.TCustomSQLDataSet.AddFieldDesc 1428
SqlExpr.TCustomSQLDataSet.AddIndexDefs 1428
SqlExpr.TCustomSQLDataSet.AfterClose 1468
SqlExpr.TCustomSQLDataSet.AfterOpen 1469
SqlExpr.TCustomSQLDataSet.AfterRefresh 1469
SqlExpr.TCustomSQLDataSet.AfterScroll 1469
SqlExpr.TCustomSQLDataSet.BeforeClose 1469
SqlExpr.TCustomSQLDataSet.BeforeOpen 1469
SqlExpr.TCustomSQLDataSet.BeforeRefresh 1470
SqlExpr.TCustomSQLDataSet.BeforeScroll 1470
SqlExpr.TCustomSQLDataSet.BlobBuffer 1454
SqlExpr.TCustomSQLDataSet.Check 1428
SqlExpr.TCustomSQLDataSet.CheckPrepareError 1428
SqlExpr.TCustomSQLDataSet.ClearIndexDefs 1429
SqlExpr.TCustomSQLDataSet.CloseCursor 1429
SqlExpr.TCustomSQLDataSet.CloseStatement 1429
SqlExpr.TCustomSQLDataSet.CommandText 1455
SqlExpr.TCustomSQLDataSet.CommandType 1456
SqlExpr.TCustomSQLDataSet.Create 1425
SqlExpr.TCustomSQLDataSet.CreateBlobStream 1429
SqlExpr.TCustomSQLDataSet.CurrentBlobSize 1456
SqlExpr.TCustomSQLDataSet.DataLink 1457
SqlExpr.TCustomSQLDataSet.DataSource 1457
SqlExpr.TCustomSQLDataSet.DefineProperties 1430
SqlExpr.TCustomSQLDataSet.DesignerData 1458
SqlExpr.TCustomSQLDataSet.Destroy 1426
SqlExpr.TCustomSQLDataSet.ExecSQL 1430
SqlExpr.TCustomSQLDataSet.ExecuteStatement 1430
SqlExpr.TCustomSQLDataSet.FreeBuffers 1431
SqlExpr.TCustomSQLDataSet.FreeCursor 1431
SqlExpr.TCustomSQLDataSet.FreeStatement 1431
SqlExpr.TCustomSQLDataSet.GetBlobFieldData 1431
SqlExpr.TCustomSQLDataSet.GetCanModify 1432
SqlExpr.TCustomSQLDataSet.GetDataSource 1432
SqlExpr.TCustomSQLDataSet.GetDetailLinkFields 1432
SqlExpr.TCustomSQLDataSet.GetFieldData 1433
SqlExpr.TCustomSQLDataSet.GetKeyFieldNames 1434
SqlExpr.TCustomSQLDataSet.GetMetadata 1458
SqlExpr.TCustomSQLDataSet.GetObjectTypeNames 1434
SqlExpr.TCustomSQLDataSet.GetOutputParams 1434
SqlExpr.TCustomSQLDataSet.GetQuoteChar 1434
SqlExpr.TCustomSQLDataSet.GetRecord 1435
SqlExpr.TCustomSQLDataSet.GetRecordCount 1435
SqlExpr.TCustomSQLDataSet.GetSortFieldNames 1435
SqlExpr.TCustomSQLDataSet.IndexDefs 1459
SqlExpr.TCustomSQLDataSet.InitRecord 1435
SqlExpr.TCustomSQLDataSet.InternalClose 1436
SqlExpr.TCustomSQLDataSet.InternalConnection 1459
SqlExpr.TCustomSQLDataSet.InternalHandleException 1436
SqlExpr.TCustomSQLDataSet.InternalInitFieldDefs 1436
SqlExpr.TCustomSQLDataSet.InternalOpen 1436
SqlExpr.TCustomSQLDataSet.InternalRefresh 1436
SqlExpr.TCustomSQLDataSet.IsCursorOpen 1437
SqlExpr.TCustomSQLDataSet.IsSequenced 1437
SqlExpr.TCustomSQLDataSet.LastError 1460
SqlExpr.TCustomSQLDataSet.Loaded 1437
SqlExpr.TCustomSQLDataSet.Locate 1438
SqlExpr.TCustomSQLDataSet.LocateRecord 1438
SqlExpr.TCustomSQLDataSet.Lookup 1439
SqlExpr.TCustomSQLDataSet.MaxBlobSize 1460
SqlExpr.TCustomSQLDataSet.NativeCommand 1460
SqlExpr.TCustomSQLDataSet.NumericMapping 1461
SqlExpr.TCustomSQLDataSet.ObjectView 1461
SqlExpr.TCustomSQLDataSet.OnCalcFields 1470
SqlExpr.TCustomSQLDataSet.OpenCursor 1439
SqlExpr.TCustomSQLDataSet.OpenSchema 1439
SqlExpr.TCustomSQLDataSet.ParamByName 1440
SqlExpr.TCustomSQLDataSet.ParamCheck 1462
SqlExpr.TCustomSQLDataSet.ParamCount 1462
SqlExpr.TCustomSQLDataSet.Params 1463
SqlExpr.TCustomSQLDataSet.ParseDeleteSql 1470
SqlExpr.TCustomSQLDataSet.ParseInsertSql 1470
SqlExpr.TCustomSQLDataSet.ParseSelectSql 1470
SqlExpr.TCustomSQLDataSet.ParseUpdateSql 1471

SqlExpr.TCustomSQLDataSet.Prepared 1464
SqlExpr.TCustomSQLDataSet.PrepareStatement 1440
SqlExpr.TCustomSQLDataSet.ProcParams 1464
SqlExpr.TCustomSQLDataSet.PropertyChanged 1440
SqlExpr.TCustomSQLDataSet.PSEndTransaction 1440
SqlExpr.TCustomSQLDataSet.PSExecute 1441
SqlExpr.TCustomSQLDataSet.PSExecuteStatement 1441
SqlExpr.TCustomSQLDataSet.PSGetAttributes 1442
SqlExpr.TCustomSQLDataSet.PSGetCommandText 1442
SqlExpr.TCustomSQLDataSet.PSGetCommandType 1443
SqlExpr.TCustomSQLDataSet.PSGetDefaultOrder 1443
SqlExpr.TCustomSQLDataSet.PSGetIndexDefs 1443
SqlExpr.TCustomSQLDataSet.PSGetKeyFieldsW 1444
SqlExpr.TCustomSQLDataSet.PSGetParams 1444
SqlExpr.TCustomSQLDataSet.PSGetQuoteCharW 1445
SqlExpr.TCustomSQLDataSet.PSGetTableNameW 1445
SqlExpr.TCustomSQLDataSet.PSGetUpdateException 1445
SqlExpr.TCustomSQLDataSet.PSInTransaction 1445
SqlExpr.TCustomSQLDataSet.PSIsSQLBased 1446
SqlExpr.TCustomSQLDataSet.PSIsSQLSupported 1446
SqlExpr.TCustomSQLDataSet.PSReset 1447
SqlExpr.TCustomSQLDataSet.PSSetCommandText 1447
SqlExpr.TCustomSQLDataSet.PSSetParams 1448
SqlExpr.TCustomSQLDataSet.PSStartTransaction 1448
SqlExpr.TCustomSQLDataSet.PSUpdateRecord 1448
SqlExpr.TCustomSQLDataSet.RecordCount 1465
SqlExpr.TCustomSQLDataSet.RowsAffected 1465
SqlExpr.TCustomSQLDataSet.SchemaInfo 1466
SqlExpr.TCustomSQLDataSet.SchemaName 1466
SqlExpr.TCustomSQLDataSet.SetBufListSize 1449
SqlExpr.TCustomSQLDataSet.SetCommandText 1449
SqlExpr.TCustomSQLDataSet.SetFCommandText 1449
SqlExpr.TCustomSQLDataSet.SetFieldData 1450
SqlExpr.TCustomSQLDataSet.SetMaxBlobSize 1450
SqlExpr.TCustomSQLDataSet.SetParamsFromCursor 1450
SqlExpr.TCustomSQLDataSet.SetSchemaInfo 1450
SqlExpr.TCustomSQLDataSet.SetSortFieldNames 1451
SqlExpr.TCustomSQLDataSet.SortFieldNames 1466
SqlExpr.TCustomSQLDataSet.SQLConnection 1467
SqlExpr.TCustomSQLDataSet.SQLError 1452
SqlExpr.TCustomSQLDataSet.TransactionLevel 1467
SqlExpr.TCustomSQLDataSet.UpdateIndexDefs 1452
SqlExpr.TErrorMessageSize 1656
SqlExpr.TErrorMessageSize constant 1656
SqlExpr.TFieldDescList 1637
SqlExpr.TFieldDescList type 1637
SqlExpr.TFieldList 1637
SqlExpr.TFieldList type 1637
SqlExpr.TFLDDesc 1471
SqlExpr.TFLDDesc.bCalcField 1478
SqlExpr.TFLDDesc.efldrRights 1479
SqlExpr.TFLDDesc.efldvVchk 1479
SqlExpr.TFLDDesc.GetbCalcField 1473
SqlExpr.TFLDDesc.GetefldrRights 1473
SqlExpr.TFLDDesc.GetefldvVchk 1473
SqlExpr.TFLDDesc.GetiFldNum 1473
SqlExpr.TFLDDesc.GetiFldType 1474
SqlExpr.TFLDDesc.GetiLen 1474
SqlExpr.TFLDDesc.GetiNullOffset 1474
SqlExpr.TFLDDesc.GetiOffset 1474
SqlExpr.TFLDDesc.GetiSubType 1474
SqlExpr.TFLDDesc.GetiUnits1 1475
SqlExpr.TFLDDesc.GetiUnits2 1475
SqlExpr.TFLDDesc.GetszName 1475
SqlExpr.TFLDDesc.iFldNum 1479
SqlExpr.TFLDDesc.iFldType 1479
SqlExpr.TFLDDesc.iLen 1479
SqlExpr.TFLDDesc.iNullOffset 1480
SqlExpr.TFLDDesc.iOffset 1480
SqlExpr.TFLDDesc.iSubType 1480
SqlExpr.TFLDDesc.iUnits1 1480
SqlExpr.TFLDDesc.iUnits2 1481
SqlExpr.TFLDDesc.SetbCalcField 1475
SqlExpr.TFLDDesc.SetefldrRights 1476
SqlExpr.TFLDDesc.SetefldvVchk 1476
SqlExpr.TFLDDesc.SetiFldNum 1476
SqlExpr.TFLDDesc.SetiFldType 1476
SqlExpr.TFLDDesc.SetiLen 1476

SqlExpr.TFLDDesc.SetiNullOffset 1477
SqlExpr.TFLDDesc.SetiOffset 1477
SqlExpr.TFLDDesc.SetiSubType 1477
SqlExpr.TFLDDesc.SetiUnits1 1477
SqlExpr.TFLDDesc.SetiUnits2 1478
SqlExpr.TFLDDesc.SetszName 1478
SqlExpr.TFLDDesc.szName 1481
SqlExpr.TFLDDescRef 1637
SqlExpr.TFLDDescRef type 1637
SqlExpr.TGetDriverFunc 1638
SqlExpr.TGetDriverFunc type 1638
SqlExpr.TISQLCommand 1481
SqlExpr.TISQLCommand.close 1482
SqlExpr.TISQLCommand.Destroy 1482
SqlExpr.TISQLCommand.execute 1483
SqlExpr.TISQLCommand.executeImmediate 1483
SqlExpr.TISQLCommand.getErrorMessage 1483, 1484
SqlExpr.TISQLCommand.getErrorMessageLen 1484
SqlExpr.TISQLCommand.getNextCursor 1484
SqlExpr.TISQLCommand.GetOption 1484
SqlExpr.TISQLCommand.getParameter 1484
SqlExpr.TISQLCommand.getRowsAffected 1485
SqlExpr.TISQLCommand.GetStringOption 1485
SqlExpr.TISQLCommand.prepare 1485
SqlExpr.TISQLCommand.SetOption 1485
SqlExpr.TISQLCommand.setParameter 1486
SqlExpr.TISQLCommand.SetStringOption 1486
SqlExpr.TISQLConnection 1486
SqlExpr.TISQLConnection.beginTransaction 1488
SqlExpr.TISQLConnection.commit 1488
SqlExpr.TISQLConnection.connect 1488
SqlExpr.TISQLConnection.Create 1487
SqlExpr.TISQLConnection.disconnect 1489
SqlExpr.TISQLConnection.getErrorMessage 1489
SqlExpr.TISQLConnection.getErrorMessageLen 1490
SqlExpr.TISQLConnection.getFldDescClass 1490
SqlExpr.TISQLConnection.GetOption 1490
SqlExpr.TISQLConnection.getSQLCommand 1490
SqlExpr.TISQLConnection.getSQLMetaData 1490
SqlExpr.TISQLConnection.GetStringOption 1491
SqlExpr.TISQLConnection.rollback 1491
SqlExpr.TISQLConnection.SetOption 1491
SqlExpr.TISQLConnection.SetStringOption 1491
SqlExpr.TISQLCursor 1492
SqlExpr.TISQLCursor.getBcd 1494
SqlExpr.TISQLCursor.getBlob 1494
SqlExpr.TISQLCursor.getBlobSize 1494
SqlExpr.TISQLCursor.getBytes 1494
SqlExpr.TISQLCursor.getColumnCount 1495
SqlExpr.TISQLCursor.getColumnLength 1495
SqlExpr.TISQLCursor.getColumnName 1495
SqlExpr.TISQLCursor.getColumnNameLength 1496
SqlExpr.TISQLCursor.getColumnPrecision 1496
SqlExpr.TISQLCursor.getColumnScale 1496
SqlExpr.TISQLCursor.getColumnType 1496
SqlExpr.TISQLCursor.getCurObjectTypeName 1497
SqlExpr.TISQLCursor.getDate 1497
SqlExpr.TISQLCursor.getDouble 1497
SqlExpr.TISQLCursor.getErrorMessage 1497, 1498
SqlExpr.TISQLCursor.getErrorMessageLen 1498
SqlExpr.TISQLCursor.getInt64 1498
SqlExpr.TISQLCursor.getLong 1498
SqlExpr.TISQLCursor.GetOption 1499
SqlExpr.TISQLCursor.getShort 1499
SqlExpr.TISQLCursor.getString 1499
SqlExpr.TISQLCursor.GetStringOption 1499
SqlExpr.TISQLCursor.getTime 1500
SqlExpr.TISQLCursor.getTimeStamp 1500
SqlExpr.TISQLCursor.getWideString 1500
SqlExpr.TISQLCursor.isAutoIncrement 1500
SqlExpr.TISQLCursor.isBlobSizeExact 1501
SqlExpr.TISQLCursor.isNullable 1501
SqlExpr.TISQLCursor.isReadOnly 1501
SqlExpr.TISQLCursor.isSearchable 1501
SqlExpr.TISQLCursor.next 1501
SqlExpr.TISQLCursor.SetOption 1502
SqlExpr.TISQLCursor.SetStringOption 1502
SqlExpr.TISQLMetaData 1502

SqlExpr.TISQLMetaData.getColumns 1503
SqlExpr.TISQLMetaData.getErrorMessage 1504
SqlExpr.TISQLMetaData.getErrorMessageLen 1504
SqlExpr.TISQLMetaData.getIndices 1504
SqlExpr.TISQLMetaData.getObjectList 1505
SqlExpr.TISQLMetaData.GetOption 1505
SqlExpr.TISQLMetaData.getProcedureParams 1505
SqlExpr.TISQLMetaData.getProcedures 1505
SqlExpr.TISQLMetaData.GetStringOption 1506
SqlExpr.TISQLMetaData.getTables 1506
SqlExpr.TISQLMetaData.SetOption 1506
SqlExpr.TISQLMetaData.SetStringOption 1506
SqlExpr.TLocale 1638
SqlExpr.TLocale type 1638
SqlExpr.TLocaleCode 1638
SqlExpr.TLocaleCode type 1638
SqlExpr.TParseInsertSqlEvent 1639
SqlExpr.TParseInsertSqlEvent type 1639
SqlExpr.TParseSqlEvent 1639
SqlExpr.TParseSqlEvent type 1639
SqlExpr.TSchemaType 1628
SqlExpr.TSchemaType enumeration 1628
SqlExpr.TSQLBlobStream 1507
SqlExpr.TSQLBlobStream.Create 1508
SqlExpr.TSQLBlobStream.Destroy 1508
SqlExpr.TSQLBlobStream.ReadBlobData 1509
SqlExpr.TSQLConnection 1510
SqlExpr.TSQLConnection.ActiveStatements 1532
SqlExpr.TSQLConnection.AfterConnect 1550
SqlExpr.TSQLConnection.AfterDisconnect 1551
SqlExpr.TSQLConnection.AutoClone 1532
SqlExpr.TSQLConnection.BeforeConnect 1551
SqlExpr.TSQLConnection.BeforeDisconnect 1552
SqlExpr.TSQLConnection.Check 1514
SqlExpr.TSQLConnection.CheckConnection 1514
SqlExpr.TSQLConnection.CheckDisconnect 1515
SqlExpr.TSQLConnection.CloneConnection 1515
SqlExpr.TSQLConnection.CloseDataSets 1515
SqlExpr.TSQLConnection.Commit 1516
SqlExpr.TSQLConnection.Connected 1533
SqlExpr.TSQLConnection.Connection 1534
SqlExpr.TSQLConnection.ConnectionName 1534
SqlExpr.TSQLConnection.ConnectionOptions 1516
SqlExpr.TSQLConnection.ConnectionRegistryFile 1535
SqlExpr.TSQLConnection.ConnectionState 1535
SqlExpr.TSQLConnection.Create 1513
SqlExpr.TSQLConnection.DataSets 1535
SqlExpr.TSQLConnection.Destroy 1513
SqlExpr.TSQLConnection.DoConnect 1516
SqlExpr.TSQLConnection.DoDisconnect 1517
SqlExpr.TSQLConnection.Driver 1536
SqlExpr.TSQLConnection.DriverName 1536
SqlExpr.TSQLConnection.DriverRegistryFile 1537
SqlExpr.TSQLConnection.Execute 1517
SqlExpr.TSQLConnection.ExecuteDirect 1518
SqlExpr.TSQLConnection.GetConnected 1518
SqlExpr.TSQLConnection.GetDataSet 1519
SqlExpr.TSQLConnection.GetDefaultSchemaName 1519
SqlExpr.TSQLConnection.GetDriverFunc 1537
SqlExpr.TSQLConnection.GetFieldNames 1519
SqlExpr.TSQLConnection.GetIndexNames 1521
SqlExpr.TSQLConnection.GetLoginUsername 1521
SqlExpr.TSQLConnection.GetPackageNames 1522
SqlExpr.TSQLConnection.GetProcedureNames 1523
SqlExpr.TSQLConnection.GetProcedureParams 1524
SqlExpr.TSQLConnection.GetSchemaNames 1525
SqlExpr.TSQLConnection.GetTableNames 1525
SqlExpr.TSQLConnection.InTransaction 1537
SqlExpr.TSQLConnection.KeepConnection 1538
SqlExpr.TSQLConnection.LastError 1538
SqlExpr.TSQLConnection.LibraryName 1538
SqlExpr.TSQLConnection.Loaded 1526
SqlExpr.TSQLConnection.LoadParamsFromIniFile 1527
SqlExpr.TSQLConnection.LoadParamsOnConnect 1539
SqlExpr.TSQLConnection.LoadSQLDll 1527
SqlExpr.TSQLConnection.LocaleCode 1540
SqlExpr.TSQLConnection.LoginPrompt 1540
SqlExpr.TSQLConnection.MaxStmtsPerConn 1544

SqlExpr.TSQLConnection.MetaData 1545
SqlExpr.TSQLConnection.MultipleTransactionsSupported 1545
SqlExpr.TSQLConnection.Notification 1528
SqlExpr.TSQLConnection.OnLogin 1552
SqlExpr.TSQLConnection.OpenSchema 1528, 1529
SqlExpr.TSQLConnection.Params 1546
SqlExpr.TSQLConnection.ParamsLoaded 1547
SqlExpr.TSQLConnection.QuoteChar 1547
SqlExpr.TSQLConnection.Rollback 1529
SqlExpr.TSQLConnection.SetTraceCallbackEvent 1529
SqlExpr.TSQLConnection.SQLConnection 1547
SqlExpr.TSQLConnection.SQLDllHandle 1547
SqlExpr.TSQLConnection.SQLError 1530
SqlExpr.TSQLConnection.SQLHourGlass 1548
SqlExpr.TSQLConnection.StartTransaction 1530
SqlExpr.TSQLConnection.TableScope 1548
SqlExpr.TSQLConnection.TraceCallbackEvent 1549
SqlExpr.TSQLConnection.TransactionsSupported 1549
SqlExpr.TSQLConnection.VendorLib 1550
SqlExpr.TSQLConnectionLoginEvent 1639
SqlExpr.TSQLConnectionLoginEvent type 1639
SqlExpr.TSQLDataLink 1553
SqlExpr.TSQLDataLink.ActiveChanged 1554
SqlExpr.TSQLDataLink.CheckBrowseMode 1555
SqlExpr.TSQLDataLink.Create 1554
SqlExpr.TSQLDataLink.GetDetailDataSet 1555
SqlExpr.TSQLDataLink.RecordChanged 1555
SqlExpr.TSQLDataSet 1556
SqlExpr.TSQLDataSet.CommandText 1561
SqlExpr.TSQLDataSet.CommandType 1562
SqlExpr.TSQLDataSet.Create 1560
SqlExpr.TSQLDataSet.DataSource 1563
SqlExpr.TSQLDataSet.ExecSQL 1560
SqlExpr.TSQLDataSet.MaxBlobSize 1564
SqlExpr.TSQLDataSet.ParamCheck 1564
SqlExpr.TSQLDataSet.Params 1565
SqlExpr.TSQLDataSet.SortFieldNames 1566
SqlExpr.TSQLDataSet.SQLConnection 1566
SqlExpr.TSQLExceptionType 1629
SqlExpr.TSQLExceptionType enumeration 1629
SqlExpr.TSQLMonitor 1567
SqlExpr.TSQLMonitor.Active 1573
SqlExpr.TSQLMonitor.AutoSave 1573
SqlExpr.TSQLMonitor.Create 1568
SqlExpr.TSQLMonitor.Destroy 1569
SqlExpr.TSQLMonitor.FileName 1574
SqlExpr.TSQLMonitor.InvokeCallBack 1569
SqlExpr.TSQLMonitor.LoadFromFile 1570
SqlExpr.TSQLMonitor.MaxTraceCount 1574
SqlExpr.TSQLMonitor.OnLogTrace 1575
SqlExpr.TSQLMonitor.OnTrace 1575
SqlExpr.TSQLMonitor.SaveToFile 1570
SqlExpr.TSQLMonitor.SetActive 1571
SqlExpr.TSQLMonitor.SetFileName 1571
SqlExpr.TSQLMonitor.SetSQLConnection 1571
SqlExpr.TSQLMonitor.SetStreamedActive 1571
SqlExpr.TSQLMonitor.SetTraceList 1571
SqlExpr.TSQLMonitor.SQLConnection 1576
SqlExpr.TSQLMonitor.SwitchConnection 1572
SqlExpr.TSQLMonitor.Trace 1572
SqlExpr.TSQLMonitor.TraceCount 1576
SqlExpr.TSQLMonitor.TraceList 1577
SqlExpr.TSQLMonitor.UpdateTraceCallBack 1572
SqlExpr.TSQLQuery 1577
SqlExpr.TSQLQuery.Create 1581
SqlExpr.TSQLQuery.DataSource 1584
SqlExpr.TSQLQuery.Destroy 1581
SqlExpr.TSQLQuery.ExecSQL 1582
SqlExpr.TSQLQuery.GetMetadata 1584
SqlExpr.TSQLQuery.MaxBlobSize 1585
SqlExpr.TSQLQuery.ParamCheck 1585
SqlExpr.TSQLQuery.Params 1586
SqlExpr.TSQLQuery.PrepareStatement 1583
SqlExpr.TSQLQuery.RowsAffected 1586
SqlExpr.TSQLQuery.SQL 1587
SqlExpr.TSQLQuery.SQLConnection 1588
SqlExpr.TSQLQuery.Text 1588
SqlExpr.TSQLSchemaInfo 1630

SqlExpr.TSQLSchemaInfo record 1630
SqlExpr.TSqlServerMethod 1589
SqlExpr.TSqlServerMethod.Create 1592
SqlExpr.TSqlServerMethod.ExecuteMethod 1593
SqlExpr.TSqlServerMethod.Params 1593
SqlExpr.TSqlServerMethod.ServerMethodName 1594
SqlExpr.TSqlServerMethod.SQLConnection 1594
SqlExpr.TSQLStoredProc 1594
SqlExpr.TSQLStoredProc.Create 1598
SqlExpr.TSQLStoredProc.ExecProc 1599
SqlExpr.TSQLStoredProc.MaxBlobSize 1600
SqlExpr.TSQLStoredProc.NextRecordSet 1599
SqlExpr.TSQLStoredProc.PackageName 1601
SqlExpr.TSQLStoredProc.ParamCheck 1601
SqlExpr.TSQLStoredProc.Params 1602
SqlExpr.TSQLStoredProc.PrepareStatement 1600
SqlExpr.TSQLStoredProc.SQLConnection 1603
SqlExpr.TSQLStoredProc.StoredProcName 1603
SqlExpr.TSQLTable 1604
SqlExpr.TSQLTable.Active 1611
SqlExpr.TSQLTable.Create 1608
SqlExpr.TSQLTable.DeleteRecords 1609
SqlExpr.TSQLTable.Destroy 1608
SqlExpr.TSQLTable.GetIndexNames 1609
SqlExpr.TSQLTable.IndexFieldCount 1613
SqlExpr.TSQLTable.IndexFieldNames 1613
SqlExpr.TSQLTable.IndexFields 1614
SqlExpr.TSQLTable.IndexName 1614
SqlExpr.TSQLTable.MasterFields 1615
SqlExpr.TSQLTable.MasterLink 1615
SqlExpr.TSQLTable.MasterSource 1615
SqlExpr.TSQLTable.MaxBlobSize 1616
SqlExpr.TSQLTable.OpenCursor 1610
SqlExpr.TSQLTable.PrepareStatement 1610
SqlExpr.TSQLTable.SetIndexField 1610
SqlExpr.TSQLTable.SQLConnection 1617
SqlExpr.TSQLTable.TableName 1617
SqlExpr.TTableScope 1630
SqlExpr.TTableScope enumeration 1630
SqlExpr.TTableScopes 1640
SqlExpr.TTableScopes type 1640
SqlExpr.TTraceEvent 1640
SqlExpr.TTraceEvent type 1640
SqlExpr.TTraceEvent25 1640
SqlExpr.TTraceEvent25 type 1640
SqlExpr.TTraceEvent30 1641
SqlExpr.TTraceEvent30 type 1641
SqlExpr.TTraceLogEvent 1641
SqlExpr.TTraceLogEvent type 1641
SqlExpr.TTraceLogEvent25 1641
SqlExpr.TTraceLogEvent25 type 1641
SqlExpr.TTraceLogEvent30 1642
SqlExpr.TTraceLogEvent30 type 1642
SqlExpr.TTransactionDesc 1631
SqlExpr.TTransactionDesc record 1631
SqlExpr.TTransactionItem 1618
SqlExpr.TTransactionItem.FNext 1618
SqlExpr.TTransactionItem.FTransaction 1619
SqlExpr.TTransactionItem.FTransactionDesc 1619
SqlExpr.TTransIsolationLevel 1631
SqlExpr.TTransIsolationLevel enumeration 1631
SqlTimSt 1657
SqlTimSt namespace 1657
    Classes 1659
    Constants 1672
    Functions 1661
    Structs, Records, Enums 1669
    Types 1671
SqlTimSt.CheckSqlTimeStamp 1661
SqlTimSt.CheckSqlTimeStamp function 1661
SqlTimSt.DateTimeToSQLTimeStamp 1662
SqlTimSt.DateTimeToSQLTimeStamp function 1662
SqlTimSt.LocalToUTC 1662
SqlTimSt.LocalToUTC function 1662
SqlTimSt.NullSQLTimeStamp 1672
SqlTimSt.NullSQLTimeStamp constant 1672
SqlTimSt.PSQLTimeStamp 1671
SqlTimSt.PSQLTimeStamp type 1671

SqlTimSt.SQLDayOfWeek 1662
SqlTimSt.SQLDayOfWeek function 1662
SqlTimSt.SQLTimeStampToDateTime 1663
SqlTimSt.SQLTimeStampToDateTime function 1663
SqlTimSt.SQLTimeStampToStr 1663
SqlTimSt.SQLTimeStampToStr function 1663
SqlTimSt.StrToSQLTimeStamp 1664
SqlTimSt.StrToSQLTimeStamp function 1664
SqlTimSt.TryStrToSQLTimeStamp 1665
SqlTimSt.TryStrToSQLTimeStamp function 1665
SqlTimSt.TSQLTimeStamp 1669
SqlTimSt.TSQLTimeStamp record 1669
SqlTimSt.TTimeZone 1659
SqlTimSt.TTimeZone.FInfo 1660
SqlTimSt.TTimeZone.FUsesDayligtSavings 1660
SqlTimSt.TTimeZone.GetTimeZone 1660
SqlTimSt.UTCToLocal 1665
SqlTimSt.UTCToLocal function 1665
SqlTimSt.VarIsSQLTimeStamp 1666
SqlTimSt.VarIsSQLTimeStamp function 1666
SqlTimSt.VarSQLTimeStamp 1666
SqlTimSt.VarSQLTimeStamp function 1666
SqlTimSt.VarSQLTimeStampCreate 1667
SqlTimSt.VarSQLTimeStampCreate function 1667
SqlTimSt.VarToSQLTimeStamp 1668
SqlTimSt.VarToSQLTimeStamp function 1668
srBottom enumeration member 4696
srLeft enumeration member 4696
srRight enumeration member 4696
srTop enumeration member 4696
ssBoth enumeration member 2912
ssHorizontal enumeration member 2912
ssNone enumeration member 2912
ssStarted enumeration member 348
ssTerminated enumeration member 348
ssTimeOut enumeration member 348
ssVertical enumeration member 2912
stAuto enumeration member 3050
stBoot enumeration member 3050
stColumns enumeration member 1628
StdActnMenus 1673
StdActnMenus namespace 1673
    Classes 1674
    Functions 1692
StdActnMenus.RegisterStandardMenus 1692
StdActnMenus.RegisterStandardMenus function 1692
StdActnMenus.TStandardAddRemoveItem 1674
StdActnMenus.TStandardAddRemoveItem.DrawBackground 1675
StdActnMenus.TStandardAddRemoveItem.DrawGlyph 1675
StdActnMenus.TStandardButtonControl 1676
StdActnMenus.TStandardButtonControl.DrawFrame 1676
StdActnMenus.TStandardButtonControl.DrawGlyph 1677
StdActnMenus.TStandardButtonControl.DrawText 1677
StdActnMenus.TStandardCustomizePopup 1677
StdActnMenus.TStandardCustomizePopup.CanAutoSize 1678
StdActnMenus.TStandardCustomizePopup.NCPaint 1678
StdActnMenus.TStandardCustomizePopup.PositionPopup 1679
StdActnMenus.TStandardDropDownButton 1679
StdActnMenus.TStandardDropDownButton.DrawFrame 1680
StdActnMenus.TStandardDropDownButton.GetPopupClass 1680
StdActnMenus.TStandardMenuButton 1680
StdActnMenus.TStandardMenuButton.DrawBackground 1681
StdActnMenus.TStandardMenuButton.DrawDesignFocus 1681
StdActnMenus.TStandardMenuButton.DrawFrame 1682
StdActnMenus.TStandardMenuButton.DrawText 1682
StdActnMenus.TStandardMenuExpandBtn 1682
StdActnMenus.TStandardMenuExpandBtn.DrawBackground 1683
StdActnMenus.TStandardMenuExpandBtn.DrawFrame 1683
StdActnMenus.TStandardMenuItem 1683
StdActnMenus.TStandardMenuItem.CMTextchanged 1685
StdActnMenus.TStandardMenuItem.DrawBackground 1685
StdActnMenus.TStandardMenuItem.DrawDesignFocus 1685
StdActnMenus.TStandardMenuItem.DrawEdge 1686
StdActnMenus.TStandardMenuItem.DrawGlyph 1686
StdActnMenus.TStandardMenuItem.DrawGlyphFrame 1686
StdActnMenus.TStandardMenuItem.DrawLargeGlyph 1687

StdActnMenus.TStandardMenuItem.DrawSeparator 1687
StdActnMenus.TStandardMenuItem.DrawText 1687
StdActnMenus.TStandardMenuPopup 1688
StdActnMenus.TStandardMenuPopup.CanAutoSize 1689
StdActnMenus.TStandardMenuPopup.GetExpandBtnClass 1689
StdActnMenus.TStandardMenuPopup.NCPaint 1689
StdActnMenus.TStandardMenuPopup.PositionPopup 1690
StdActnMenus.TStandardToolScrollBtn 1690
StdActnMenus.TStandardToolScrollBtn.DrawFrame 1691
StdActns 1693
StdActns namespace 1693
Classes 1695
Structs, Records, Enums 1875
Types 1877
StdActns.TBrowseForFolder 1696
StdActns.TBrowseForFolder.BeforeExecute 1704
StdActns.TBrowseForFolder.BrowseOptions 1699
StdActns.TBrowseForFolder.Caption 1699
StdActns.TBrowseForFolder.DialogCaption 1700
StdActns.TBrowseForFolder.DoCallback 1697
StdActns.TBrowseForFolder.DoValidate 1697
StdActns.TBrowseForFolder.Enabled 1700
StdActns.TBrowseForFolder.ExecuteTarget 1698
StdActns.TBrowseForFolder.Folder 1700
StdActns.TBrowseForFolder.GetDialogClass 1698
StdActns.TBrowseForFolder.HelpContext 1700
StdActns.TBrowseForFolder.HelpKeyword 1701
StdActns.TBrowseForFolder.HelpType 1701
StdActns.TBrowseForFolder.Hint 1702
StdActns.TBrowseForFolder.ImageIndex 1702
StdActns.TBrowseForFolder.OnAccept 1704
StdActns.TBrowseForFolder.OnCallback 1705
StdActns.TBrowseForFolder.OnCancel 1705
StdActns.TBrowseForFolder.OnHint 1705
StdActns.TBrowseForFolder.OnValidate 1705
StdActns.TBrowseForFolder.RootDir 1702
StdActns.TBrowseForFolder.SecondaryShortCuts 1703
StdActns.TBrowseForFolder.ShortCut 1703
StdActns.TBrowseForFolder.Visible 1703
StdActns.TBrowseForFolderCallbackEvent 1877
StdActns.TBrowseForFolderCallbackEvent type 1877
StdActns.TBrowseForFolderOption 1875
StdActns.TBrowseForFolderOption enumeration 1875
StdActns.TBrowseOptions 1877
StdActns.TBrowseOptions type 1877
StdActns.TColorSelect 1706
StdActns.TColorSelect.BeforeExecute 1712
StdActns.TColorSelect.Caption 1708
StdActns.TColorSelect.Dialog 1708
StdActns.TColorSelect.Enabled 1709
StdActns.TColorSelect.GetDialogClass 1707
StdActns.TColorSelect.HelpContext 1709
StdActns.TColorSelect.HelpKeyword 1709
StdActns.TColorSelect.HelpType 1710
StdActns.TColorSelect.Hint 1710
StdActns.TColorSelect.ImageIndex 1711
StdActns.TColorSelect.OnAccept 1713
StdActns.TColorSelect.OnCancel 1713
StdActns.TColorSelect.OnHint 1713
StdActns.TColorSelect.OnUpdate 1714
StdActns.TColorSelect.SecondaryShortCuts 1711
StdActns.TColorSelect.ShortCut 1712
StdActns.TColorSelect.Visible 1712
StdActns.TCommonDialogAction 1715
StdActns.TCommonDialogAction.BeforeExecute 1721
StdActns.TCommonDialogAction.Create 1716
StdActns.TCommonDialogAction.Destroy 1717
StdActns.TCommonDialogAction.DoAccept 1718
StdActns.TCommonDialogAction.DoCancel 1718
StdActns.TCommonDialogAction.ExecuteResult 1720
StdActns.TCommonDialogAction.ExecuteTarget 1718
StdActns.TCommonDialogAction.FDialog 1717
StdActns.TCommonDialogAction.GetDialogClass 1719
StdActns.TCommonDialogAction.HandlesTarget 1719
StdActns.TCommonDialogAction.Notification 1720
StdActns.TCommonDialogAction.OnAccept 1721
StdActns.TCommonDialogAction.OnCancel 1722
StdActns.TCommonDialogAction.SetupDialog 1720

StdActns.TCommonDialogClass 1878
StdActns.TCommonDialogClass type 1878
StdActns.TEditAction 1722
StdActns.TEditAction.Control 1726
StdActns.TEditAction.Destroy 1723
StdActns.TEditAction.GetControl 1724
StdActns.TEditAction.HandlesTarget 1724
StdActns.TEditAction.Notification 1725
StdActns.TEditAction.UpdateTarget 1725
StdActns.TEditCopy 1726
StdActns.TEditCopy.ExecuteTarget 1727
StdActns.TEditCut 1728
StdActns.TEditCut.ExecuteTarget 1729
StdActns.TEditDelete 1730
StdActns.TEditDelete.ExecuteTarget 1731
StdActns.TEditDelete.UpdateTarget 1731
StdActns.TEditPaste 1732
StdActns.TEditPaste.ExecuteTarget 1733
StdActns.TEditPaste.UpdateTarget 1734
StdActns.TEditSelectAll 1734
StdActns.TEditSelectAll.ExecuteTarget 1736
StdActns.TEditSelectAll.UpdateTarget 1736
StdActns.TEditUndo 1736
StdActns.TEditUndo.ExecuteTarget 1738
StdActns.TEditUndo.UpdateTarget 1738
StdActns.TFileAction 1739
StdActns.TFileAction.FileName 1741
StdActns.TFileAction.GetDialog 1740
StdActns.TFileExit 1741
StdActns.TFileExit.Caption 1743
StdActns.TFileExit.Enabled 1744
StdActns.TFileExit.ExecuteTarget 1742
StdActns.TFileExit.HandlesTarget 1743
StdActns.TFileExit.HelpContext 1744
StdActns.TFileExit.HelpKeyword 1745
StdActns.TFileExit.HelpType 1745
StdActns.TFileExit.Hint 1746
StdActns.TFileExit.ImageIndex 1746
StdActns.TFileExit.OnHint 1747
StdActns.TFileExit.OnUpdate 1748
StdActns.TFileExit.SecondaryShortCuts 1746
StdActns.TFileExit.ShortCut 1747
StdActns.TFileExit.Visible 1747
StdActns.TFileOpen 1749
StdActns.TFileOpen.BeforeExecute 1758
StdActns.TFileOpen.Caption 1753
StdActns.TFileOpen.Create 1751
StdActns.TFileOpen.Dialog 1753
StdActns.TFileOpen.Enabled 1753
StdActns.TFileOpen.ExecuteTarget 1751
StdActns.TFileOpen.GetDialogClass 1752
StdActns.TFileOpen.HelpContext 1754
StdActns.TFileOpen.HelpKeyword 1754
StdActns.TFileOpen.HelpType 1755
StdActns.TFileOpen.Hint 1755
StdActns.TFileOpen.ImageIndex 1755
StdActns.TFileOpen.OnAccept 1758
StdActns.TFileOpen.OnCancel 1758
StdActns.TFileOpen.OnHint 1758
StdActns.TFileOpen.OnUpdate 1759
StdActns.TFileOpen.SecondaryShortCuts 1756
StdActns.TFileOpen.ShortCut 1756
StdActns.TFileOpen.UseDefaultApp 1757
StdActns.TFileOpen.Visible 1757
StdActns.TFileOpenWith 1760
StdActns.TFileOpenWith.AfterOpen 1765
StdActns.TFileOpenWith.BeforeExecute 1765
StdActns.TFileOpenWith.ExecuteTarget 1762
StdActns.TFileOpenWith.FileName 1763
StdActns.TFileOpenWith.HelpContext 1763
StdActns.TFileOpenWith.HelpKeyword 1764
StdActns.TFileOpenWith.HelpType 1764
StdActns.TFileOpenWith.OnUpdate 1766
StdActns.TFilePageSetup 1767
StdActns.TFilePageSetup.BeforeExecute 1774
StdActns.TFilePageSetup.Caption 1769
StdActns.TFilePageSetup.Dialog 1770
StdActns.TFilePageSetup.Enabled 1770

StdActns.TFilePageSetup.GetDialogClass 1769
StdActns.TFilePageSetup.HelpContext 1770
StdActns.TFilePageSetup.HelpKeyword 1771
StdActns.TFilePageSetup.HelpType 1771
StdActns.TFilePageSetup.Hint 1772
StdActns.TFilePageSetup.ImageIndex 1772
StdActns.TFilePageSetup.OnAccept 1774
StdActns.TFilePageSetup.OnCancel 1774
StdActns.TFilePageSetup.OnHint 1775
StdActns.TFilePageSetup.OnUpdate 1775
StdActns.TFilePageSetup.SecondaryShortCuts 1772
StdActns.TFilePageSetup.ShortCut 1773
StdActns.TFilePageSetup.Visible 1773
StdActns.TFilePrintSetup 1776
StdActns.TFilePrintSetup.BeforeExecute 1783
StdActns.TFilePrintSetup.Caption 1778
StdActns.TFilePrintSetup.Dialog 1779
StdActns.TFilePrintSetup.Enabled 1779
StdActns.TFilePrintSetup.GetDialogClass 1778
StdActns.TFilePrintSetup.HelpContext 1779
StdActns.TFilePrintSetup.HelpKeyword 1780
StdActns.TFilePrintSetup.HelpType 1780
StdActns.TFilePrintSetup.Hint 1781
StdActns.TFilePrintSetup.ImageIndex 1781
StdActns.TFilePrintSetup.OnAccept 1783
StdActns.TFilePrintSetup.OnCancel 1783
StdActns.TFilePrintSetup.OnHint 1784
StdActns.TFilePrintSetup.OnUpdate 1784
StdActns.TFilePrintSetup.SecondaryShortCuts 1781
StdActns.TFilePrintSetup.ShortCut 1782
StdActns.TFilePrintSetup.Visible 1782
StdActns.TFileSaveAs 1785
StdActns.TFileSaveAs.BeforeExecute 1791
StdActns.TFileSaveAs.Caption 1788
StdActns.TFileSaveAs.Create 1787
StdActns.TFileSaveAs.Dialog 1788
StdActns.TFileSaveAs.Enabled 1789
StdActns.TFileSaveAs.GetDialogClass 1787
StdActns.TFileSaveAs.HelpContext 1789
StdActns.TFileSaveAs.Hint 1789
StdActns.TFileSaveAs.ImageIndex 1790
StdActns.TFileSaveAs.OnAccept 1792
StdActns.TFileSaveAs.OnCancel 1792
StdActns.TFileSaveAs.OnHint 1792
StdActns.TFileSaveAs.OnUpdate 1793
StdActns.TFileSaveAs.SecondaryShortCuts 1790
StdActns.TFileSaveAs.ShortCut 1791
StdActns.TFileSaveAs.Visible 1791
StdActns.TFontEdit 1794
StdActns.TFontEdit.BeforeExecute 1801
StdActns.TFontEdit.Caption 1796
StdActns.TFontEdit.Dialog 1796
StdActns.TFontEdit.Enabled 1797
StdActns.TFontEdit.GetDialogClass 1795
StdActns.TFontEdit.HelpContext 1797
StdActns.TFontEdit.HelpKeyword 1798
StdActns.TFontEdit.HelpType 1798
StdActns.TFontEdit.Hint 1799
StdActns.TFontEdit.ImageIndex 1799
StdActns.TFontEdit.OnAccept 1801
StdActns.TFontEdit.OnCancel 1801
StdActns.TFontEdit.OnHint 1801
StdActns.TFontEdit.OnUpdate 1802
StdActns.TFontEdit.SecondaryShortCuts 1799
StdActns.TFontEdit.ShortCut 1800
StdActns.TFontEdit.Visible 1800
StdActns.THelpAction 1803
StdActns.THelpAction.Create 1804
StdActns.THelpAction.HandlesTarget 1804
StdActns.THelpAction.UpdateTarget 1805
StdActns.THelpContents 1805
StdActns.THelpContents.ExecuteTarget 1806
StdActns.THelpContextAction 1806
StdActns.THelpContextAction.ExecuteTarget 1807
StdActns.THelpContextAction.UpdateTarget 1808
StdActns.THelpOnHelp 1808
StdActns.THelpOnHelp.ExecuteTarget 1809
StdActns.THelpTopicSearch 1810

StdActns.THelpTopicSearch.ExecuteTarget 1811
StdActns.THintAction 1811
StdActns.THintAction.Create 1812
StdActns.THintAction.Hint 1813
StdActns.TPrintDlg 1814
StdActns.TPrintDlg.BeforeExecute 1821
StdActns.TPrintDlg.Caption 1816
StdActns.TPrintDlg.Dialog 1816
StdActns.TPrintDlg.Enabled 1817
StdActns.TPrintDlg.GetDialogClass 1815
StdActns.TPrintDlg.HelpContext 1817
StdActns.TPrintDlg.HelpKeyword 1818
StdActns.TPrintDlg.HelpType 1818
StdActns.TPrintDlg.Hint 1819
StdActns.TPrintDlg.ImageIndex 1819
StdActns.TPrintDlg.OnAccept 1821
StdActns.TPrintDlg.OnCancel 1821
StdActns.TPrintDlg.OnHint 1821
StdActns.TPrintDlg.OnUpdate 1822
StdActns.TPrintDlg.SecondaryShortCuts 1819
StdActns.TPrintDlg.ShortCut 1820
StdActns.TPrintDlg.Visible 1820
StdActns.TSearchAction 1823
StdActns.TSearchAction.Create 1824
StdActns.TSearchAction.Destroy 1825
StdActns.TSearchAction.ExecuteTarget 1826
StdActns.TSearchAction.FControl 1825
StdActns.TSearchAction.FFindFirst 1826
StdActns.TSearchAction.HandlesTarget 1827
StdActns.TSearchAction.Notification 1827
StdActns.TSearchAction.Search 1827
StdActns.TSearchAction.UpdateTarget 1828
StdActns.TSearchFind 1828
StdActns.TSearchFind.BeforeExecute 1835
StdActns.TSearchFind.Caption 1831
StdActns.TSearchFind.Dialog 1831
StdActns.TSearchFind.Enabled 1832
StdActns.TSearchFind.GetDialogClass 1830
StdActns.TSearchFind.HelpContext 1832
StdActns.TSearchFind.HelpKeyword 1832
StdActns.TSearchFind.HelpType 1833
StdActns.TSearchFind.Hint 1833
StdActns.TSearchFind.ImageIndex 1834
StdActns.TSearchFind.OnAccept 1836
StdActns.TSearchFind.OnCancel 1836
StdActns.TSearchFind.OnHint 1836
StdActns.TSearchFind.OnUpdate 1837
StdActns.TSearchFind.SecondaryShortCuts 1834
StdActns.TSearchFind.ShortCut 1835
StdActns.TSearchFind.Visible 1835
StdActns.TSearchFindFirst 1838
StdActns.TSearchFindFirst.Create 1840
StdActns.TSearchFindNext 1840
StdActns.TSearchFindNext.Caption 1844
StdActns.TSearchFindNext.Create 1842
StdActns.TSearchFindNext.Enabled 1844
StdActns.TSearchFindNext.ExecuteTarget 1842
StdActns.TSearchFindNext.HandlesTarget 1843
StdActns.TSearchFindNext.HelpContext 1844
StdActns.TSearchFindNext.HelpKeyword 1845
StdActns.TSearchFindNext.HelpType 1845
StdActns.TSearchFindNext.Hint 1846
StdActns.TSearchFindNext.ImageIndex 1846
StdActns.TSearchFindNext.OnHint 1848
StdActns.TSearchFindNext.OnUpdate 1849
StdActns.TSearchFindNext.SearchFind 1847
StdActns.TSearchFindNext.SecondaryShortCuts 1847
StdActns.TSearchFindNext.ShortCut 1847
StdActns.TSearchFindNext.UpdateTarget 1843
StdActns.TSearchFindNext.Visible 1848
StdActns.TSearchReplace 1850
StdActns.TSearchReplace.BeforeExecute 1858
StdActns.TSearchReplace.Caption 1853
StdActns.TSearchReplace.Dialog 1853
StdActns.TSearchReplace.Enabled 1854
StdActns.TSearchReplace.ExecuteTarget 1852
StdActns.TSearchReplace.GetDialogClass 1852
StdActns.TSearchReplace.HelpContext 1854

StdActns.TSearchReplace.HelpKeyword 1855
StdActns.TSearchReplace.HelpType 1855
StdActns.TSearchReplace.Hint 1856
StdActns.TSearchReplace.ImageIndex 1856
StdActns.TSearchReplace.OnAccept 1858
StdActns.TSearchReplace.OnCancel 1858
StdActns.TSearchReplace.OnHint 1858
StdActns.TSearchReplace.OnUpdate 1859
StdActns.TSearchReplace.SecondaryShortCuts 1856
StdActns.TSearchReplace.ShortCut 1857
StdActns.TSearchReplace.Visible 1857
StdActns.TWindowAction 1860
StdActns.TWindowAction.Create 1861
StdActns.TWindowAction.Form 1863
StdActns.TWindowAction.GetForm 1862
StdActns.TWindowAction.HandlesTarget 1862
StdActns.TWindowAction.Notification 1862
StdActns.TWindowAction.UpdateTarget 1863
StdActns.TWindowArrange 1864
StdActns.TWindowArrange.ExecuteTarget 1865
StdActns.TWindowCascade 1865
StdActns.TWindowCascade.ExecuteTarget 1866
StdActns.TWindowClose 1867
StdActns.TWindowClose.ExecuteTarget 1868
StdActns.TWindowClose.UpdateTarget 1869
StdActns.TWindowMinimizeAll 1869
StdActns.TWindowMinimizeAll.ExecuteTarget 1870
StdActns.TWindowTileHorizontal 1871
StdActns.TWindowTileHorizontal.ExecuteTarget 1872
StdActns.TWindowTileVertical 1873
StdActns.TWindowTileVertical.ExecuteTarget 1874
StdConvs 1879
StdConvs namespace 1879
Constants 1936
Functions 1884
Variables 1886
StdConvs.auAcres 1888
StdConvs.auAcres variable 1888
StdConvs.auAres 1889
StdConvs.auAres variable 1889
StdConvs.auCentares 1889
StdConvs.auCentares variable 1889
StdConvs.auHectares 1889
StdConvs.auHectares variable 1889
StdConvs.auSquareCentimeters 1890
StdConvs.auSquareCentimeters variable 1890
StdConvs.auSquareDecameters 1890
StdConvs.auSquareDecameters variable 1890
StdConvs.auSquareDecimeters 1890
StdConvs.auSquareDecimeters variable 1890
StdConvs.auSquareFeet 1891
StdConvs.auSquareFeet variable 1891
StdConvs.auSquareHectometers 1891
StdConvs.auSquareHectometers variable 1891
StdConvs.auSquareInches 1891
StdConvs.auSquareInches variable 1891
StdConvs.auSquareKilometers 1892
StdConvs.auSquareKilometers variable 1892
StdConvs.auSquareMeters 1892
StdConvs.auSquareMeters variable 1892
StdConvs.auSquareMiles 1892
StdConvs.auSquareMiles variable 1892
StdConvs.auSquareMillimeters 1893
StdConvs.auSquareMillimeters variable 1893
StdConvs.auSquareRods 1893
StdConvs.auSquareRods variable 1893
StdConvs.auSquareYards 1893
StdConvs.auSquareYards variable 1893
StdConvs.cbArea 1894
StdConvs.cbArea variable 1894
StdConvs.cbDistance 1894
StdConvs.cbDistance variable 1894
StdConvs.cbMass 1895
StdConvs.cbMass variable 1895
StdConvs.cbTemperature 1895
StdConvs.cbTemperature variable 1895
StdConvs.cbTime 1895
StdConvs.cbTime variable 1895

StdConvs.cbVolume 1896
StdConvs.cbVolume variable 1896
StdConvs.CelsiusToFahrenheit 1884
StdConvs.CelsiusToFahrenheit function 1884
StdConvs.CubicMetersPerAcreFoot 1937
StdConvs.CubicMetersPerAcreFoot constant 1937
StdConvs.CubicMetersPerAcreInch 1937
StdConvs.CubicMetersPerAcreInch constant 1937
StdConvs.CubicMetersPerCord 1938
StdConvs.CubicMetersPerCord constant 1938
StdConvs.CubicMetersPerCordFoot 1938
StdConvs.CubicMetersPerCordFoot constant 1938
StdConvs.CubicMetersPerCubicFoot 1938
StdConvs.CubicMetersPerCubicFoot constant 1938
StdConvs.CubicMetersPerCubicInch 1939
StdConvs.CubicMetersPerCubicInch constant 1939
StdConvs.CubicMetersPerCubicMile 1939
StdConvs.CubicMetersPerCubicMile constant 1939
StdConvs.CubicMetersPerCubicYard 1939
StdConvs.CubicMetersPerCubicYard constant 1939
StdConvs.CubicMetersPerUKBucket 1940
StdConvs.CubicMetersPerUKBucket constant 1940
StdConvs.CubicMetersPerUKBushel 1940
StdConvs.CubicMetersPerUKBushel constant 1940
StdConvs.CubicMetersPerUKGallon 1940
StdConvs.CubicMetersPerUKGallon constant 1940
StdConvs.CubicMetersPerUKGill 1941
StdConvs.CubicMetersPerUKGill constant 1941
StdConvs.CubicMetersPerUKOunce 1941
StdConvs.CubicMetersPerUKOunce constant 1941
StdConvs.CubicMetersPerUKPeck 1941
StdConvs.CubicMetersPerUKPeck constant 1941
StdConvs.CubicMetersPerUKPint 1942
StdConvs.CubicMetersPerUKPint constant 1942
StdConvs.CubicMetersPerUKPottle 1942
StdConvs.CubicMetersPerUKPottle constant 1942
StdConvs.CubicMetersPerUKQuart 1942
StdConvs.CubicMetersPerUKQuart constant 1942
StdConvs.CubicMetersPerUSDryBucket 1943
StdConvs.CubicMetersPerUSDryBucket constant 1943
StdConvs.CubicMetersPerUSDryBushel 1943
StdConvs.CubicMetersPerUSDryBushel constant 1943
StdConvs.CubicMetersPerUSDryGallon 1943
StdConvs.CubicMetersPerUSDryGallon constant 1943
StdConvs.CubicMetersPerUSDryPeck 1944
StdConvs.CubicMetersPerUSDryPeck constant 1944
StdConvs.CubicMetersPerUSDryPint 1944
StdConvs.CubicMetersPerUSDryPint constant 1944
StdConvs.CubicMetersPerUSDryQuart 1944
StdConvs.CubicMetersPerUSDryQuart constant 1944
StdConvs.CubicMetersPerUSFluidCup 1945
StdConvs.CubicMetersPerUSFluidCup constant 1945
StdConvs.CubicMetersPerUSFluidGallon 1945
StdConvs.CubicMetersPerUSFluidGallon constant 1945
StdConvs.CubicMetersPerUSFluidGill 1945
StdConvs.CubicMetersPerUSFluidGill constant 1945
StdConvs.CubicMetersPerUSFluidOunce 1946
StdConvs.CubicMetersPerUSFluidOunce constant 1946
StdConvs.CubicMetersPerUSFluidPint 1946
StdConvs.CubicMetersPerUSFluidPint constant 1946
StdConvs.CubicMetersPerUSFluidQuart 1946
StdConvs.CubicMetersPerUSFluidQuart constant 1946
StdConvs.CubicMetersPerUSFluidTablespoon 1947
StdConvs.CubicMetersPerUSFluidTablespoon constant 1947
StdConvs.CubicMetersPerUSFluidTeaspoon 1947
StdConvs.CubicMetersPerUSFluidTeaspoon constant 1947
StdConvs.duAngstroms 1897
StdConvs.duAngstroms variable 1897
StdConvs.duAstronomicalUnits 1897
StdConvs.duAstronomicalUnits variable 1897
StdConvs.duCentimeters 1898
StdConvs.duCentimeters variable 1898
StdConvs.duChains 1898
StdConvs.duChains variable 1898
StdConvs.duCubits 1898
StdConvs.duCubits variable 1898
StdConvs.duDecameters 1899
StdConvs.duDecameters variable 1899

StdConvs.duDecimeters 1899
StdConvs.duDecimeters variable 1899
StdConvs.duFathoms 1899
StdConvs.duFathoms variable 1899
StdConvs.duFeet 1900
StdConvs.duFeet variable 1900
StdConvs.duFurlongs 1900
StdConvs.duFurlongs variable 1900
StdConvs.duGigameters 1900
StdConvs.duGigameters variable 1900
StdConvs.duHands 1901
StdConvs.duHands variable 1901
StdConvs.duHectometers 1901
StdConvs.duHectometers variable 1901
StdConvs.duInches 1901
StdConvs.duInches variable 1901
StdConvs.duKilometers 1902
StdConvs.duKilometers variable 1902
StdConvs.duLightYears 1902
StdConvs.duLightYears variable 1902
StdConvs.duLinks 1902
StdConvs.duLinks variable 1902
StdConvs.duMegameters 1903
StdConvs.duMegameters variable 1903
StdConvs.duMeters 1903
StdConvs.duMeters variable 1903
StdConvs.duMicromicrons 1903
StdConvs.duMicromicrons variable 1903
StdConvs.duMicrons 1904
StdConvs.duMicrons variable 1904
StdConvs.duMiles 1904
StdConvs.duMiles variable 1904
StdConvs.duMillimeters 1904
StdConvs.duMillimeters variable 1904
StdConvs.duMillimicrons 1905
StdConvs.duMillimicrons variable 1905
StdConvs.duNauticalMiles 1905
StdConvs.duNauticalMiles variable 1905
StdConvs.duPaces 1905
StdConvs.duPaces variable 1905
StdConvs.duParsecs 1906
StdConvs.duParsecs variable 1906
StdConvs.duPicas 1906
StdConvs.duPicas variable 1906
StdConvs.duPoints 1906
StdConvs.duPoints variable 1906
StdConvs.duRods 1907
StdConvs.duRods variable 1907
StdConvs.duYards 1907
StdConvs.duYards variable 1907
StdConvs.FahrenheitToCelsius 1884
StdConvs.FahrenheitToCelsius function 1884
StdConvs.GramsPerDrams 1947
StdConvs.GramsPerDrams constant 1947
StdConvs.GramsPerGrains 1948
StdConvs.GramsPerGrains constant 1948
StdConvs.GramsPerLongTons 1948
StdConvs.GramsPerLongTons constant 1948
StdConvs.GramsPerOunces 1948
StdConvs.GramsPerOunces constant 1948
StdConvs.GramsPerPound 1949
StdConvs.GramsPerPound constant 1949
StdConvs.GramsPerStones 1949
StdConvs.GramsPerStones constant 1949
StdConvs.GramsPerTons 1949
StdConvs.GramsPerTons constant 1949
StdConvs.InitStdConvs 1885
StdConvs.InitStdConvs function 1885
StdConvs.MetersPerAstronomicalUnit 1950
StdConvs.MetersPerAstronomicalUnit constant 1950
StdConvs.MetersPerChain 1950
StdConvs.MetersPerChain constant 1950
StdConvs.MetersPerCubit 1950
StdConvs.MetersPerCubit constant 1950
StdConvs.MetersPerFathom 1951
StdConvs.MetersPerFathom constant 1951
StdConvs.MetersPerFoot 1951
StdConvs.MetersPerFoot constant 1951

StdConvs.MetersPerFurlong 1951
StdConvs.MetersPerFurlong constant 1951
StdConvs.MetersPerHand 1952
StdConvs.MetersPerHand constant 1952
StdConvs.MetersPerInch 1952
StdConvs.MetersPerInch constant 1952
StdConvs.MetersPerLightSecond 1952
StdConvs.MetersPerLightSecond constant 1952
StdConvs.MetersPerLightYear 1953
StdConvs.MetersPerLightYear constant 1953
StdConvs.MetersPerLink 1953
StdConvs.MetersPerLink constant 1953
StdConvs.MetersPerMile 1953
StdConvs.MetersPerMile constant 1953
StdConvs.MetersPerNauticalMiles 1954
StdConvs.MetersPerNauticalMiles constant 1954
StdConvs.MetersPerPace 1954
StdConvs.MetersPerPace constant 1954
StdConvs.MetersPerParsec 1954
StdConvs.MetersPerParsec constant 1954
StdConvs.MetersPerPica 1955
StdConvs.MetersPerPica constant 1955
StdConvs.MetersPerPoint 1955
StdConvs.MetersPerPoint constant 1955
StdConvs.MetersPerRod 1955
StdConvs.MetersPerRod constant 1955
StdConvs.MetersPerYard 1956
StdConvs.MetersPerYard constant 1956
StdConvs.muCentigrams 1907
StdConvs.muCentigrams variable 1907
StdConvs.muDecagrams 1908
StdConvs.muDecagrams variable 1908
StdConvs.muDecigrams 1908
StdConvs.muDecigrams variable 1908
StdConvs.muDrams 1908
StdConvs.muDrams variable 1908
StdConvs.muGrains 1909
StdConvs.muGrains variable 1909
StdConvs.muGrams 1909
StdConvs.muGrams variable 1909
StdConvs.muHectograms 1909
StdConvs.muHectograms variable 1909
StdConvs.muKilograms 1910
StdConvs.muKilograms variable 1910
StdConvs.muLongTons 1910
StdConvs.muLongTons variable 1910
StdConvs.muMetricTons 1910
StdConvs.muMetricTons variable 1910
StdConvs.muMicrograms 1911
StdConvs.muMicrograms variable 1911
StdConvs.muMilligrams 1911
StdConvs.muMilligrams variable 1911
StdConvs.muNanograms 1911
StdConvs.muNanograms variable 1911
StdConvs.muOunces 1912
StdConvs.muOunces variable 1912
StdConvs.muPounds 1912
StdConvs.muPounds variable 1912
StdConvs.muStones 1912
StdConvs.muStones variable 1912
StdConvs.muTons 1913
StdConvs.muTons variable 1913
StdConvs.SquareMetersPerAcre 1956
StdConvs.SquareMetersPerAcre constant 1956
StdConvs.SquareMetersPerSquareFoot 1956
StdConvs.SquareMetersPerSquareFoot constant 1956
StdConvs.SquareMetersPerSquareInch 1957
StdConvs.SquareMetersPerSquareInch constant 1957
StdConvs.SquareMetersPerSquareMile 1957
StdConvs.SquareMetersPerSquareMile constant 1957
StdConvs.SquareMetersPerSquareRod 1957
StdConvs.SquareMetersPerSquareRod constant 1957
StdConvs.SquareMetersPerSquareYard 1958
StdConvs.SquareMetersPerSquareYard constant 1958
StdConvs.tuCelsius 1913
StdConvs.tuCelsius variable 1913
StdConvs.tuCenturies 1913
StdConvs.tuCenturies variable 1913

StdConvs.tuDateTime 1914
StdConvs.tuDateTime variable 1914
StdConvs.tuDays 1914
StdConvs.tuDays variable 1914
StdConvs.tuDecades 1914
StdConvs.tuDecades variable 1914
StdConvs.tuFahrenheit 1915
StdConvs.tuFahrenheit variable 1915
StdConvs.tuFortnights 1915
StdConvs.tuFortnights variable 1915
StdConvs.tuHours 1915
StdConvs.tuHours variable 1915
StdConvs.tuJulianDate 1916
StdConvs.tuJulianDate variable 1916
StdConvs.tuKelvin 1916
StdConvs.tuKelvin variable 1916
StdConvs.tuMillennia 1916
StdConvs.tuMillennia variable 1916
StdConvs.tuMilliSeconds 1917
StdConvs.tuMilliSeconds variable 1917
StdConvs.tuMinutes 1917
StdConvs.tuMinutes variable 1917
StdConvs.tuModifiedJulianDate 1917
StdConvs.tuModifiedJulianDate variable 1917
StdConvs.tuMonths 1918
StdConvs.tuMonths variable 1918
StdConvs.tuRankine 1918
StdConvs.tuRankine variable 1918
StdConvs.tuReaumur 1918
StdConvs.tuReaumur variable 1918
StdConvs.tuSeconds 1919
StdConvs.tuSeconds variable 1919
StdConvs.tuWeeks 1919
StdConvs.tuWeeks variable 1919
StdConvs.tuYears 1919
StdConvs.tuYears variable 1919
StdConvs.vuAcreFeet 1920
StdConvs.vuAcreFeet variable 1920
StdConvs.vuAcreInches 1920
StdConvs.vuAcreInches variable 1920
StdConvs.vuCentiLiters 1920
StdConvs.vuCentiLiters variable 1920
StdConvs.vuCordFeet 1921
StdConvs.vuCordFeet variable 1921
StdConvs.vuCords 1921
StdConvs.vuCords variable 1921
StdConvs.vuCubicCentimeters 1921
StdConvs.vuCubicCentimeters variable 1921
StdConvs.vuCubicDecameters 1922
StdConvs.vuCubicDecameters variable 1922
StdConvs.vuCubicDecimeters 1922
StdConvs.vuCubicDecimeters variable 1922
StdConvs.vuCubicFeet 1922
StdConvs.vuCubicFeet variable 1922
StdConvs.vuCubicHectometers 1923
StdConvs.vuCubicHectometers variable 1923
StdConvs.vuCubicInches 1923
StdConvs.vuCubicInches variable 1923
StdConvs.vuCubicKilometers 1923
StdConvs.vuCubicKilometers variable 1923
StdConvs.vuCubicMeters 1924
StdConvs.vuCubicMeters variable 1924
StdConvs.vuCubicMiles 1924
StdConvs.vuCubicMiles variable 1924
StdConvs.vuCubicMillimeters 1924
StdConvs.vuCubicMillimeters variable 1924
StdConvs.vuCubicYards 1925
StdConvs.vuCubicYards variable 1925
StdConvs.vuDecaLiters 1925
StdConvs.vuDecaLiters variable 1925
StdConvs.vuDecasteres 1925
StdConvs.vuDecasteres variable 1925
StdConvs.vuDeciLiters 1926
StdConvs.vuDeciLiters variable 1926
StdConvs.vuDecisteres 1926
StdConvs.vuDecisteres variable 1926
StdConvs.vuDryBuckets 1926
StdConvs.vuDryBuckets variable 1926

StdConvs.vuDryBushels 1927
StdConvs.vuDryBushels variable 1927
StdConvs.vuDryGallons 1927
StdConvs.vuDryGallons variable 1927
StdConvs.vuDryPecks 1927
StdConvs.vuDryPecks variable 1927
StdConvs.vuDryPints 1928
StdConvs.vuDryPints variable 1928
StdConvs.vuDryQuarts 1928
StdConvs.vuDryQuarts variable 1928
StdConvs.vuFluidCups 1928
StdConvs.vuFluidCups variable 1928
StdConvs.vuFluidGallons 1929
StdConvs.vuFluidGallons variable 1929
StdConvs.vuFluidGills 1929
StdConvs.vuFluidGills variable 1929
StdConvs.vuFluidOunces 1929
StdConvs.vuFluidOunces variable 1929
StdConvs.vuFluidPints 1930
StdConvs.vuFluidPints variable 1930
StdConvs.vuFluidQuarts 1930
StdConvs.vuFluidQuarts variable 1930
StdConvs.vuFluidTablespoons 1930
StdConvs.vuFluidTablespoons variable 1930
StdConvs.vuFluidTeaspoons 1931
StdConvs.vuFluidTeaspoons variable 1931
StdConvs.vuHectoLiters 1931
StdConvs.vuHectoLiters variable 1931
StdConvs.vuKiloLiters 1931
StdConvs.vuKiloLiters variable 1931
StdConvs.vuLiters 1932
StdConvs.vuLiters variable 1932
StdConvs.vuMilliLiters 1932
StdConvs.vuMilliLiters variable 1932
StdConvs.vuSteres 1932
StdConvs.vuSteres variable 1932
StdConvs.vuUKBuckets 1933
StdConvs.vuUKBuckets variable 1933
StdConvs.vuUKBushels 1933
StdConvs.vuUKBushels variable 1933
StdConvs.vuUKGallons 1933
StdConvs.vuUKGallons variable 1933
StdConvs.vuUKGills 1934
StdConvs.vuUKGills variable 1934
StdConvs.vuUKOunces 1934
StdConvs.vuUKOunces variable 1934
StdConvs.vuUKPecks 1934
StdConvs.vuUKPecks variable 1934
StdConvs.vuUKPints 1935
StdConvs.vuUKPints variable 1935
StdConvs.vuUKPottles 1935
StdConvs.vuUKPottles variable 1935
StdConvs.vuUKQuarts 1935
StdConvs.vuUKQuarts variable 1935
StdCtrls 1959
StdCtrls namespace 1959
    Classes 1961
    Structs, Records, Enums 2906
    Types 2915
StdCtrls.TButton 1961
StdCtrls.TButton.Action 1965
StdCtrls.TButton.Align 1966
StdCtrls.TButton.Anchors 1969
StdCtrls.TButton.BiDiMode 1970
StdCtrls.TButton.Cancel 1970
StdCtrls.TButton.Caption 1971
StdCtrls.TButton.CommandLinkHint 1973
StdCtrls.TButton.Constraints 1973
StdCtrls.TButton.Default 1974
StdCtrls.TButton.DisabledImageIndex 1975
StdCtrls.TButton.DoubleBuffered 1976
StdCtrls.TButton.DragCursor 1977
StdCtrls.TButton.DragKind 1978
StdCtrls.TButton.DragMode 1978
StdCtrls.TButton.DropDownMenu 1980
StdCtrls.TButton.ElevationRequired 1980
StdCtrls.TButton.Enabled 1981
StdCtrls.TButton.Font 1981

StdCtrls.TButton.HotImageIndex 1982
StdCtrls.TButton.ImageAlignment 1983
StdCtrls.TButton.ImageIndex 1983
StdCtrls.TButton.ImageMargins 1984
StdCtrls.TButton.Images 1984
StdCtrls.TButton.ModalResult 1985
StdCtrls.TButton.OnClick 1998
StdCtrls.TButton.OnContextPopup 2000
StdCtrls.TButton.OnDragDrop 2001
StdCtrls.TButton.OnDragOver 2004
StdCtrls.TButton.OnDropDownClick 2007
StdCtrls.TButton.OnEndDock 2008
StdCtrls.TButton.OnEndDrag 2008
StdCtrls.TButton.OnEnter 2009
StdCtrls.TButton.OnExit 2011
StdCtrls.TButton.OnKeyDown 2014
StdCtrls.TButton.OnKeyPress 2018
StdCtrls.TButton.OnKeyUp 2019
StdCtrls.TButton.OnMouseActivate 2021
StdCtrls.TButton.OnMouseDown 2022
StdCtrls.TButton.OnMouseEnter 2024
StdCtrls.TButton.OnMouseLeave 2024
StdCtrls.TButton.OnMouseMove 2024
StdCtrls.TButton.OnMouseUp 2027
StdCtrls.TButton.OnStartDock 2029
StdCtrls.TButton.OnStartDrag 2030
StdCtrls.TButton.ParentBiDiMode 1985
StdCtrls.TButton.ParentDoubleBuffered 1986
StdCtrls.TButton.ParentFont 1986
StdCtrls.TButton.ParentShowHint 1987
StdCtrls.TButton.PopupMenu 1987
StdCtrls.TButton.PressedImageIndex 1992
StdCtrls.TButton.SelectedImageIndex 1993
StdCtrls.TButton.ShowHint 1993
StdCtrls.TButton.Style 1995
StdCtrls.TButton.TabOrder 1995
StdCtrls.TButton.TabStop 1996
StdCtrls.TButton.Visible 1996
StdCtrls.TButton.WordWrap 1997
StdCtrls.TButtonActionLink 2030
StdCtrls.TButtonActionLink.AssignClient 2032
StdCtrls.TButtonActionLink.FClient 2031
StdCtrls.TButtonActionLink.IsCheckedLinked 2032
StdCtrls.TButtonActionLink.SetChecked 2032
StdCtrls.TButtonActionLinkClass 2915
StdCtrls.TButtonActionLinkClass type 2915
StdCtrls.TButtonControl 2033
StdCtrls.TButtonControl.ActionChange 2034
StdCtrls.TButtonControl.Checked 2037
StdCtrls.TButtonControl.ClicksDisabled 2039
StdCtrls.TButtonControl.Create 2034
StdCtrls.TButtonControl.CreateParams 2035
StdCtrls.TButtonControl.GetActionLinkClass 2035
StdCtrls.TButtonControl.GetChecked 2036
StdCtrls.TButtonControl.SetChecked 2036
StdCtrls.TButtonControl.WndProc 2036
StdCtrls.TButtonControl.WordWrap 2039
StdCtrls.TCheckBox 2040
StdCtrls.TCheckBox.Action 2043
StdCtrls.TCheckBox.Align 2043
StdCtrls.TCheckBox.Alignment 2047
StdCtrls.TCheckBox.AllowGrayed 2047
StdCtrls.TCheckBox.Anchors 2048
StdCtrls.TCheckBox.BiDiMode 2049
StdCtrls.TCheckBox.Caption 2049
StdCtrls.TCheckBox.Checked 2052
StdCtrls.TCheckBox.Color 2054
StdCtrls.TCheckBox.Constraints 2055
StdCtrls.TCheckBox.Ctl3D 2055
StdCtrls.TCheckBox.DoubleBuffered 2056
StdCtrls.TCheckBox.DragCursor 2057
StdCtrls.TCheckBox.DragKind 2058
StdCtrls.TCheckBox.DragMode 2058
StdCtrls.TCheckBox.Enabled 2060
StdCtrls.TCheckBox.Font 2060
StdCtrls.TCheckBox.OnClick 2074
StdCtrls.TCheckBox.OnContextPopup 2076
StdCtrls.TCheckBox.OnDragDrop 2077

StdCtrls.TCheckBox.OnDragOver 2080
StdCtrls.TCheckBox.OnEndDock 2083
StdCtrls.TCheckBox.OnEndDrag 2084
StdCtrls.TCheckBox.OnEnter 2085
StdCtrls.TCheckBox.OnExit 2087
StdCtrls.TCheckBox.OnKeyDown 2090
StdCtrls.TCheckBox.OnKeyPress 2093
StdCtrls.TCheckBox.OnKeyUp 2095
StdCtrls.TCheckBox.OnMouseActivate 2096
StdCtrls.TCheckBox.OnMouseDown 2097
StdCtrls.TCheckBox.OnMouseEnter 2100
StdCtrls.TCheckBox.OnMouseLeave 2100
StdCtrls.TCheckBox.OnMouseMove 2100
StdCtrls.TCheckBox.OnMouseUp 2102
StdCtrls.TCheckBox.OnStartDock 2105
StdCtrls.TCheckBox.OnStartDrag 2105
StdCtrls.TCheckBox.ParentBiDiMode 2061
StdCtrls.TCheckBox.ParentColor 2062
StdCtrls.TCheckBox.ParentCtl3D 2062
StdCtrls.TCheckBox.ParentDoubleBuffered 2062
StdCtrls.TCheckBox.ParentFont 2063
StdCtrls.TCheckBox.ParentShowHint 2063
StdCtrls.TCheckBox.PopupMenu 2064
StdCtrls.TCheckBox.ShowHint 2069
StdCtrls.TCheckBox.State 2070
StdCtrls.TCheckBox.TabOrder 2071
StdCtrls.TCheckBox.TabStop 2072
StdCtrls.TCheckBox.Visible 2072
StdCtrls.TCheckBox.WordWrap 2073
StdCtrls.TCheckBoxState 2906
StdCtrls.TCheckBoxState enumeration 2906
StdCtrls.TComboBox 2106
StdCtrls.TComboBox.Align 2112
StdCtrls.TComboBox.Anchors 2115
StdCtrls.TComboBox.AutoCloseUp 2116
StdCtrls.TComboBox.AutoComplete 2116
StdCtrls.TComboBox.AutoCompleteDelay 2117
StdCtrls.TComboBox.AutoDropDown 2117
StdCtrls.TComboBox.BevelEdges 2117
StdCtrls.TComboBox.BevelInner 2118
StdCtrls.TComboBox.BevelKind 2118
StdCtrls.TComboBox.BevelOuter 2119
StdCtrls.TComboBox.BiDiMode 2119
StdCtrls.TComboBox.CharCase 2120
StdCtrls.TComboBox.Color 2120
StdCtrls.TComboBox.Constraints 2121
StdCtrls.TComboBox.Ctl3D 2121
StdCtrls.TComboBox.DoubleBuffered 2122
StdCtrls.TComboBox.DragCursor 2124
StdCtrls.TComboBox.DragKind 2124
StdCtrls.TComboBox.DragMode 2124
StdCtrls.TComboBox.DropDownCount 2126
StdCtrls.TComboBox.Enabled 2126
StdCtrls.TComboBox.Font 2127
StdCtrls.TComboBox.ImeMode 2128
StdCtrls.TComboBox.ImeName 2128
StdCtrls.TComboBox.ItemHeight 2129
StdCtrls.TComboBox.ItemIndex 2130
StdCtrls.TComboBox.Items 2131
StdCtrls.TComboBox.MaxLength 2133
StdCtrls.TComboBox.OnChange 2149
StdCtrls.TComboBox.OnClick 2152
StdCtrls.TComboBox.OnCloseUp 2153
StdCtrls.TComboBox.OnContextPopup 2154
StdCtrls.TComboBox.OnDblClick 2154
StdCtrls.TComboBox.OnDragDrop 2155
StdCtrls.TComboBox.OnDragOver 2158
StdCtrls.TComboBox.OnDrawItem 2161
StdCtrls.TComboBox.OnDropDown 2161
StdCtrls.TComboBox.OnEndDock 2163
StdCtrls.TComboBox.OnEndDrag 2163
StdCtrls.TComboBox.OnEnter 2164
StdCtrls.TComboBox.OnExit 2167
StdCtrls.TComboBox.OnKeyDown 2170
StdCtrls.TComboBox.OnKeyPress 2173
StdCtrls.TComboBox.OnKeyUp 2174
StdCtrls.TComboBox.OnMeasureItem 2176
StdCtrls.TComboBox.OnMouseEnter 2177

StdCtrls.TComboBox.OnMouseLeave 2177
StdCtrls.TComboBox.OnSelect 2177
StdCtrls.TComboBox.OnStartDock 2177
StdCtrls.TComboBox.OnStartDrag 2178
StdCtrls.TComboBox.ParentBiDiMode 2134
StdCtrls.TComboBox.ParentColor 2134
StdCtrls.TComboBox.ParentCtl3D 2135
StdCtrls.TComboBox.ParentDoubleBuffered 2135
StdCtrls.TComboBox.ParentFont 2135
StdCtrls.TComboBox.ParentShowHint 2136
StdCtrls.TComboBox.PopupMenu 2136
StdCtrls.TComboBox.ShowHint 2141
StdCtrls.TComboBox.Sorted 2143
StdCtrls.TComboBox.Style 2144
StdCtrls.TComboBox.TabOrder 2145
StdCtrls.TComboBox.TabStop 2145
StdCtrls.TComboBox.Text 2146
StdCtrls.TComboBox.TextHint 2147
StdCtrls.TComboBox.Visible 2147
StdCtrls.TComboBoxStyle 2907
StdCtrls.TComboBoxStyle enumeration 2907
StdCtrls.TCustomButton 2179
StdCtrls.TCustomButton.ActionChange 2182
StdCtrls.TCustomButton.Cancel 2187
StdCtrls.TCustomButton.Click 2183
StdCtrls.TCustomButton.CommandLinkHint 2187
StdCtrls.TCustomButton.Create 2180
StdCtrls.TCustomButton.CreateParams 2183
StdCtrls.TCustomButton.CreateWnd 2184
StdCtrls.TCustomButton.Default 2188
StdCtrls.TCustomButton.Destroy 2181
StdCtrls.TCustomButton.DisabledImageIndex 2189
StdCtrls.TCustomButton.DropDownMenu 2189
StdCtrls.TCustomButton.ElevationRequired 2190
StdCtrls.TCustomButton.GetActionLinkClass 2184
StdCtrls.TCustomButton.HotImageIndex 2190
StdCtrls.TCustomButton.ImageAlignment 2191
StdCtrls.TCustomButton.ImageIndex 2191
StdCtrls.TCustomButton.ImageMargins 2192
StdCtrls.TCustomButton.Images 2192
StdCtrls.TCustomButton.Loaded 2184
StdCtrls.TCustomButton.ModalResult 2193
StdCtrls.TCustomButton.Notification 2185
StdCtrls.TCustomButton.OnDropDownClick 2195
StdCtrls.TCustomButton.PressedImageIndex 2193
StdCtrls.TCustomButton.SelectedImageIndex 2194
StdCtrls.TCustomButton.SetButtonStyle 2185
StdCtrls.TCustomButton.Style 2194
StdCtrls.TCustomButton.TabStop 2195
StdCtrls.TCustomButton.TButtonStyle 2181
StdCtrls.TCustomButton.TButtonStyle enumeration 2181
StdCtrls.TCustomButton.UpdateImageList 2186
StdCtrls.TCustomButton.UpdateImages 2186
StdCtrls.TCustomButton.UseRightToLeftAlignment 2186
StdCtrls.TCustomCheckBox 2196
StdCtrls.TCustomCheckBox.Alignment 2200
StdCtrls.TCustomCheckBox.AllowGrayed 2201
StdCtrls.TCustomCheckBox.Click 2197
StdCtrls.TCustomCheckBox.Create 2197
StdCtrls.TCustomCheckBox.CreateParams 2198
StdCtrls.TCustomCheckBox.CreateWnd 2198
StdCtrls.TCustomCheckBox.GetChecked 2199
StdCtrls.TCustomCheckBox.GetControlsAlignment 2199
StdCtrls.TCustomCheckBox.SetChecked 2199
StdCtrls.TCustomCheckBox.State 2201
StdCtrls.TCustomCheckBox.TabStop 2202
StdCtrls.TCustomCheckBox.Toggle 2200
StdCtrls.TCustomCombo 2203
StdCtrls.TCustomCombo.AddItem 2209
StdCtrls.TCustomCombo.AdjustDropDown 2211
StdCtrls.TCustomCombo.Canvas 2224
StdCtrls.TCustomCombo.Change 2211
StdCtrls.TCustomCombo.Clear 2212
StdCtrls.TCustomCombo.ClearSelection 2212
StdCtrls.TCustomCombo.CloseUp 2212
StdCtrls.TCustomCombo.ComboWndProc 2213
StdCtrls.TCustomCombo.CopySelection 2213
StdCtrls.TCustomCombo.Create 2205

StdCtrls.TCustomCombo.CreateWnd 2214
StdCtrls.TCustomCombo.DeleteSelected 2214
StdCtrls.TCustomCombo.Destroy 2205
StdCtrls.TCustomCombo.DestroyWindowHandle 2214
StdCtrls.TCustomCombo.DropDown 2215
StdCtrls.TCustomCombo.DropDownCount 2225
StdCtrls.TCustomCombo.DroppedDown 2225
StdCtrls.TCustomCombo.EditHandle 2225
StdCtrls.TCustomCombo.EditWndProc 2215
StdCtrls.TCustomCombo.FDefEditProc 2206
StdCtrls.TCustomCombo.FDefListProc 2206
StdCtrls.TCustomCombo.FDropHandle 2207
StdCtrls.TCustomCombo.FDroppingDown 2207
StdCtrls.TCustomCombo.FEditHandle 2207
StdCtrls.TCustomCombo.FEditInstance 2207
StdCtrls.TCustomCombo.FFocusChanged 2207
StdCtrls.TCustomCombo.FIsFocused 2208
StdCtrls.TCustomCombo.FListHandle 2208
StdCtrls.TCustomCombo.FListInstance 2208
StdCtrls.TCustomCombo.Focused 2215
StdCtrls.TCustomCombo.FSaveIndex 2208
StdCtrls.TCustomCombo.GetCount 2216
StdCtrls.TCustomCombo.GetDroppedDown 2216
StdCtrls.TCustomCombo.GetItemCount 2216
StdCtrls.TCustomCombo.GetItemHt 2217
StdCtrls.TCustomCombo.GetItemIndex 2217
StdCtrls.TCustomCombo.GetItemsClass 2217
StdCtrls.TCustomCombo.GetSelLength 2218
StdCtrls.TCustomCombo.GetSelStart 2218
StdCtrls.TCustomCombo.ItemCount 2226
StdCtrls.TCustomCombo.ItemHeight 2226
StdCtrls.TCustomCombo.Items 2227
StdCtrls.TCustomCombo.ListHandle 2229
StdCtrls.TCustomCombo.ListWndProc 2218
StdCtrls.TCustomCombo.Loaded 2219
StdCtrls.TCustomCombo.MaxLength 2229
StdCtrls.TCustomCombo.OnChange 2237
StdCtrls.TCustomCombo.OnCloseUp 2239
StdCtrls.TCustomCombo.OnDropDown 2239
StdCtrls.TCustomCombo.OnSelect 2241
StdCtrls.TCustomCombo.ParentColor 2230
StdCtrls.TCustomCombo.Select 2219
StdCtrls.TCustomCombo.SelectAll 2220
StdCtrls.TCustomCombo.SelLength 2230
StdCtrls.TCustomCombo.SelStart 2233
StdCtrls.TCustomCombo.SetDropDownCount 2220
StdCtrls.TCustomCombo.SetDroppedDown 2220
StdCtrls.TCustomCombo.SetItemHeight 2221
StdCtrls.TCustomCombo.SetItemIndex 2221
StdCtrls.TCustomCombo.SetItems 2221
StdCtrls.TCustomCombo.SetMaxLength 2222
StdCtrls.TCustomCombo.SetSelLength 2222
StdCtrls.TCustomCombo.SetSelStart 2222
StdCtrls.TCustomCombo.TabStop 2236
StdCtrls.TCustomCombo.WndProc 2223
StdCtrls.TCustomComboBox 2241
StdCtrls.TCustomComboBox.AutoCloseUp 2250
StdCtrls.TCustomComboBox.AutoComplete 2251
StdCtrls.TCustomComboBox.AutoCompleteDelay 2252
StdCtrls.TCustomComboBox.AutoDropDown 2253
StdCtrls.TCustomComboBox.CharCase 2253
StdCtrls.TCustomComboBox.Create 2244
StdCtrls.TCustomComboBox.CreateParams 2245
StdCtrls.TCustomComboBox.CreateWnd 2245
StdCtrls.TCustomComboBox.Destroy 2244
StdCtrls.TCustomComboBox.DestroyWnd 2246
StdCtrls.TCustomComboBox.DoSetTextHint 2246
StdCtrls.TCustomComboBox.DrawItem 2246
StdCtrls.TCustomComboBox.GetItemCount 2247
StdCtrls.TCustomComboBox.GetItemHt 2247
StdCtrls.TCustomComboBox.GetItemsClass 2247
StdCtrls.TCustomComboBox.GetSelText 2248
StdCtrls.TCustomComboBox.KeyPress 2248
StdCtrls.TCustomComboBox.MeasureItem 2249
StdCtrls.TCustomComboBox.OnDrawItem 2256
StdCtrls.TCustomComboBox.OnMeasureItem 2257
StdCtrls.TCustomComboBox.SelectItem 2249
StdCtrls.TCustomComboBox.SelText 2253

StdCtrls.TCustomComboBox.SetStyle 2249
StdCtrls.TCustomComboBox.Sorted 2254
StdCtrls.TCustomComboBox.Style 2255
StdCtrls.TCustomComboBox.TextHint 2256
StdCtrls.TCustomComboBox.WndProc 2250
StdCtrls.TCustomComboBoxStrings 2257
StdCtrls.TCustomComboBoxStrings.Clear 2259
StdCtrls.TCustomComboBoxStrings.ComboBox 2263
StdCtrls.TCustomComboBoxStrings.Create 2258
StdCtrls.TCustomComboBoxStrings.Delete 2260
StdCtrls.TCustomComboBoxStrings.Destroy 2259
StdCtrls.TCustomComboBoxStrings.Get 2260
StdCtrls.TCustomComboBoxStrings.GetCount 2261
StdCtrls.TCustomComboBoxStrings.GetObject 2261
StdCtrls.TCustomComboBoxStrings.IndexOf 2261
StdCtrls.TCustomComboBoxStrings.PutObject 2262
StdCtrls.TCustomComboBoxStrings.SetUpdateState 2262
StdCtrls.TCustomComboBoxStringsClass 2915
StdCtrls.TCustomComboBoxStringsClass type 2915
StdCtrls.TCustomEdit 2263
StdCtrls.TCustomEdit.Alignment 2283
StdCtrls.TCustomEdit.AutoSelect 2284
StdCtrls.TCustomEdit.AutoSize 2284
StdCtrls.TCustomEdit.BorderStyle 2285
StdCtrls.TCustomEdit.CanUndo 2285
StdCtrls.TCustomEdit.Change 2266
StdCtrls.TCustomEdit.CharCase 2286
StdCtrls.TCustomEdit.Clear 2267
StdCtrls.TCustomEdit.ClearSelection 2267
StdCtrls.TCustomEdit.ClearUndo 2268
StdCtrls.TCustomEdit.CopyToClipboard 2268
StdCtrls.TCustomEdit.Create 2265
StdCtrls.TCustomEdit.CreateParams 2269
StdCtrls.TCustomEdit.CreateWindowHandle 2270
StdCtrls.TCustomEdit.CreateWnd 2270
StdCtrls.TCustomEdit.CutToClipboard 2271
StdCtrls.TCustomEdit.DefaultHandler 2272
StdCtrls.TCustomEdit.DestroyWnd 2273
StdCtrls.TCustomEdit.DoSetMaxLength 2273
StdCtrls.TCustomEdit.DoSetTextHint 2273
StdCtrls.TCustomEdit.GetControlsAlignment 2274
StdCtrls.TCustomEdit.GetSelLength 2274
StdCtrls.TCustomEdit.GetSelStart 2274
StdCtrls.TCustomEdit.GetSelText 2275
StdCtrls.TCustomEdit.GetSelTextBuf 2275
StdCtrls.TCustomEdit.HideSelection 2286
StdCtrls.TCustomEdit.MaxLength 2286
StdCtrls.TCustomEdit.Modified 2287
StdCtrls.TCustomEdit.NumbersOnly 2287
StdCtrls.TCustomEdit.OEMConvert 2288
StdCtrls.TCustomEdit.OnChange 2299
StdCtrls.TCustomEdit.ParentColor 2288
StdCtrls.TCustomEdit.PasswordChar 2289
StdCtrls.TCustomEdit.PasteFromClipboard 2276
StdCtrls.TCustomEdit.ReadOnly 2289
StdCtrls.TCustomEdit.SelectAll 2278
StdCtrls.TCustomEdit.SelLength 2289
StdCtrls.TCustomEdit.SelStart 2293
StdCtrls.TCustomEdit.SelText 2296
StdCtrls.TCustomEdit.SetAlignment 2279
StdCtrls.TCustomEdit.SetAutoSize 2279
StdCtrls.TCustomEdit.SetSelLength 2280
StdCtrls.TCustomEdit.SetSelStart 2280
StdCtrls.TCustomEdit.SetSelText 2281
StdCtrls.TCustomEdit.SetSelTextBuf 2281
StdCtrls.TCustomEdit.TabStop 2297
StdCtrls.TCustomEdit.Text 2297
StdCtrls.TCustomEdit.TextHint 2299
StdCtrls.TCustomEdit.Undo 2282
StdCtrls.TCustomGroupBox 2299
StdCtrls.TCustomGroupBox.AdjustClientRect 2301
StdCtrls.TCustomGroupBox.Create 2300
StdCtrls.TCustomGroupBox.CreateParams 2301
StdCtrls.TCustomGroupBox.Paint 2302
StdCtrls.TCustomLabel 2302
StdCtrls.TCustomLabel.AdjustBounds 2304
StdCtrls.TCustomLabel.Alignment 2307
StdCtrls.TCustomLabel.AutoSize 2307

StdCtrls.TCustomLabel.Canvas 2308
StdCtrls.TCustomLabel.Caption 2310
StdCtrls.TCustomLabel.Create 2303
StdCtrls.TCustomLabel.DoDrawText 2304
StdCtrls.TCustomLabel.EllipsisPosition 2313
StdCtrls.TCustomLabel.FocusControl 2313
StdCtrls.TCustomLabel.GetLabelText 2305
StdCtrls.TCustomLabel.GlowSize 2314
StdCtrls.TCustomLabel.Layout 2314
StdCtrls.TCustomLabel.Loaded 2305
StdCtrls.TCustomLabel.Notification 2305
StdCtrls.TCustomLabel.Paint 2306
StdCtrls.TCustomLabel.SetAutoSize 2306
StdCtrls.TCustomLabel.ShowAccelChar 2314
StdCtrls.TCustomLabel.Transparent 2315
StdCtrls.TCustomLabel.WordWrap 2316
StdCtrls.TCustomListBox 2316
StdCtrls.TCustomListBox.AddItem 2320
StdCtrls.TCustomListBox.AutoComplete 2340
StdCtrls.TCustomListBox.AutoCompleteDelay 2341
StdCtrls.TCustomListBox.BorderStyle 2341
StdCtrls.TCustomListBox.Canvas 2341
StdCtrls.TCustomListBox.Clear 2321
StdCtrls.TCustomListBox.ClearSelection 2321
StdCtrls.TCustomListBox.Columns 2344
StdCtrls.TCustomListBox.CopySelection 2322
StdCtrls.TCustomListBox.Count 2344
StdCtrls.TCustomListBox.Create 2318
StdCtrls.TCustomListBox.CreateParams 2322
StdCtrls.TCustomListBox.CreateWnd 2323
StdCtrls.TCustomListBox.DeleteSelected 2324
StdCtrls.TCustomListBox.DeleteString 2324
StdCtrls.TCustomListBox.Destroy 2319
StdCtrls.TCustomListBox.DestroyWnd 2324
StdCtrls.TCustomListBox.DoFindData 2325
StdCtrls.TCustomListBox.DoGetData 2325
StdCtrls.TCustomListBox.DoGetDataObject 2326
StdCtrls.TCustomListBox.DragCanceled 2326
StdCtrls.TCustomListBox.DrawItem 2326
StdCtrls.TCustomListBox.ExtendedSelect 2345
StdCtrls.TCustomListBox.FMoving 2319
StdCtrls.TCustomListBox.GetCount 2327
StdCtrls.TCustomListBox.GetItemData 2327
StdCtrls.TCustomListBox.GetItemIndex 2328
StdCtrls.TCustomListBox.GetSelCount 2328
StdCtrls.TCustomListBox.GetSelected 2328
StdCtrls.TCustomListBox.IntegralHeight 2345
StdCtrls.TCustomListBox.InternalGetItemData 2328
StdCtrls.TCustomListBox.InternalSetItemData 2329
StdCtrls.TCustomListBox.ItemAtPos 2329
StdCtrls.TCustomListBox.ItemHeight 2346
StdCtrls.TCustomListBox.ItemRect 2331
StdCtrls.TCustomListBox.Items 2346
StdCtrls.TCustomListBox.KeyPress 2332
StdCtrls.TCustomListBox.LoadRecreateItems 2333
StdCtrls.TCustomListBox.MeasureItem 2333
StdCtrls.TCustomListBox.OnData 2356
StdCtrls.TCustomListBox.OnDataFind 2356
StdCtrls.TCustomListBox.OnDataObject 2357
StdCtrls.TCustomListBox.OnDrawItem 2357
StdCtrls.TCustomListBox.OnMeasureItem 2359
StdCtrls.TCustomListBox.ParentColor 2349
StdCtrls.TCustomListBox.ResetContent 2334
StdCtrls.TCustomListBox.SaveRecreateItems 2334
StdCtrls.TCustomListBox.ScrollWidth 2349
StdCtrls.TCustomListBox.SelectAll 2334
StdCtrls.TCustomListBox.Selected 2350
StdCtrls.TCustomListBox.SetItemData 2334
StdCtrls.TCustomListBox.SetItemIndex 2335
StdCtrls.TCustomListBox.SetMultiSelect 2335
StdCtrls.TCustomListBox.Sorted 2352
StdCtrls.TCustomListBox.Style 2353
StdCtrls.TCustomListBox.TabStop 2354
StdCtrls.TCustomListBox.TabWidth 2354
StdCtrls.TCustomListBox.TopIndex 2354
StdCtrls.TCustomListBox.WndProc 2335
StdCtrls.TCustomMemo 2361
StdCtrls.TCustomMemo.CaretPos 2368

StdCtrls.TCustomMemo.Create 2363
StdCtrls.TCustomMemo.CreateParams 2364
StdCtrls.TCustomMemo.CreateWindowHandle 2365
StdCtrls.TCustomMemo.Destroy 2364
StdCtrls.TCustomMemo.GetCaretPos 2365
StdCtrls.TCustomMemo.KeyPress 2366
StdCtrls.TCustomMemo.Lines 2368
StdCtrls.TCustomMemo.Loaded 2366
StdCtrls.TCustomMemo.ScrollBars 2369
StdCtrls.TCustomMemo.SetCaretPos 2366
StdCtrls.TCustomMemo.SetLines 2367
StdCtrls.TCustomMemo.SetScrollBars 2367
StdCtrls.TCustomMemo.SetWordWrap 2367
StdCtrls.TCustomMemo.WantReturns 2372
StdCtrls.TCustomMemo.WantTabs 2372
StdCtrls.TCustomMemo.WordWrap 2372
StdCtrls.TCustomStaticText 2375
StdCtrls.TCustomStaticText.Alignment 2378
StdCtrls.TCustomStaticText.AutoSize 2379
StdCtrls.TCustomStaticText.BorderStyle 2379
StdCtrls.TCustomStaticText.Create 2376
StdCtrls.TCustomStaticText.CreateParams 2376
StdCtrls.TCustomStaticText.FocusControl 2379
StdCtrls.TCustomStaticText.Loaded 2377
StdCtrls.TCustomStaticText.Notification 2377
StdCtrls.TCustomStaticText.SetAutoSize 2378
StdCtrls.TCustomStaticText.ShowAccelChar 2380
StdCtrls.TCustomStaticText.Transparent 2380
StdCtrls.TDrawItemEvent 2916
StdCtrls.TDrawItemEvent type 2916
StdCtrls.TEdit 2380
StdCtrls.TEdit.Align 2385
StdCtrls.TEdit.Alignment 2388
StdCtrls.TEdit.Anchors 2389
StdCtrls.TEdit.AutoSelect 2389
StdCtrls.TEdit.AutoSize 2390
StdCtrls.TEdit.BevelEdges 2391
StdCtrls.TEdit.BevelInner 2391
StdCtrls.TEdit.BevelKind 2392
StdCtrls.TEdit.BevelOuter 2392
StdCtrls.TEdit.BevelWidth 2392
StdCtrls.TEdit.BiDiMode 2393
StdCtrls.TEdit.BorderStyle 2393
StdCtrls.TEdit.CharCase 2394
StdCtrls.TEdit.Color 2394
StdCtrls.TEdit.Constraints 2395
StdCtrls.TEdit.Ctl3D 2395
StdCtrls.TEdit.DoubleBuffered 2396
StdCtrls.TEdit.DragCursor 2398
StdCtrls.TEdit.DragKind 2398
StdCtrls.TEdit.DragMode 2398
StdCtrls.TEdit.Enabled 2400
StdCtrls.TEdit.Font 2400
StdCtrls.TEdit.HideSelection 2402
StdCtrls.TEdit.ImeMode 2402
StdCtrls.TEdit.ImeName 2403
StdCtrls.TEdit.MaxLength 2403
StdCtrls.TEdit.NumbersOnly 2404
StdCtrls.TEdit.OEMConvert 2404
StdCtrls.TEdit.OnChange 2419
StdCtrls.TEdit.OnClick 2419
StdCtrls.TEdit.OnContextPopup 2421
StdCtrls.TEdit.OnDblClick 2422
StdCtrls.TEdit.OnDragDrop 2422
StdCtrls.TEdit.OnDragOver 2425
StdCtrls.TEdit.OnEndDock 2428
StdCtrls.TEdit.OnEndDrag 2429
StdCtrls.TEdit.OnEnter 2430
StdCtrls.TEdit.OnExit 2432
StdCtrls.TEdit.OnKeyDown 2435
StdCtrls.TEdit.OnKeyPress 2438
StdCtrls.TEdit.OnKeyUp 2440
StdCtrls.TEdit.OnMouseActivate 2441
StdCtrls.TEdit.OnMouseDown 2442
StdCtrls.TEdit.OnMouseEnter 2445
StdCtrls.TEdit.OnMouseLeave 2445
StdCtrls.TEdit.OnMouseMove 2445
StdCtrls.TEdit.OnMouseUp 2447

StdCtrls.TEdit.OnStartDock 2450
StdCtrls.TEdit.OnStartDrag 2450
StdCtrls.TEdit.ParentBiDiMode 2404
StdCtrls.TEdit.ParentColor 2405
StdCtrls.TEdit.ParentCtl3D 2405
StdCtrls.TEdit.ParentDoubleBuffered 2406
StdCtrls.TEdit.ParentFont 2406
StdCtrls.TEdit.ParentShowHint 2406
StdCtrls.TEdit.PasswordChar 2407
StdCtrls.TEdit.PopupMenu 2407
StdCtrls.TEdit.ReadOnly 2412
StdCtrls.TEdit.ShowHint 2413
StdCtrls.TEdit.TabOrder 2414
StdCtrls.TEdit.TabStop 2415
StdCtrls.TEdit.Text 2415
StdCtrls.TEdit.TextHint 2416
StdCtrls.TEdit.Visible 2417
StdCtrls.TEditCharCase 2907
StdCtrls.TEditCharCase enumeration 2907
StdCtrls.TEllipsisPosition 2908
StdCtrls.TEllipsisPosition enumeration 2908
StdCtrls.TFNDrawText 2916
StdCtrls.TFNDrawText type 2916
StdCtrls.TGroupBox 2451
StdCtrls.TGroupBox.Align 2454
StdCtrls.TGroupBox.Anchors 2457
StdCtrls.TGroupBox.BiDiMode 2458
StdCtrls.TGroupBox.Caption 2458
StdCtrls.TGroupBox.Color 2461
StdCtrls.TGroupBox.Constraints 2462
StdCtrls.TGroupBox.Ctl3D 2462
StdCtrls.TGroupBox.DockSite 2463
StdCtrls.TGroupBox.DoubleBuffered 2464
StdCtrls.TGroupBox.DragCursor 2465
StdCtrls.TGroupBox.DragKind 2466
StdCtrls.TGroupBox.DragMode 2466
StdCtrls.TGroupBox.Enabled 2468
StdCtrls.TGroupBox.Font 2468
StdCtrls.TGroupBox.OnAlignInsertBefore 2482
StdCtrls.TGroupBox.OnAlignPosition 2482
StdCtrls.TGroupBox.OnClick 2483
StdCtrls.TGroupBox.OnContextPopup 2485
StdCtrls.TGroupBox.OnDblClick 2486
StdCtrls.TGroupBox.OnDockDrop 2486
StdCtrls.TGroupBox.OnDockOver 2488
StdCtrls.TGroupBox.OnDragDrop 2490
StdCtrls.TGroupBox.OnDragOver 2493
StdCtrls.TGroupBox.OnEndDock 2496
StdCtrls.TGroupBox.OnEndDrag 2497
StdCtrls.TGroupBox.OnEnter 2498
StdCtrls.TGroupBox.OnExit 2500
StdCtrls.TGroupBox.OnGetSiteInfo 2503
StdCtrls.TGroupBox.OnMouseActivate 2505
StdCtrls.TGroupBox.OnMouseDown 2506
StdCtrls.TGroupBox.OnMouseEnter 2508
StdCtrls.TGroupBox.OnMouseLeave 2508
StdCtrls.TGroupBox.OnMouseMove 2509
StdCtrls.TGroupBox.OnMouseUp 2511
StdCtrls.TGroupBox.OnStartDock 2513
StdCtrls.TGroupBox.OnStartDrag 2514
StdCtrls.TGroupBox.OnUnDock 2514
StdCtrls.TGroupBox.Padding 2469
StdCtrls.TGroupBox.ParentBackground 2469
StdCtrls.TGroupBox.ParentBiDiMode 2470
StdCtrls.TGroupBox.ParentColor 2470
StdCtrls.TGroupBox.ParentCtl3D 2471
StdCtrls.TGroupBox.ParentDoubleBuffered 2471
StdCtrls.TGroupBox.ParentFont 2471
StdCtrls.TGroupBox.ParentShowHint 2472
StdCtrls.TGroupBox.PopupMenu 2472
StdCtrls.TGroupBox.ShowHint 2477
StdCtrls.TGroupBox.TabOrder 2479
StdCtrls.TGroupBox.TabStop 2479
StdCtrls.TGroupBox.Visible 2480
StdCtrls.TImageAlignment 2909
StdCtrls.TImageAlignment enumeration 2909
StdCtrls.TImageMargins 2517
StdCtrls.TImageMargins.Assign 2518

StdCtrls.TImageMargins.Bottom 2520
StdCtrls.TImageMargins.Change 2519
StdCtrls.TImageMargins.Left 2520
StdCtrls.TImageMargins.OnChange 2521
StdCtrls.TImageMargins.Right 2520
StdCtrls.TImageMargins.Top 2521
StdCtrls.TLabel 2522
StdCtrls.TLabel.Align 2525
StdCtrls.TLabel.Alignment 2528
StdCtrls.TLabel.Anchors 2528
StdCtrls.TLabel.AutoSize 2529
StdCtrls.TLabel.BiDiMode 2530
StdCtrls.TLabel.Caption 2530
StdCtrls.TLabel.Color 2533
StdCtrls.TLabel.Constraints 2534
StdCtrls.TLabel.DragCursor 2534
StdCtrls.TLabel.DragKind 2535
StdCtrls.TLabel.DragMode 2535
StdCtrls.TLabel.EllipsisPosition 2537
StdCtrls.TLabel.Enabled 2537
StdCtrls.TLabel.FocusControl 2538
StdCtrls.TLabel.Font 2538
StdCtrls.TLabel.GlowSize 2539
StdCtrls.TLabel.Layout 2539
StdCtrls.TLabel.OnClick 2552
StdCtrls.TLabel.OnContextPopup 2554
StdCtrls.TLabel.OnDblClick 2554
StdCtrls.TLabel.OnDragDrop 2555
StdCtrls.TLabel.OnDragOver 2558
StdCtrls.TLabel.OnEndDock 2561
StdCtrls.TLabel.OnEndDrag 2561
StdCtrls.TLabel.OnMouseActivate 2562
StdCtrls.TLabel.OnMouseDown 2563
StdCtrls.TLabel.OnMouseEnter 2566
StdCtrls.TLabel.OnMouseLeave 2566
StdCtrls.TLabel.OnMouseMove 2566
StdCtrls.TLabel.OnMouseUp 2568
StdCtrls.TLabel.OnStartDock 2571
StdCtrls.TLabel.OnStartDrag 2571
StdCtrls.TLabel.ParentBiDiMode 2540
StdCtrls.TLabel.ParentColor 2540
StdCtrls.TLabel.ParentFont 2540
StdCtrls.TLabel.ParentShowHint 2541
StdCtrls.TLabel.PopupMenu 2542
StdCtrls.TLabel.ShowAccelChar 2547
StdCtrls.TLabel.ShowHint 2547
StdCtrls.TLabel.Transparent 2549
StdCtrls.TLabel.Visible 2549
StdCtrls.TLabel.WordWrap 2551
StdCtrls.TLBFindDataEvent 2917
StdCtrls.TLBFindDataEvent type 2917
StdCtrls.TLBGetDataEvent 2917
StdCtrls.TLBGetDataEvent type 2917
StdCtrls.TLBGetDataObjectEvent 2918
StdCtrls.TLBGetDataObjectEvent type 2918
StdCtrls.TListBox 2572
StdCtrls.TListBox.Align 2576
StdCtrls.TListBox.Anchors 2580
StdCtrls.TListBox.AutoComplete 2580
StdCtrls.TListBox.AutoCompleteDelay 2580
StdCtrls.TListBox.BevelEdges 2581
StdCtrls.TListBox.BevelInner 2581
StdCtrls.TListBox.BevelKind 2582
StdCtrls.TListBox.BevelOuter 2582
StdCtrls.TListBox.BevelWidth 2582
StdCtrls.TListBox.BiDiMode 2583
StdCtrls.TListBox.BorderStyle 2583
StdCtrls.TListBox.Color 2584
StdCtrls.TListBox.Columns 2585
StdCtrls.TListBox.Constraints 2585
StdCtrls.TListBox.Ctl3D 2586
StdCtrls.TListBox.DoubleBuffered 2586
StdCtrls.TListBox.DragCursor 2588
StdCtrls.TListBox.DragKind 2588
StdCtrls.TListBox.DragMode 2589
StdCtrls.TListBox.Enabled 2590
StdCtrls.TListBox.ExtendedSelect 2591
StdCtrls.TListBox.Font 2591

StdCtrls.TListBox.ImeMode 2592
StdCtrls.TListBox.ImeName 2593
StdCtrls.TListBox.IntegralHeight 2593
StdCtrls.TListBox.ItemHeight 2594
StdCtrls.TListBox.Items 2594
StdCtrls.TListBox.MultiSelect 2597
StdCtrls.TListBox.OnClick 2612
StdCtrls.TListBox.OnContextPopup 2614
StdCtrls.TListBox.OnData 2615
StdCtrls.TListBox.OnDataFind 2615
StdCtrls.TListBox.OnDataObject 2616
StdCtrls.TListBox.OnDblClick 2616
StdCtrls.TListBox.OnDragDrop 2616
StdCtrls.TListBox.OnDragOver 2619
StdCtrls.TListBox.OnDrawItem 2623
StdCtrls.TListBox.OnEndDock 2625
StdCtrls.TListBox.OnEndDrag 2625
StdCtrls.TListBox.OnEnter 2626
StdCtrls.TListBox.OnExit 2629
StdCtrls.TListBox.OnKeyDown 2631
StdCtrls.TListBox.OnKeyPress 2635
StdCtrls.TListBox.OnKeyUp 2636
StdCtrls.TListBox.OnMeasureItem 2638
StdCtrls.TListBox.OnMouseActivate 2639
StdCtrls.TListBox.OnMouseDown 2640
StdCtrls.TListBox.OnMouseEnter 2643
StdCtrls.TListBox.OnMouseLeave 2643
StdCtrls.TListBox.OnMouseMove 2643
StdCtrls.TListBox.OnMouseUp 2646
StdCtrls.TListBox.OnStartDock 2648
StdCtrls.TListBox.OnStartDrag 2648
StdCtrls.TListBox.ParentBiDiMode 2597
StdCtrls.TListBox.ParentColor 2598
StdCtrls.TListBox.ParentCtl3D 2598
StdCtrls.TListBox.ParentDoubleBuffered 2599
StdCtrls.TListBox.ParentFont 2599
StdCtrls.TListBox.ParentShowHint 2599
StdCtrls.TListBox.PopupMenu 2600
StdCtrls.TListBox.ScrollWidth 2605
StdCtrls.TListBox.ShowHint 2605
StdCtrls.TListBox.Sorted 2607
StdCtrls.TListBox.Style 2608
StdCtrls.TListBox.TabOrder 2609
StdCtrls.TListBox.TabStop 2609
StdCtrls.TListBox.TabWidth 2610
StdCtrls.TListBox.Visible 2610
StdCtrls.TListBoxStyle 2910
StdCtrls.TListBoxStyle enumeration 2910
StdCtrls.TMeasureItemEvent 2918
StdCtrls.TMeasureItemEvent type 2918
StdCtrls.TMemo 2649
StdCtrls.TMemo.Align 2654
StdCtrls.TMemo.Alignment 2657
StdCtrls.TMemo.Anchors 2658
StdCtrls.TMemo.BevelEdges 2658
StdCtrls.TMemo.BevelInner 2659
StdCtrls.TMemo.BevelKind 2659
StdCtrls.TMemo.BevelOuter 2660
StdCtrls.TMemo.BiDiMode 2660
StdCtrls.TMemo.BorderStyle 2660
StdCtrls.TMemo.CharCase 2661
StdCtrls.TMemo.Color 2661
StdCtrls.TMemo.Constraints 2662
StdCtrls.TMemo.Ctl3D 2663
StdCtrls.TMemo.DoubleBuffered 2663
StdCtrls.TMemo.DragCursor 2665
StdCtrls.TMemo.DragKind 2665
StdCtrls.TMemo.DragMode 2666
StdCtrls.TMemo.Enabled 2667
StdCtrls.TMemo.Font 2668
StdCtrls.TMemo.HideSelection 2669
StdCtrls.TMemo.ImeMode 2669
StdCtrls.TMemo.ImeName 2670
StdCtrls.TMemo.Lines 2670
StdCtrls.TMemo.MaxLength 2671
StdCtrls.TMemo.OEMConvert 2672
StdCtrls.TMemo.OnChange 2690
StdCtrls.TMemo.OnClick 2690

StdCtrls.TMemo.OnContextPopup 2692
StdCtrls.TMemo.OnDblClick 2693
StdCtrls.TMemo.OnDragDrop 2693
StdCtrls.TMemo.OnDragOver 2696
StdCtrls.TMemo.OnEndDock 2699
StdCtrls.TMemo.OnEndDrag 2700
StdCtrls.TMemo.OnEnter 2701
StdCtrls.TMemo.OnExit 2703
StdCtrls.TMemo.OnKeyDown 2706
StdCtrls.TMemo.OnKeyPress 2710
StdCtrls.TMemo.OnKeyUp 2711
StdCtrls.TMemo.OnMouseActivate 2713
StdCtrls.TMemo.OnMouseDown 2714
StdCtrls.TMemo.OnMouseEnter 2716
StdCtrls.TMemo.OnMouseLeave 2716
StdCtrls.TMemo.OnMouseMove 2716
StdCtrls.TMemo.OnMouseUp 2719
StdCtrls.TMemo.OnStartDock 2721
StdCtrls.TMemo.OnStartDrag 2721
StdCtrls.TMemo.ParentBiDiMode 2672
StdCtrls.TMemo.ParentColor 2673
StdCtrls.TMemo.ParentCtl3D 2673
StdCtrls.TMemo.ParentDoubleBuffered 2673
StdCtrls.TMemo.ParentFont 2674
StdCtrls.TMemo.ParentShowHint 2674
StdCtrls.TMemo.PopupMenu 2675
StdCtrls.TMemo.ReadOnly 2680
StdCtrls.TMemo.ScrollBars 2680
StdCtrls.TMemo.ShowHint 2682
StdCtrls.TMemo.TabOrder 2684
StdCtrls.TMemo.TabStop 2684
StdCtrls.TMemo.Visible 2685
StdCtrls.TMemo.WantReturns 2686
StdCtrls.TMemo.WantTabs 2687
StdCtrls.TMemo.WordWrap 2687
StdCtrls.TOwnerDrawState 2918
StdCtrls.TOwnerDrawState type 2918
StdCtrls.TPushButtonActionLink 2722
StdCtrls.TPushButtonActionLink.IsImageIndexLinked 2723
StdCtrls.TPushButtonActionLink.SetImageIndex 2724
StdCtrls.TPushButtonActionLinkClass 2919
StdCtrls.TPushButtonActionLinkClass type 2919
StdCtrls.TRadioButton 2724
StdCtrls.TRadioButton.Action 2730
StdCtrls.TRadioButton.Align 2730
StdCtrls.TRadioButton.Alignment 2734
StdCtrls.TRadioButton.Anchors 2734
StdCtrls.TRadioButton.BiDiMode 2735
StdCtrls.TRadioButton.Caption 2735
StdCtrls.TRadioButton.Checked 2738
StdCtrls.TRadioButton.Color 2738
StdCtrls.TRadioButton.Constraints 2739
StdCtrls.TRadioButton.Create 2726
StdCtrls.TRadioButton.CreateParams 2727
StdCtrls.TRadioButton.CreateWnd 2728
StdCtrls.TRadioButton.Ctl3D 2740
StdCtrls.TRadioButton.DoubleBuffered 2740
StdCtrls.TRadioButton.DragCursor 2742
StdCtrls.TRadioButton.DragKind 2742
StdCtrls.TRadioButton.DragMode 2743
StdCtrls.TRadioButton.Enabled 2744
StdCtrls.TRadioButton.Font 2745
StdCtrls.TRadioButton.GetChecked 2728
StdCtrls.TRadioButton.GetControlsAlignment 2728
StdCtrls.TRadioButton.OnClick 2758
StdCtrls.TRadioButton.OnContextPopup 2760
StdCtrls.TRadioButton.OnDblClick 2761
StdCtrls.TRadioButton.OnDragDrop 2761
StdCtrls.TRadioButton.OnDragOver 2764
StdCtrls.TRadioButton.OnEndDock 2767
StdCtrls.TRadioButton.OnEndDrag 2768
StdCtrls.TRadioButton.OnEnter 2769
StdCtrls.TRadioButton.OnExit 2771
StdCtrls.TRadioButton.OnKeyDown 2774
StdCtrls.TRadioButton.OnKeyPress 2777
StdCtrls.TRadioButton.OnKeyUp 2779
StdCtrls.TRadioButton.OnMouseActivate 2780
StdCtrls.TRadioButton.OnMouseDown 2781

StdCtrls.TRadioButton.OnMouseEnter 2784
StdCtrls.TRadioButton.OnMouseLeave 2784
StdCtrls.TRadioButton.OnMouseMove 2784
StdCtrls.TRadioButton.OnMouseUp 2786
StdCtrls.TRadioButton.OnStartDock 2789
StdCtrls.TRadioButton.OnStartDrag 2789
StdCtrls.TRadioButton.ParentBiDiMode 2746
StdCtrls.TRadioButton.ParentColor 2746
StdCtrls.TRadioButton.ParentCtl3D 2747
StdCtrls.TRadioButton.ParentDoubleBuffered 2747
StdCtrls.TRadioButton.ParentFont 2747
StdCtrls.TRadioButton.ParentShowHint 2748
StdCtrls.TRadioButton.PopupMenu 2748
StdCtrls.TRadioButton.SetChecked 2729
StdCtrls.TRadioButton.ShowHint 2753
StdCtrls.TRadioButton.TabOrder 2755
StdCtrls.TRadioButton.TabStop 2755
StdCtrls.TRadioButton.Visible 2756
StdCtrls.TRadioButton.WordWrap 2757
StdCtrls.TScrollBar 2790
StdCtrls.TScrollBar.Align 2795
StdCtrls.TScrollBar.Anchors 2799
StdCtrls.TScrollBar.BiDiMode 2799
StdCtrls.TScrollBar.Change 2793
StdCtrls.TScrollBar.Constraints 2800
StdCtrls.TScrollBar.Create 2792
StdCtrls.TScrollBar.CreateParams 2793
StdCtrls.TScrollBar.CreateWnd 2793
StdCtrls.TScrollBar.Ctl3D 2800
StdCtrls.TScrollBar.DoubleBuffered 2801
StdCtrls.TScrollBar.DragCursor 2802
StdCtrls.TScrollBar.DragKind 2803
StdCtrls.TScrollBar.DragMode 2803
StdCtrls.TScrollBar.Enabled 2805
StdCtrls.TScrollBar.Kind 2805
StdCtrls.TScrollBar.LargeChange 2806
StdCtrls.TScrollBar.Max 2808
StdCtrls.TScrollBar.Min 2810
StdCtrls.TScrollBar.OnChange 2829
StdCtrls.TScrollBar.OnContextPopup 2829
StdCtrls.TScrollBar.OnDragDrop 2830
StdCtrls.TScrollBar.OnDragOver 2833
StdCtrls.TScrollBar.OnEndDock 2836
StdCtrls.TScrollBar.OnEndDrag 2836
StdCtrls.TScrollBar.OnEnter 2837
StdCtrls.TScrollBar.OnExit 2840
StdCtrls.TScrollBar.OnKeyDown 2842
StdCtrls.TScrollBar.OnKeyPress 2846
StdCtrls.TScrollBar.OnKeyUp 2847
StdCtrls.TScrollBar.OnMouseEnter 2849
StdCtrls.TScrollBar.OnMouseLeave 2849
StdCtrls.TScrollBar.OnScroll 2849
StdCtrls.TScrollBar.OnStartDock 2851
StdCtrls.TScrollBar.OnStartDrag 2852
StdCtrls.TScrollBar.PageSize 2812
StdCtrls.TScrollBar.ParentBiDiMode 2814
StdCtrls.TScrollBar.ParentCtl3D 2814
StdCtrls.TScrollBar.ParentDoubleBuffered 2814
StdCtrls.TScrollBar.ParentShowHint 2815
StdCtrls.TScrollBar.PopupMenu 2815
StdCtrls.TScrollBar.Position 2820
StdCtrls.TScrollBar.Scroll 2794
StdCtrls.TScrollBar.SetParams 2794
StdCtrls.TScrollBar.ShowHint 2822
StdCtrls.TScrollBar.SmallChange 2824
StdCtrls.TScrollBar.TabOrder 2826
StdCtrls.TScrollBar.TabStop 2826
StdCtrls.TScrollBar.Visible 2827
StdCtrls.TScrollCode 2911
StdCtrls.TScrollCode enumeration 2911
StdCtrls.TScrollEvent 2920
StdCtrls.TScrollEvent type 2920
StdCtrls.TScrollStyle 2912
StdCtrls.TScrollStyle enumeration 2912
StdCtrls.TSelection 2913
StdCtrls.TSelection record 2913
StdCtrls.TStaticBorderStyle 2913
StdCtrls.TStaticBorderStyle enumeration 2913

StdCtrls.TStaticText 2853
StdCtrls.TStaticText.Align 2855
StdCtrls.TStaticText.Alignment 2859
StdCtrls.TStaticText.Anchors 2859
StdCtrls.TStaticText.AutoSize 2860
StdCtrls.TStaticText.BevelEdges 2860
StdCtrls.TStaticText.BevelInner 2861
StdCtrls.TStaticText.BevelKind 2861
StdCtrls.TStaticText.BevelOuter 2862
StdCtrls.TStaticText.BiDiMode 2862
StdCtrls.TStaticText.BorderStyle 2862
StdCtrls.TStaticText.Caption 2863
StdCtrls.TStaticText.Color 2865
StdCtrls.TStaticText.Constraints 2866
StdCtrls.TStaticText.DoubleBuffered 2867
StdCtrls.TStaticText.DragCursor 2868
StdCtrls.TStaticText.DragKind 2869
StdCtrls.TStaticText.DragMode 2869
StdCtrls.TStaticText.Enabled 2871
StdCtrls.TStaticText.FocusControl 2871
StdCtrls.TStaticText.Font 2872
StdCtrls.TStaticText.OnClick 2885
StdCtrls.TStaticText.OnContextPopup 2887
StdCtrls.TStaticText.OnDblClick 2887
StdCtrls.TStaticText.OnDragDrop 2888
StdCtrls.TStaticText.OnDragOver 2891
StdCtrls.TStaticText.OnEndDock 2894
StdCtrls.TStaticText.OnEndDrag 2894
StdCtrls.TStaticText.OnMouseActivate 2895
StdCtrls.TStaticText.OnMouseDown 2896
StdCtrls.TStaticText.OnMouseEnter 2899
StdCtrls.TStaticText.OnMouseLeave 2899
StdCtrls.TStaticText.OnMouseMove 2899
StdCtrls.TStaticText.OnMouseUp 2901
StdCtrls.TStaticText.OnStartDock 2904
StdCtrls.TStaticText.OnStartDrag 2904
StdCtrls.TStaticText.ParentBiDiMode 2873
StdCtrls.TStaticText.ParentColor 2873
StdCtrls.TStaticText.ParentDoubleBuffered 2874
StdCtrls.TStaticText.ParentFont 2874
StdCtrls.TStaticText.ParentShowHint 2874
StdCtrls.TStaticText.PopupMenu 2875
StdCtrls.TStaticText.ShowAccelChar 2880
StdCtrls.TStaticText.ShowHint 2880
StdCtrls.TStaticText.TabOrder 2882
StdCtrls.TStaticText.TabStop 2882
StdCtrls.TStaticText.Transparent 2883
StdCtrls.TStaticText.Visible 2883
StdCtrls.TTextLayout 2914
StdCtrls.TTextLayout enumeration 2914
stDevice enumeration member 3049
stDgram enumeration member 1406
stDisabled enumeration member 3050
StdStyleActnCtrls 2921
StdStyleActnCtrls namespace 2921
Classes 2922
Variables 2925
StdStyleActnCtrls.StandardStyle 2925
StdStyleActnCtrls.StandardStyle variable 2925
StdStyleActnCtrls.TStandardStyleActionBars 2922
StdStyleActnCtrls.TStandardStyleActionBars.GetAddRemoveItemClass 2923
StdStyleActnCtrls.TStandardStyleActionBars.GetColorMapClass 2923
StdStyleActnCtrls.TStandardStyleActionBars.GetControlClass 2923
StdStyleActnCtrls.TStandardStyleActionBars.GetPopupClass 2924
StdStyleActnCtrls.TStandardStyleActionBars.GetScrollBtnClass 2924
StdStyleActnCtrls.TStandardStyleActionBars.GetStyleName 2924
StdVCL 2927
StdVCL namespace 2927
Constants 2944
Interfaces 2928
StdVCL.IDataBroker 2928
StdVCL.IDataBroker.GetProviderNames 2929
StdVCL.IDataBrokerDisp 2929

StdVCL.IDataBrokerDisp.GetProviderNames 2930
StdVCL.IID_IDataBroker 2944
StdVCL.IID_IDataBroker constant 2944
StdVCL.IID_IProvider 2944
StdVCL.IID_IProvider constant 2944
StdVCL.IID_IStrings 2945
StdVCL.IID_IStrings constant 2945
StdVCL.IProvider 2930
StdVCL.IProvider.ApplyUpdates 2931
StdVCL.IProvider.Constraints 2933
StdVCL.IProvider.Data 2933
StdVCL.IProvider.DataRequest 2931
StdVCL.IProvider.Get_Constraints 2931
StdVCL.IProvider.Get_Data 2932
StdVCL.IProvider.GetMetaData 2932
StdVCL.IProvider.GetRecords 2932
StdVCL.IProvider.Reset 2932
StdVCL.IProvider.Set_Constraints 2933
StdVCL.IProvider.SetParams 2933
StdVCL.IProviderDisp 2934
StdVCL.IProviderDisp.ApplyUpdates 2935
StdVCL.IProviderDisp.Constraints 2936
StdVCL.IProviderDisp.Data 2936
StdVCL.IProviderDisp.DataRequest 2935
StdVCL.IProviderDisp.GetMetaData 2935
StdVCL.IProviderDisp.GetRecords 2935
StdVCL.IProviderDisp.Reset 2936
StdVCL.IProviderDisp.SetParams 2936
StdVCL.IStrings 2937
StdVCL.IStrings._NewEnum 2938
StdVCL.IStrings.Add 2938
StdVCL.IStrings.Clear 2938
StdVCL.IStrings.ControlDefault 2940
StdVCL.IStrings.Count 2938
StdVCL.IStrings.Get_ControlDefault 2939
StdVCL.IStrings.Get_Item 2939
StdVCL.IStrings.Item 2940
StdVCL.IStrings.Remove 2939
StdVCL.IStrings.Set_ControlDefault 2939
StdVCL.IStrings.Set_Item 2940
StdVCL.IStringsDisp 2940
StdVCL.IStringsDisp._NewEnum 2941
StdVCL.IStringsDisp.Add 2942
StdVCL.IStringsDisp.Clear 2942
StdVCL.IStringsDisp.ControlDefault 2943
StdVCL.IStringsDisp.Count 2942
StdVCL.IStringsDisp.Item 2943
StdVCL.IStringsDisp.Remove 2942
StdVCL.LIBID_StdVCL 2945
StdVCL.LIBID_StdVCL constant 2945
stFileSystem enumeration member 3049
stIndexes enumeration member 1628
stManual enumeration member 3050
stNonBlocking enumeration member 137
stNoSchema enumeration member 1628
stPackages enumeration member 1628
stProcedureParams enumeration member 1628
stProcedures enumeration member 1628
stRaw enumeration member 1406
stRdm enumeration member 1406
StrHlpr 2947
StrHlpr namespace 2947
    Functions 2949
StrHlpr.AnsiAppend 2950
StrHlpr.AnsiAppend function 2950
StrHlpr.AnsiAssign 2950
StrHlpr.AnsiAssign function 2950
StrHlpr.AnsiCat 2950
StrHlpr.AnsiCat function 2950
StrHlpr.AnsiCopy 2951
StrHlpr.AnsiCopy function 2951
StrHlpr.AnsiDelete 2951
StrHlpr.AnsiDelete function 2951
StrHlpr.AnsiEqual 2951
StrHlpr.AnsiEqual function 2951
StrHlpr.AnsiFree 2952
StrHlpr.AnsiFree function 2952
StrHlpr.AnsiFromPChar 2952

StrHlpr.AnsiFromPChar function 2952
StrHlpr.AnsiFromPWideChar 2952
StrHlpr.AnsiFromPWideChar function 2952
StrHlpr.AnsiFromUnicode 2953
StrHlpr.AnsiFromUnicode function 2953
StrHlpr.AnsiFromWide 2953
StrHlpr.AnsiFromWide function 2953
StrHlpr.AnsiInsert 2953
StrHlpr.AnsiInsert function 2953
StrHlpr.AnsiPos 2954
StrHlpr.AnsiPos function 2954
StrHlpr.AnsiSetLength 2954
StrHlpr.AnsiSetLength function 2954
StrHlpr.UnicodeAppend 2954
StrHlpr.UnicodeAppend function 2954
StrHlpr.UnicodeAssign 2955
StrHlpr.UnicodeAssign function 2955
StrHlpr.UnicodeCat 2955
StrHlpr.UnicodeCat function 2955
StrHlpr.UnicodeCopy 2955
StrHlpr.UnicodeCopy function 2955
StrHlpr.UnicodeDelete 2956
StrHlpr.UnicodeDelete function 2956
StrHlpr.UnicodeEqual 2956
StrHlpr.UnicodeEqual function 2956
StrHlpr.UnicodeFree 2956
StrHlpr.UnicodeFree function 2956
StrHlpr.UnicodeFromAnsi 2957
StrHlpr.UnicodeFromAnsi function 2957
StrHlpr.UnicodeFromPChar 2957
StrHlpr.UnicodeFromPChar function 2957
StrHlpr.UnicodeFromPWideChar 2957
StrHlpr.UnicodeFromPWideChar function 2957
StrHlpr.UnicodeFromWide 2958
StrHlpr.UnicodeFromWide function 2958
StrHlpr.UnicodeGreater 2958
StrHlpr.UnicodeGreater function 2958
StrHlpr.UnicodeInsert 2958
StrHlpr.UnicodeInsert function 2958
StrHlpr.UnicodeLess 2959
StrHlpr.UnicodeLess function 2959
StrHlpr.UnicodePos 2959
StrHlpr.UnicodePos function 2959
StrHlpr.UnicodeSetLength 2959
StrHlpr.UnicodeSetLength function 2959
StrHlpr.WideAppend 2960
StrHlpr.WideAppend function 2960
StrHlpr.WideAssign 2960
StrHlpr.WideAssign function 2960
StrHlpr.WideCat 2960
StrHlpr.WideCat function 2960
StrHlpr.WideCopy 2961
StrHlpr.WideCopy function 2961
StrHlpr.WideDelete 2961
StrHlpr.WideDelete function 2961
StrHlpr.WideEqual 2961
StrHlpr.WideEqual function 2961
StrHlpr.WideFree 2962
StrHlpr.WideFree function 2962
StrHlpr.WideFromAnsi 2962
StrHlpr.WideFromAnsi function 2962
StrHlpr.WideFromPChar 2962
StrHlpr.WideFromPChar function 2962
StrHlpr.WideFromUnicode 2963
StrHlpr.WideFromUnicode function 2963
StrHlpr.WideGreater 2963
StrHlpr.WideGreater function 2963
StrHlpr.WideInsert 2963
StrHlpr.WideInsert function 2963
StrHlpr.WideLength 2964
StrHlpr.WideLength function 2964
StrHlpr.WideLess 2964
StrHlpr.WideLess function 2964
StrHlpr.WidePos 2964
StrHlpr.WidePos function 2964
StrHlpr.WideSetLength 2965
StrHlpr.WideSetLength function 2965
StrUtils 2967

StrUtils namespace 2967
Constants 3005
Functions 2970
Structs, Records, Enums 3000
Types 3001
Variables 3003
StrUtils.AnsiContainsStr 2971
StrUtils.AnsiContainsStr function 2971
StrUtils.AnsiContainsText 2972
StrUtils.AnsiContainsText function 2972
StrUtils.AnsiEndsStr 2972
StrUtils.AnsiEndsStr function 2972
StrUtils.AnsiEndsText 2973
StrUtils.AnsiEndsText function 2973
StrUtils.AnsiIndexStr 2973
StrUtils.AnsiIndexStr function 2973
StrUtils.AnsiIndexText 2974
StrUtils.AnsiIndexText function 2974
StrUtils.AnsiLeftStr 2974
StrUtils.AnsiLeftStr function 2974
StrUtils.AnsiMatchStr 2975
StrUtils.AnsiMatchStr function 2975
StrUtils.AnsiMatchText 2975
StrUtils.AnsiMatchText function 2975
StrUtils.AnsiMidStr 2976
StrUtils.AnsiMidStr function 2976
StrUtils.AnsiReplaceStr 2977
StrUtils.AnsiReplaceStr function 2977
StrUtils.AnsiReplaceText 2977
StrUtils.AnsiReplaceText function 2977
StrUtils.AnsiResemblesProc 3003
StrUtils.AnsiResemblesProc variable 3003
StrUtils.AnsiResemblesText 2978
StrUtils.AnsiResemblesText function 2978
StrUtils.AnsiReverseString 2978
StrUtils.AnsiReverseString function 2978
StrUtils.AnsiRightStr 2979
StrUtils.AnsiRightStr function 2979
StrUtils.AnsiStartsStr 2979
StrUtils.AnsiStartsStr function 2979
StrUtils.AnsiStartsText 2980
StrUtils.AnsiStartsText function 2980
StrUtils.ContainsStr 2980
StrUtils.ContainsStr function 2980
StrUtils.ContainsText 2981
StrUtils.ContainsText function 2981
StrUtils.DecodeSoundexInt 2981
StrUtils.DecodeSoundexInt function 2981
StrUtils.DecodeSoundexWord 2982
StrUtils.DecodeSoundexWord function 2982
StrUtils.DupeString 2982
StrUtils.DupeString function 2982
StrUtils.EndsStr 2983
StrUtils.EndsStr function 2983
StrUtils.EndsText 2983
StrUtils.EndsText function 2983
StrUtils.IfThen 2984
StrUtils.IfThen function 2984
StrUtils.IndexStr 2984
StrUtils.IndexStr function 2984
StrUtils.IndexText 2985
StrUtils.IndexText function 2985
StrUtils.LeftBStr 2985
StrUtils.LeftBStr function 2985
StrUtils.LeftStr 2986
StrUtils.LeftStr function 2986
StrUtils.MatchStr 2987
StrUtils.MatchStr function 2987
StrUtils.MatchText 2987
StrUtils.MatchText function 2987
StrUtils.MidBStr 2988
StrUtils.MidBStr function 2988
StrUtils.MidStr 2988
StrUtils.MidStr function 2988
StrUtils.PosEx 2989
StrUtils.PosEx function 2989
StrUtils.RandomFrom 2989
StrUtils.RandomFrom function 2989

StrUtils.ReplaceStr 2990
StrUtils.ReplaceStr function 2990
StrUtils.ReplaceText 2990
StrUtils.ReplaceText function 2990
StrUtils.ResemblesProc 3003
StrUtils.ResemblesProc variable 3003
StrUtils.ResemblesText 2991
StrUtils.ResemblesText function 2991
StrUtils.ReverseString 2991
StrUtils.ReverseString function 2991
StrUtils.RightBStr 2992
StrUtils.RightBStr function 2992
StrUtils.RightStr 2992
StrUtils.RightStr function 2992
StrUtils.SearchBuf 2993
StrUtils.SearchBuf function 2993
StrUtils.Soundex 2994
StrUtils.Soundex function 2994
StrUtils.SoundexCompare 2994
StrUtils.SoundexCompare function 2994
StrUtils.SoundexInt 2995
StrUtils.SoundexInt function 2995
StrUtils.SoundexProc 2996
StrUtils.SoundexProc function 2996
StrUtils.SoundexSimilar 2996
StrUtils.SoundexSimilar function 2996
StrUtils.SoundexWord 2997
StrUtils.SoundexWord function 2997
StrUtils.StartsStr 2998
StrUtils.StartsStr function 2998
StrUtils.StartsText 2998
StrUtils.StartsText function 2998
StrUtils.StuffString 2999
StrUtils.StuffString function 2999
StrUtils.TCompareTextProc 3001
StrUtils.TCompareTextProc type 3001
StrUtils.TSoundexIntLength 3001
StrUtils.TSoundexIntLength type 3001
StrUtils.TSoundexLength 3002
StrUtils.TSoundexLength type 3002
StrUtils.TStringSeachOption 3000
StrUtils.TStringSeachOption enumeration 3000
StrUtils.TStringSearchOptions 3002
StrUtils.TStringSearchOptions type 3002
StrUtils.WordDelimiters 3005
StrUtils.WordDelimiters constant 3005
stSeqPacket enumeration member 1406
stStream enumeration member 1406
stSysTables enumeration member 1628
stSystem enumeration member 3050
stTables enumeration member 1628
stThreadBlocking enumeration member 137
stUserNames enumeration member 1628
stWin32 enumeration member 3049
svApplication enumeration member 6686
SvcMgr 3007
SvcMgr namespace 3007
    Classes 3009
    Constants 3056
    Structs, Records, Enums 3048
    Types 3052
    Variables 3055
SvcMgr.Application 3055
SvcMgr.Application variable 3055
SvcMgr.CM_SERVICE_CONTROL_CODE 3056
SvcMgr.CM_SERVICE_CONTROL_CODE constant 3056
SvcMgr.TContinueEvent 3052
SvcMgr.TContinueEvent type 3052
SvcMgr.TCurrentStatus 3048
SvcMgr.TCurrentStatus enumeration 3048
SvcMgr.TDependencies 3009
SvcMgr.TDependencies.Create 3010
SvcMgr.TDependencies.GetOwner 3010
SvcMgr.TDependencies.Items 3011
SvcMgr.TDependency 3012
SvcMgr.TDependency.GetDisplayName 3013
SvcMgr.TDependency.IsGroup 3013
SvcMgr.TDependency.Name 3014

SvcMgr.TErrorSeverity 3049
SvcMgr.TErrorSeverity enumeration 3049
SvcMgr.TEventLogger 3014
SvcMgr.TEventLogger.Create 3015
SvcMgr.TEventLogger.Destroy 3016
SvcMgr.TEventLogger.LogMessage 3016
SvcMgr.TPauseEvent 3052
SvcMgr.TPauseEvent type 3052
SvcMgr.TService 3017
SvcMgr.TService.AfterInstall 3031
SvcMgr.TService.AfterUninstall 3032
SvcMgr.TService.AllowPause 3024
SvcMgr.TService.AllowStop 3024
SvcMgr.TService.BeforeInstall 3032
SvcMgr.TService.BeforeUninstall 3033
SvcMgr.TService.Controller 3020
SvcMgr.TService.CreateNew 3019
SvcMgr.TService.Dependencies 3024
SvcMgr.TService.Destroy 3019
SvcMgr.TService.DisplayName 3025
SvcMgr.TService.DoContinue 3020
SvcMgr.TService.DoCustomControl 3020
SvcMgr.TService.DoInterrogate 3020
SvcMgr.TService.DoPause 3021
SvcMgr.TService.DoShutdown 3021
SvcMgr.TService.DoStart 3021
SvcMgr.TService.DoStop 3021
SvcMgr.TService.ErrCode 3025
SvcMgr.TService.ErrorSeverity 3026
SvcMgr.TService.GetServiceController 3022
SvcMgr.TService.Interactive 3026
SvcMgr.TService.LoadGroup 3026
SvcMgr.TService.LogMessage 3022
SvcMgr.TService.Main 3023
SvcMgr.TService.OnContinue 3033
SvcMgr.TService.OnExecute 3034
SvcMgr.TService.OnPause 3034
SvcMgr.TService.OnShutdown 3034
SvcMgr.TService.OnStart 3035
SvcMgr.TService.OnStop 3035
SvcMgr.TService.Param 3027
SvcMgr.TService.ParamCount 3027
SvcMgr.TService.Password 3027
SvcMgr.TService.ReportStatus 3023
SvcMgr.TService.ServiceStartName 3028
SvcMgr.TService.ServiceThread 3028
SvcMgr.TService.ServiceType 3029
SvcMgr.TService.StartType 3029
SvcMgr.TService.Status 3029
SvcMgr.TService.TagID 3030
SvcMgr.TService.Terminated 3030
SvcMgr.TService.WaitHint 3030
SvcMgr.TService.Win32ErrCode 3031
SvcMgr.TServiceApplication 3035
SvcMgr.TServiceApplication.Create 3037
SvcMgr.TServiceApplication.CreateForm 3038
SvcMgr.TServiceApplication.DelayInitialize 3040
SvcMgr.TServiceApplication.Destroy 3037
SvcMgr.TServiceApplication.DispatchServiceMain 3038
SvcMgr.TServiceApplication.DoHandleException 3038
SvcMgr.TServiceApplication.Hook 3039
SvcMgr.TServiceApplication.Initialize 3039
SvcMgr.TServiceApplication.Installing 3039
SvcMgr.TServiceApplication.RegisterServices 3039
SvcMgr.TServiceApplication.Run 3040
SvcMgr.TServiceApplication.ServiceCount 3040
SvcMgr.TServiceApplication.Title 3041
SvcMgr.TServiceController 3053
SvcMgr.TServiceController type 3053
SvcMgr.TServiceEvent 3053
SvcMgr.TServiceEvent type 3053
SvcMgr.TServiceThread 3041
SvcMgr.TServiceThread.Create 3042
SvcMgr.TServiceThread.Execute 3043
SvcMgr.TServiceThread.ProcessRequests 3046
SvcMgr.TServiceType 3049
SvcMgr.TServiceType enumeration 3049
SvcMgr.TStartEvent 3053

SvcMgr.TStartEvent type 3053
SvcMgr.TStartType 3050
SvcMgr.TStartType enumeration 3050
SvcMgr.TStopEvent 3054
SvcMgr.TStopEvent type 3054
svEndUser enumeration member 6686
svHTMLItems enumeration member 6686
svModules enumeration member 6686
svPage enumeration member 6686
svPages enumeration member 6686
svProducer enumeration member 6686
svRequest enumeration member 6686
svResponse enumeration member 6686
svSession enumeration member 6686
SyncObjs 3057
SyncObjs namespace 3057
Classes 3058
Structs, Records, Enums 3094
Types 3095
SyncObjs.TConditionVariableCS 3058
SyncObjs.TConditionVariableCS.Acquire 3060
SyncObjs.TConditionVariableCS.Create 3059
SyncObjs.TConditionVariableCS.FConditionVariable 3060
SyncObjs.TConditionVariableCS.Release 3061
SyncObjs.TConditionVariableCS.ReleaseAll 3061
SyncObjs.TConditionVariableCS.WaitFor 3062
SyncObjs.TConditionVariableHelper 3062
SyncObjs.TConditionVariableHelper.Create 3063
SyncObjs.TConditionVariableHelper.Free 3063
SyncObjs.TConditionVariableHelper.SleepCS 3063
SyncObjs.TConditionVariableHelper.Wake 3064
SyncObjs.TConditionVariableHelper.WakeAll 3064
SyncObjs.TConditionVariableMutex 3064
SyncObjs.TConditionVariableMutex.Acquire 3066
SyncObjs.TConditionVariableMutex.Create 3065
SyncObjs.TConditionVariableMutex.Destroy 3066
SyncObjs.TConditionVariableMutex.Release 3067
SyncObjs.TConditionVariableMutex.ReleaseAll 3067
SyncObjs.TConditionVariableMutex.WaitFor 3067
SyncObjs.TCriticalSection 3068
SyncObjs.TCriticalSection.Acquire 3070
SyncObjs.TCriticalSection.Create 3069
SyncObjs.TCriticalSection.Destroy 3069
SyncObjs.TCriticalSection.Enter 3071
SyncObjs.TCriticalSection.FSection 3070
SyncObjs.TCriticalSection.Leave 3071
SyncObjs.TCriticalSection.Release 3072
SyncObjs.TCriticalSection.TryEnter 3072
SyncObjs.TCriticalSectionHelper 3072
SyncObjs.TCriticalSectionHelper.Destroy 3073
SyncObjs.TCriticalSectionHelper.Enter 3073
SyncObjs.TCriticalSectionHelper.Free 3074
SyncObjs.TCriticalSectionHelper.Initialize 3074
SyncObjs.TCriticalSectionHelper.Leave 3074
SyncObjs.TCriticalSectionHelper.TryEnter 3074
SyncObjs.TEvent 3074
SyncObjs.TEvent.Create 3076
SyncObjs.TEvent.ResetEvent 3077
SyncObjs.TEvent.SetEvent 3078
SyncObjs.THandleObject 3078
SyncObjs.THandleObject.Create 3080
SyncObjs.THandleObject.Destroy 3080
SyncObjs.THandleObject.FHandle 3081
SyncObjs.THandleObject.FLastError 3081
SyncObjs.THandleObject.FUseCOMWait 3081
SyncObjs.THandleObject.Handle 3082
SyncObjs.THandleObject.LastError 3083
SyncObjs.THandleObject.WaitFor 3082
SyncObjs.THandleObject.WaitForMultiple 3082
SyncObjs.THandleObjectArray 3095
SyncObjs.THandleObjectArray type 3095
SyncObjs.TMutex 3083
SyncObjs.TMutex.Acquire 3086
SyncObjs.TMutex.Create 3085
SyncObjs.TMutex.Release 3086
SyncObjs.TSemaphore 3086
SyncObjs.TSemaphore.Acquire 3089
SyncObjs.TSemaphore.Create 3088, 3089

SyncObjs.TSemaphore.Release 3090
SyncObjs.TSimpleEvent 3090
SyncObjs.TSynchroObject 3091
SyncObjs.TSynchroObject.Acquire 3092
SyncObjs.TSynchroObject.Release 3093
SyncObjs.TWaitResult 3094
SyncObjs.TWaitResult enumeration 3094
SysConst 3097
SysConst namespace 3097
Constants 3101
SysConst.SAbstractError 3104
SysConst.SAbstractError constant 3104
SysConst.SAccessDenied 3104
SysConst.SAccessDenied constant 3104
SysConst.SAccessViolationArg3 3104
SysConst.SAccessViolationArg3 constant 3104
SysConst.SAccessViolationNoArg 3105
SysConst.SAccessViolationNoArg constant 3105
SysConst.SArgumentMissing 3105
SysConst.SArgumentMissing constant 3105
SysConst.SAssertError 3105
SysConst.SAssertError constant 3105
SysConst.SAssertionFailed 3106
SysConst.SAssertionFailed constant 3106
SysConst.SByteIndexOutOfBounds 3106
SysConst.SByteIndexOutOfBounds constant 3106
SysConst.SCannotCreateDir 3106
SysConst.SCannotCreateDir constant 3106
SysConst.SCannotReadPackageInfo 3107
SysConst.SCannotReadPackageInfo constant 3107
SysConst.SCharIndexOutOfBounds 3107
SysConst.SCharIndexOutOfBounds constant 3107
SysConst.SCodesetConversionError 3107
SysConst.SCodesetConversionError constant 3107
SysConst.SControlC 3108
SysConst.SControlC constant 3108
SysConst.SDateEncodeError 3108
SysConst.SDateEncodeError constant 3108
SysConst.SDiskFull 3108
SysConst.SDiskFull constant 3108
SysConst.SDispatchError 3109
SysConst.SDispatchError constant 3109
SysConst.SDivByZero 3109
SysConst.SDivByZero constant 3109
SysConst.SDuplicatePackageUnit 3109
SysConst.SDuplicatePackageUnit constant 3109
SysConst.SEndOfFile 3110
SysConst.SEndOfFile constant 3110
SysConst.sErrorLoadingPackage 3110
SysConst.sErrorLoadingPackage constant 3110
SysConst.SException 3110
SysConst.SException constant 3110
SysConst.SExceptTitle 3111
SysConst.SExceptTitle constant 3111
SysConst.SExternalException 3111
SysConst.SExternalException constant 3111
SysConst.SFileNotFound 3111
SysConst.SFileNotFound constant 3111
SysConst.SFormatTooLong 3112
SysConst.SFormatTooLong constant 3112
SysConst.SInOutError 3112
SysConst.SInOutError constant 3112
SysConst.SIntfCastError 3112
SysConst.SIntfCastError constant 3112
SysConst.SIntOverflow 3113
SysConst.SIntOverflow constant 3113
SysConst.SInvalidBoolean 3113
SysConst.SInvalidBoolean constant 3113
SysConst.SInvalidCast 3113
SysConst.SInvalidCast constant 3113
SysConst.SInvalidCharCount 3114
SysConst.SInvalidCharCount constant 3114
SysConst.SInvalidCodePage 3114
SysConst.SInvalidCodePage constant 3114
SysConst.SInvalidCurrency 3114
SysConst.SInvalidCurrency constant 3114
SysConst.SInvalidDate 3115
SysConst.SInvalidDate constant 3115

SysConst.SInvalidDateTime 3115
SysConst.SInvalidDateTime constant 3115
SysConst.SInvalidDateTimeFloat 3115
SysConst.SInvalidDateTimeFloat constant 3115
SysConst.SInvalidDestinationArray 3116
SysConst.SInvalidDestinationArray constant 3116
SysConst.SInvalidDestinationIndex 3116
SysConst.SInvalidDestinationIndex constant 3116
SysConst.SInvalidFilename 3116
SysConst.SInvalidFilename constant 3116
SysConst.SInvalidFloat 3117
SysConst.SInvalidFloat constant 3117
SysConst.SInvalidFormat 3117
SysConst.SInvalidFormat constant 3117
SysConst.SInvalidGUID 3117
SysConst.SInvalidGUID constant 3117
SysConst.SInvalidInput 3118
SysConst.SInvalidInput constant 3118
SysConst.SInvalidInteger 3118
SysConst.SInvalidInteger constant 3118
SysConst.SInvalidOp 3118
SysConst.SInvalidOp constant 3118
SysConst.SInvalidPackageFile 3119
SysConst.SInvalidPackageFile constant 3119
SysConst.SInvalidPackageHandle 3119
SysConst.SInvalidPackageHandle constant 3119
SysConst.SInvalidPointer 3119
SysConst.SInvalidPointer constant 3119
SysConst.SInvalidSourceArray 3120
SysConst.SInvalidSourceArray constant 3120
SysConst.SInvalidTime 3120
SysConst.SInvalidTime constant 3120
SysConst.SInvalidTimeStamp 3120
SysConst.SInvalidTimeStamp constant 3120
SysConst.SInvalidVarCast 3121
SysConst.SInvalidVarCast constant 3121
SysConst.SInvalidVarNullOp 3121
SysConst.SInvalidVarNullOp constant 3121
SysConst.SInvalidVarOp 3121
SysConst.SInvalidVarOp constant 3121
SysConst.SInvalidVarOpWithHResult 3122
SysConst.SInvalidVarOpWithHResult constant 3122
SysConst.SInvalidVarOpWithHResultWithPrefix 3122
SysConst.SInvalidVarOpWithHResultWithPrefix constant 3122
SysConst.SLongDayNameFri 3122
SysConst.SLongDayNameFri constant 3122
SysConst.SLongDayNameMon 3123
SysConst.SLongDayNameMon constant 3123
SysConst.SLongDayNameSat 3123
SysConst.SLongDayNameSat constant 3123
SysConst.SLongDayNameSun 3123
SysConst.SLongDayNameSun constant 3123
SysConst.SLongDayNameThu 3124
SysConst.SLongDayNameThu constant 3124
SysConst.SLongDayNameTue 3124
SysConst.SLongDayNameTue constant 3124
SysConst.SLongDayNameWed 3124
SysConst.SLongDayNameWed constant 3124
SysConst.SLongMonthNameApr 3125
SysConst.SLongMonthNameApr constant 3125
SysConst.SLongMonthNameAug 3125
SysConst.SLongMonthNameAug constant 3125
SysConst.SLongMonthNameDec 3125
SysConst.SLongMonthNameDec constant 3125
SysConst.SLongMonthNameFeb 3126
SysConst.SLongMonthNameFeb constant 3126
SysConst.SLongMonthNameJan 3126
SysConst.SLongMonthNameJan constant 3126
SysConst.SLongMonthNameJul 3126
SysConst.SLongMonthNameJul constant 3126
SysConst.SLongMonthNameJun 3127
SysConst.SLongMonthNameJun constant 3127
SysConst.SLongMonthNameMar 3127
SysConst.SLongMonthNameMar constant 3127
SysConst.SLongMonthNameMay 3127
SysConst.SLongMonthNameMay constant 3127
SysConst.SLongMonthNameNov 3128
SysConst.SLongMonthNameNov constant 3128

SysConst.SLongMonthNameOct 3128
SysConst.SLongMonthNameOct constant 3128
SysConst.SLongMonthNameSep 3128
SysConst.SLongMonthNameSep constant 3128
SysConst.SModuleAccessViolation 3129
SysConst.SModuleAccessViolation constant 3129
SysConst.SMonitorLockException 3129
SysConst.SMonitorLockException constant 3129
SysConst.SNL 3129
SysConst.SNL constant 3129
SysConst.SNoMonitorSupportException 3130
SysConst.SNoMonitorSupportException constant 3130
SysConst.SOperationAborted 3130
SysConst.SOperationAborted constant 3130
SysConst.SOSError 3130
SysConst.SOSError constant 3130
SysConst.SOutOfMemory 3131
SysConst.SOutOfMemory constant 3131
SysConst.SOverflow 3131
SysConst.SOverflow constant 3131
SysConst.SPrivilege 3131
SysConst.SPrivilege constant 3131
SysConst.SQuit 3132
SysConst.SQuit constant 3132
SysConst.SRangeError 3132
SysConst.SRangeError constant 3132
SysConst.SReadAccess 3132
SysConst.SReadAccess constant 3132
SysConst.SResultTooLong 3133
SysConst.SResultTooLong constant 3133
SysConst.SSafecallException 3133
SysConst.SSafecallException constant 3133
SysConst.SShortDayNameFri 3133
SysConst.SShortDayNameFri constant 3133
SysConst.SShortDayNameMon 3134
SysConst.SShortDayNameMon constant 3134
SysConst.SShortDayNameSat 3134
SysConst.SShortDayNameSat constant 3134
SysConst.SShortDayNameSun 3134
SysConst.SShortDayNameSun constant 3134
SysConst.SShortDayNameThu 3135
SysConst.SShortDayNameThu constant 3135
SysConst.SShortDayNameTue 3135
SysConst.SShortDayNameTue constant 3135
SysConst.SShortDayNameWed 3135
SysConst.SShortDayNameWed constant 3135
SysConst.SShortMonthNameApr 3136
SysConst.SShortMonthNameApr constant 3136
SysConst.SShortMonthNameAug 3136
SysConst.SShortMonthNameAug constant 3136
SysConst.SShortMonthNameDec 3136
SysConst.SShortMonthNameDec constant 3136
SysConst.SShortMonthNameFeb 3137
SysConst.SShortMonthNameFeb constant 3137
SysConst.SShortMonthNameJan 3137
SysConst.SShortMonthNameJan constant 3137
SysConst.SShortMonthNameJul 3137
SysConst.SShortMonthNameJul constant 3137
SysConst.SShortMonthNameJun 3138
SysConst.SShortMonthNameJun constant 3138
SysConst.SShortMonthNameMar 3138
SysConst.SShortMonthNameMar constant 3138
SysConst.SShortMonthNameMay 3138
SysConst.SShortMonthNameMay constant 3138
SysConst.SShortMonthNameNov 3139
SysConst.SShortMonthNameNov constant 3139
SysConst.SShortMonthNameOct 3139
SysConst.SShortMonthNameOct constant 3139
SysConst.SShortMonthNameSep 3139
SysConst.SShortMonthNameSep constant 3139
SysConst.SStackOverflow 3140
SysConst.SStackOverflow constant 3140
SysConst.STimeEncodeError 3140
SysConst.STimeEncodeError constant 3140
SysConst.STooManyOpenFiles 3140
SysConst.STooManyOpenFiles constant 3140
SysConst.SUnderflow 3141
SysConst.SUnderflow constant 3141

SysConst.SUnknown 3141
SysConst.SUnknown constant 3141
SysConst.sUnknownFromType 3141
SysConst.sUnknownFromType constant 3141
SysConst.sUnknownToType 3142
SysConst.sUnknownToType constant 3142
SysConst.SUnkOSError 3142
SysConst.SUnkOSError constant 3142
SysConst.SUnkWin32Error 3142
SysConst.SUnkWin32Error constant 3142
SysConst.SVarArrayBounds 3143
SysConst.SVarArrayBounds constant 3143
SysConst.SVarArrayCreate 3143
SysConst.SVarArrayCreate constant 3143
SysConst.SVarArrayLocked 3143
SysConst.SVarArrayLocked constant 3143
SysConst.SVarArrayWithHResult 3144
SysConst.SVarArrayWithHResult constant 3144
SysConst.SVarBadType 3144
SysConst.SVarBadType constant 3144
SysConst.SVarDataCastToRecursing 3144
SysConst.SVarDataCastToRecursing constant 3144
SysConst.SVarDataClearRecursing 3145
SysConst.SVarDataClearRecursing constant 3145
SysConst.SVarDataCopyNoIndRecursing 3145
SysConst.SVarDataCopyNoIndRecursing constant 3145
SysConst.SVarDataCopyRecursing 3145
SysConst.SVarDataCopyRecursing constant 3145
SysConst.SVarDataInitRecursing 3146
SysConst.SVarDataInitRecursing constant 3146
SysConst.SVarInvalid 3146
SysConst.SVarInvalid constant 3146
SysConst.SVarIsEmpty 3146
SysConst.SVarIsEmpty constant 3146
SysConst.SVarNotArray 3147
SysConst.SVarNotArray constant 3147
SysConst.SVarNotImplemented 3147
SysConst.SVarNotImplemented constant 3147
SysConst.SVarOutOfMemory 3147
SysConst.SVarOutOfMemory constant 3147
SysConst.SVarOverflow 3148
SysConst.SVarOverflow constant 3148
SysConst.SVarTypeAlreadyUsed 3148
SysConst.SVarTypeAlreadyUsed constant 3148
SysConst.SVarTypeAlreadyUsedWithPrefix 3148
SysConst.SVarTypeAlreadyUsedWithPrefix constant 3148
SysConst.SVarTypeConvertOverflow 3149
SysConst.SVarTypeConvertOverflow constant 3149
SysConst.SVarTypeCouldNotConvert 3149
SysConst.SVarTypeCouldNotConvert constant 3149
SysConst.SVarTypeNotUsable 3149
SysConst.SVarTypeNotUsable constant 3149
SysConst.SVarTypeNotUsableWithPrefix 3150
SysConst.SVarTypeNotUsableWithPrefix constant 3150
SysConst.SVarTypeOutOfRange 3150
SysConst.SVarTypeOutOfRange constant 3150
SysConst.SVarTypeOutOfRangeWithPrefix 3150
SysConst.SVarTypeOutOfRangeWithPrefix constant 3150
SysConst.SVarTypeRangeCheck1 3151
SysConst.SVarTypeRangeCheck1 constant 3151
SysConst.SVarTypeRangeCheck2 3151
SysConst.SVarTypeRangeCheck2 constant 3151
SysConst.SVarTypeTooManyCustom 3151
SysConst.SVarTypeTooManyCustom constant 3151
SysConst.SVarTypeUnknown 3152
SysConst.SVarTypeUnknown constant 3152
SysConst.SVarUnexpected 3152
SysConst.SVarUnexpected constant 3152
SysConst.SWin32Error 3152
SysConst.SWin32Error constant 3152
SysConst.SWriteAccess 3153
SysConst.SWriteAccess constant 3153
SysConst.SZeroDivide 3153
SysConst.SZeroDivide constant 3153
SysInit 3155
SysInit namespace 3155
    Constants 3163
    Functions 3156

Variables 3159
SysInit._GetTls 3156
SysInit._GetTls function 3156
SysInit._InitExe 3156
SysInit._InitExe function 3156
SysInit._InitLib 3157
SysInit._InitLib function 3157
SysInit._InitPkg 3157
SysInit._InitPkg function 3157
SysInit._PackageLoad 3157
SysInit._PackageLoad function 3157
SysInit._PackageUnload 3158
SysInit._PackageUnload function 3158
SysInit.CoverageLibraryName 3159
SysInit.CoverageLibraryName variable 3159
SysInit.DataMark 3159
SysInit.DataMark variable 3159
SysInit.DllProc 3160
SysInit.DllProc variable 3160
SysInit.DllProcEx 3160
SysInit.DllProcEx variable 3160
SysInit.HInstance 3160
SysInit.HInstance variable 3160
SysInit.ModuleIsCpp 3161
SysInit.ModuleIsCpp variable 3161
SysInit.ModuleIsLib 3161
SysInit.ModuleIsLib variable 3161
SysInit.ModuleIsPackage 3161
SysInit.ModuleIsPackage variable 3161
SysInit.PtrToNil 3163
SysInit.PtrToNil constant 3163
SysInit.TlsIndex 3162
SysInit.TlsIndex variable 3162
SysInit.TlsLast 3162
SysInit.TlsLast variable 3162
SysInit.VclExit 3158
SysInit.VclExit function 3158
SysInit.VclInit 3158
SysInit.VclInit function 3158
System 3165
System namespace 3165
Classes 3185
Constants 3909
Functions 3507
Interfaces 3497
Macros 3960
Structs, Records, Enums 3794
Types 3822
Variables 3874
System.__ArrayToPasStr 3518
System.__ArrayToPasStr function 3518
System.__CLenToPasStr 3518
System.__CLenToPasStr function 3518
System.__CToPasStr 3518
System.__CToPasStr function 3518
System.__IOTest 3519
System.__IOTest function 3519
System.__lldiv 3519
System.__lldiv function 3519
System.__lldivo 3520
System.__lldivo function 3520
System.__llmod 3520
System.__llmod function 3520
System.__llmodo 3520
System.__llmodo function 3520
System.__llmul 3521
System.__llmul function 3521
System.__llmulo 3521
System.__llmulo function 3521
System.__llshl 3521
System.__llshl function 3521
System.__lludiv 3522
System.__lludiv function 3522
System.__llumod 3522
System.__llumod function 3522
System.__llushr 3522
System.__llushr function 3522
System.__PasToCStr 3523

System.__PasToCStr function 3523
System._AbstractError 3523
System._AbstractError function 3523
System._AddRef 3523
System._AddRef function 3523
System._AddRefArray 3524
System._AddRefArray function 3524
System._AddRefRecord 3524
System._AddRefRecord function 3524
System._AfterConstruction 3524
System._AfterConstruction function 3524
System._Append 3525
System._Append function 3525
System._AsClass 3525
System._AsClass function 3525
System._Assert 3525
System._Assert function 3525
System._Assign 3526
System._Assign function 3526
System._AStrCmp 3526
System._AStrCmp function 3526
System._BeforeDestruction 3526
System._BeforeDestruction function 3526
System._BlockRead 3527
System._BlockRead function 3527
System._BlockWrite 3527
System._BlockWrite function 3527
System._BoundErr 3527
System._BoundErr function 3527
System._CallDynaClass 3528
System._CallDynaClass function 3528
System._CallDynaInst 3528
System._CallDynaInst function 3528
System._CheckAutoResult 3528
System._CheckAutoResult function 3528
System._ClassCreate 3529
System._ClassCreate function 3529
System._ClassDestroy 3529
System._ClassDestroy function 3529
System._Close 3529
System._Close function 3529
System._Copy 3530
System._Copy function 3530
System._CopyArray 3530
System._CopyArray function 3530
System._CopyObject 3530
System._CopyObject function 3530
System._CopyRecord 3531
System._CopyRecord function 3531
System._CVR_PROBE 3531
System._CVR_PROBE function 3531
System._CVR_STMTPROBE 3531
System._CVR_STMTPROBE function 3531
System._Delete 3532
System._Delete function 3532
System._Dispose 3532
System._Dispose function 3532
System._DoneExcept 3532
System._DoneExcept function 3532
System._DynArrayAddRef 3533
System._DynArrayAddRef function 3533
System._DynArrayAsg 3533
System._DynArrayAsg function 3533
System._DynArrayClear 3533
System._DynArrayClear function 3533
System._DynArrayCopy 3534
System._DynArrayCopy function 3534
System._DynArrayCopyRange 3534
System._DynArrayCopyRange function 3534
System._DynArrayHigh 3534
System._DynArrayHigh function 3534
System._DynArrayLength 3535
System._DynArrayLength function 3535
System._DynArraySetLength 3535
System._DynArraySetLength function 3535
System._EnsureAnsiString 3535
System._EnsureAnsiString function 3535
System._EnsureUnicodeString 3536

System._EnsureUnicodeString function 3536
System._EofFile 3536
System._EofFile function 3536
System._EofText 3536
System._EofText function 3536
System._Eoln 3537
System._Eoln function 3537
System._Erase 3537
System._Erase function 3537
System._Ext2Real 3537
System._Ext2Real function 3537
System._Fail 3538
System._Fail function 3538
System._FilePos 3538
System._FilePos function 3538
System._FileSize 3538
System._FileSize function 3538
System._FillChar 3539
System._FillChar function 3539
System._Finalize 3539
System._Finalize function 3539
System._FinalizeArray 3539
System._FinalizeArray function 3539
System._FinalizeRecord 3540
System._FinalizeRecord function 3540
System._FindDynaClass 3540
System._FindDynaClass function 3540
System._FindDynaInst 3540
System._FindDynaInst function 3540
System._Flush 3541
System._Flush function 3541
System._FreeMem 3541
System._FreeMem function 3541
System._FSafeDivide 3541
System._FSafeDivide function 3541
System._FSafeDivideR 3542
System._FSafeDivideR function 3542
System._GetHelperIntf 3542
System._GetHelperIntf function 3542
System._GetMem 3542
System._GetMem function 3542
System._Halt 3543
System._Halt function 3543
System._Halt0 3543
System._Halt0 function 3543
System._HandleAnyException 3543
System._HandleAnyException function 3543
System._HandleAutoException 3544
System._HandleAutoException function 3544
System._HandleFinally 3544
System._HandleFinally function 3544
System._HandleOnException 3544
System._HandleOnException function 3544
System._Initialize 3545
System._Initialize function 3545
System._InitializeArray 3545
System._InitializeArray function 3545
System._InitializeRecord 3545
System._InitializeRecord function 3545
System._InitImports 3546
System._InitImports function 3546
System._InitResStringImports 3546
System._InitResStringImports function 3546
System._InitResStrings 3546
System._InitResStrings function 3546
System._InitUnicodeStrings 3547
System._InitUnicodeStrings function 3547
System._InitWideStrings 3547
System._InitWideStrings function 3547
System._Insert 3547
System._Insert function 3547
System._InternalLStrFromUStr 3548
System._InternalLStrFromUStr function 3548
System._InternalUStrFromLStr 3548
System._InternalUStrFromLStr function 3548
System._IntfAddRef 3548
System._IntfAddRef function 3548
System._IntfCast 3549

System._IntfCast function 3549
System._IntfClear 3549
System._IntfClear function 3549
System._IntfCopy 3549
System._IntfCopy function 3549
System._IntfDispCall 3550
System._IntfDispCall function 3550
System._IntfVarCall 3550
System._IntfVarCall function 3550
System._IntOver 3550
System._IntOver function 3550
System._IsClass 3551
System._IsClass function 3551
System._LGetDir 3551
System._LGetDir function 3551
System._LStrAddRef 3551
System._LStrAddRef function 3551
System._LStrArrayClr 3552
System._LStrArrayClr function 3552
System._LStrAsg 3552
System._LStrAsg function 3552
System._LStrCat 3552
System._LStrCat function 3552
System._LStrCat3 3553
System._LStrCat3 function 3553
System._LStrCatN 3553
System._LStrCatN function 3553
System._LStrClr 3553
System._LStrClr function 3553
System._LStrCmp 3554
System._LStrCmp function 3554
System._LStrCopy 3554
System._LStrCopy function 3554
System._LStrDelete 3554
System._LStrDelete function 3554
System._LStrEqual 3555
System._LStrEqual function 3555
System._LStrFromArray 3555
System._LStrFromArray function 3555
System._LStrFromChar 3555
System._LStrFromChar function 3555
System._LStrFromPChar 3556
System._LStrFromPChar function 3556
System._LStrFromPCharLen 3556
System._LStrFromPCharLen function 3556
System._LStrFromPWChar 3556
System._LStrFromPWChar function 3556
System._LStrFromPWCharLen 3557
System._LStrFromPWCharLen function 3557
System._LStrFromString 3557
System._LStrFromString function 3557
System._LStrFromUStr 3557
System._LStrFromUStr function 3557
System._LStrFromWArray 3558
System._LStrFromWArray function 3558
System._LStrFromWChar 3558
System._LStrFromWChar function 3558
System._LStrFromWStr 3558
System._LStrFromWStr function 3558
System._LStrInsert 3559
System._LStrInsert function 3559
System._LStrLAsg 3559
System._LStrLAsg function 3559
System._LStrLen 3559
System._LStrLen function 3559
System._LStrPos 3560
System._LStrPos function 3560
System._LStrSetLength 3560
System._LStrSetLength function 3560
System._LStrToPChar 3560
System._LStrToPChar function 3560
System._LStrToString 3561
System._LStrToString function 3561
System._New 3561
System._New function 3561
System._NewAnsiString 3561
System._NewAnsiString function 3561
System._NewUnicodeString 3562

System._NewUnicodeString function 3562
System._NewWideString 3562
System._NewWideString function 3562
System._ObjCopy 3562
System._ObjCopy function 3562
System._ObjSetup 3563
System._ObjSetup function 3563
System._PackageLoad 3563
System._PackageLoad function 3563
System._PackageUnload 3563
System._PackageUnload function 3563
System._PCharLen 3564
System._PCharLen function 3564
System._Pos 3564
System._Pos function 3564
System._Pow10 3564
System._Pow10 function 3564
System._PStrCat 3565
System._PStrCat function 3565
System._PStrCmp 3565
System._PStrCmp function 3565
System._PStrCpy 3565
System._PStrCpy function 3565
System._PStrNCat 3566
System._PStrNCat function 3566
System._PStrNCpy 3566
System._PStrNCpy function 3566
System._PWCharLen 3566
System._PWCharLen function 3566
System._RaiseAgain 3567
System._RaiseAgain function 3567
System._RaiseExcept 3567
System._RaiseExcept function 3567
System._ReadChar 3567
System._ReadChar function 3567
System._ReadCString 3568
System._ReadCString function 3568
System._ReadExt 3568
System._ReadExt function 3568
System._ReadInt64 3568
System._ReadInt64 function 3568
System._ReadLn 3569
System._ReadLn function 3569
System._ReadLong 3569
System._ReadLong function 3569
System._ReadLString 3569
System._ReadLString function 3569
System._ReadRec 3570
System._ReadRec function 3570
System._ReadString 3570
System._ReadString function 3570
System._ReadUString 3570
System._ReadUString function 3570
System._ReadWChar 3571
System._ReadWChar function 3571
System._ReadWCString 3571
System._ReadWCString function 3571
System._ReadWString 3571
System._ReadWString function 3571
System._Real2Ext 3572
System._Real2Ext function 3572
System._ReallocMem 3572
System._ReallocMem function 3572
System._Rename 3572
System._Rename function 3572
System._ResetFile 3573
System._ResetFile function 3573
System._ResetText 3573
System._ResetText function 3573
System._RewritFile 3573
System._RewritFile function 3573
System._RewritText 3574
System._RewritText function 3574
System._ROUND 3574
System._ROUND function 3574
System._Run0Error 3575
System._Run0Error function 3575
System._RunError 3575

System._RunError function 3575
System._Seek 3575
System._Seek function 3575
System._SeekEof 3576
System._SeekEof function 3576
System._SeekEoln 3576
System._SeekEoln function 3576
System._SetElem 3576
System._SetElem function 3576
System._SetEq 3577
System._SetEq function 3577
System._SetExpand 3577
System._SetExpand function 3577
System._SetIntersect 3577
System._SetIntersect function 3577
System._SetIntersect3 3578
System._SetIntersect3 function 3578
System._SetLe 3578
System._SetLe function 3578
System._SetLength 3578
System._SetLength function 3578
System._SetRange 3579
System._SetRange function 3579
System._SetString 3579
System._SetString function 3579
System._SetSub 3579
System._SetSub function 3579
System._SetSub3 3580
System._SetSub3 function 3580
System._SetTextBuf 3580
System._SetTextBuf function 3580
System._SetUnion 3580
System._SetUnion function 3580
System._SetUnion3 3581
System._SetUnion3 function 3581
System._SGetDir 3581
System._SGetDir function 3581
System._StartExe 3581
System._StartExe function 3581
System._StartLib 3582
System._StartLib function 3582
System._Str0Ext 3582
System._Str0Ext function 3582
System._Str0Int64 3582
System._Str0Int64 function 3582
System._Str0Long 3583
System._Str0Long function 3583
System._Str0UInt64 3583
System._Str0UInt64 function 3583
System._Str1Ext 3583
System._Str1Ext function 3583
System._Str2Ext 3584
System._Str2Ext function 3584
System._StrInt64 3584
System._StrInt64 function 3584
System._StrLong 3584
System._StrLong function 3584
System._StrUInt64 3585
System._StrUInt64 function 3585
System._TRUNC 3585
System._TRUNC function 3585
System._Truncate 3585
System._Truncate function 3585
System._TryFinallyExit 3586
System._TryFinallyExit function 3586
System._UGetDir 3586
System._UGetDir function 3586
System._UniqueStringA 3586
System._UniqueStringA function 3586
System._UniqueStringU 3587
System._UniqueStringU function 3587
System._UniqueStringW 3587
System._UniqueStringW function 3587
System._UStrAddRef 3587
System._UStrAddRef function 3587
System._UStrArrayClr 3588
System._UStrArrayClr function 3588
System._UStrAsg 3588

System._UStrAsg function 3588
System._UStrCat 3588
System._UStrCat function 3588
System._UStrCat3 3589
System._UStrCat3 function 3589
System._UStrCatN 3589
System._UStrCatN function 3589
System._UStrClr 3589
System._UStrClr function 3589
System._UStrCmp 3590
System._UStrCmp function 3590
System._UStrCopy 3590
System._UStrCopy function 3590
System._UStrDelete 3590
System._UStrDelete function 3590
System._UStrEqual 3591
System._UStrEqual function 3591
System._UStrFromArray 3591
System._UStrFromArray function 3591
System._UStrFromChar 3591
System._UStrFromChar function 3591
System._UStrFromLStr 3592
System._UStrFromLStr function 3592
System._UStrFromPChar 3592
System._UStrFromPChar function 3592
System._UStrFromPCharLen 3592
System._UStrFromPCharLen function 3592
System._UStrFromPWChar 3593
System._UStrFromPWChar function 3593
System._UStrFromPWCharLen 3593
System._UStrFromPWCharLen function 3593
System._UStrFromString 3593
System._UStrFromString function 3593
System._UStrFromWArray 3594
System._UStrFromWArray function 3594
System._UStrFromWChar 3594
System._UStrFromWChar function 3594
System._UStrFromWStr 3594
System._UStrFromWStr function 3594
System._UStrInsert 3595
System._UStrInsert function 3595
System._UStrLAsg 3595
System._UStrLAsg function 3595
System._UStrLen 3595
System._UStrLen function 3595
System._UStrSetLength 3596
System._UStrSetLength function 3596
System._UStrToPWChar 3596
System._UStrToPWChar function 3596
System._UStrToString 3596
System._UStrToString function 3596
System._ValExt 3597
System._ValExt function 3597
System._ValInt64 3597
System._ValInt64 function 3597
System._ValLong 3597
System._ValLong function 3597
System._WCharToString 3598
System._WCharToString function 3598
System._WGetDir 3598
System._WGetDir function 3598
System._Write0Bool 3598
System._Write0Bool function 3598
System._Write0Char 3599
System._Write0Char function 3599
System._Write0CString 3599
System._Write0CString function 3599
System._Write0Ext 3599
System._Write0Ext function 3599
System._Write0Int64 3600
System._Write0Int64 function 3600
System._Write0Long 3600
System._Write0Long function 3600
System._Write0LString 3600
System._Write0LString function 3600
System._Write0String 3601
System._Write0String function 3601
System._Write0UInt64 3601

System._Write0UInt64 function 3601
System._Write0UString 3601
System._Write0UString function 3601
System._Write0Variant 3602
System._Write0Variant function 3602
System._Write0WChar 3602
System._Write0WChar function 3602
System._Write0WCString 3602
System._Write0WCString function 3602
System._Write0WString 3603
System._Write0WString function 3603
System._Write1Ext 3603
System._Write1Ext function 3603
System._Write2Ext 3603
System._Write2Ext function 3603
System._WriteBool 3604
System._WriteBool function 3604
System._WriteChar 3604
System._WriteChar function 3604
System._WriteCString 3604
System._WriteCString function 3604
System._WriteInt64 3605
System._WriteInt64 function 3605
System._WriteLn 3605
System._WriteLn function 3605
System._WriteLong 3605
System._WriteLong function 3605
System._WriteLString 3606
System._WriteLString function 3606
System._WriteRec 3606
System._WriteRec function 3606
System._WriteString 3606
System._WriteString function 3606
System._WriteUInt64 3607
System._WriteUInt64 function 3607
System._WriteUString 3607
System._WriteUString function 3607
System._WriteVariant 3607
System._WriteVariant function 3607
System._WriteWChar 3608
System._WriteWChar function 3608
System._WriteWCString 3608
System._WriteWCString function 3608
System._WriteWString 3608
System._WriteWString function 3608
System._WStrAddRef 3609
System._WStrAddRef function 3609
System._WStrArrayClr 3609
System._WStrArrayClr function 3609
System._WStrAsg 3609
System._WStrAsg function 3609
System._WStrCat 3610
System._WStrCat function 3610
System._WStrCat3 3610
System._WStrCat3 function 3610
System._WStrCatN 3610
System._WStrCatN function 3610
System._WStrClr 3611
System._WStrClr function 3611
System._WStrCmp 3611
System._WStrCmp function 3611
System._WStrCopy 3611
System._WStrCopy function 3611
System._WStrDelete 3612
System._WStrDelete function 3612
System._WStrEqual 3612
System._WStrEqual function 3612
System._WStrFromArray 3612
System._WStrFromArray function 3612
System._WStrFromChar 3613
System._WStrFromChar function 3613
System._WStrFromLStr 3613
System._WStrFromLStr function 3613
System._WStrFromPChar 3613
System._WStrFromPChar function 3613
System._WStrFromPCharLen 3614
System._WStrFromPCharLen function 3614
System._WStrFromPWChar 3614

System._WStrFromPWChar function 3614
System._WStrFromPWCharLen 3614
System._WStrFromPWCharLen function 3614
System._WStrFromString 3615
System._WStrFromString function 3615
System._WStrFromUStr 3615
System._WStrFromUStr function 3615
System._WStrFromWArray 3615
System._WStrFromWArray function 3615
System._WStrFromWChar 3616
System._WStrFromWChar function 3616
System._WStrInsert 3616
System._WStrInsert function 3616
System._WStrLAsg 3616
System._WStrLAsg function 3616
System._WStrLCmp 3617
System._WStrLCmp function 3617
System._WStrLen 3617
System._WStrLen function 3617
System._WStrSetLength 3617
System._WStrSetLength function 3617
System._WStrToPWChar 3618
System._WStrToPWChar function 3618
System._WStrToString 3618
System._WStrToString function 3618
System.Abs 3619
System.Abs function 3619
System.AbstractErrorProc 3875
System.AbstractErrorProc variable 3875
System.AcquireExceptionObject 3619
System.AcquireExceptionObject function 3619
System.AddModuleUnloadProc 3620
System.AddModuleUnloadProc function 3620
System.Addr 3621
System.Addr function 3621
System.AllocMem 3622
System.AllocMem function 3622
System.AllocMemCount 3875
System.AllocMemCount variable 3875
System.AllocMemSize 3877
System.AllocMemSize variable 3877
System.AnsiToUtf8 3624
System.AnsiToUtf8 function 3624
System.Append 3624
System.Append function 3624
System.ArcTan 3625
System.ArcTan function 3625
System.Assert 3627
System.Assert function 3627
System.AssertErrorProc 3878
System.AssertErrorProc variable 3878
System.Assigned 3628
System.Assigned function 3628
System.AssignFile 3629
System.AssignFile function 3629
System.AttemptToUseSharedMemoryManager 3631
System.AttemptToUseSharedMemoryManager function 3631
System.BeginThread 3632
System.BeginThread function 3632
System.BlockRead 3633
System.BlockRead function 3633
System.BlockWrite 3634
System.BlockWrite function 3634
System.Break 3636
System.Break function 3636
System.ChDir 3638
System.ChDir function 3638
System.Chr 3640
System.Chr function 3640
System.Close 3643
System.Close function 3643
System.CloseFile 3643
System.CloseFile function 3643
System.CmdLine 3880
System.CmdLine variable 3880
System.CmdShow 3880
System.CmdShow variable 3880
System.CompareTo 3645

System.CompareTo function 3645
System.CompilerVersion 3911
System.CompilerVersion constant 3911
System.CompToCurrency 3645
System.CompToCurrency function 3645
System.CompToDouble 3646
System.CompToDouble function 3646
System.Concat 3646
System.Concat function 3646
System.Continue 3647
System.Continue function 3647
System.Copy 3648
System.Copy function 3648
System.Cos 3650
System.Cos function 3650
System.CppLongInt 3824
System.CppLongInt type 3824
System.CppULongInt 3825
System.CppULongInt type 3825
System.CPUCount 3881
System.CPUCount variable 3881
System.CPUi386 3911
System.CPUi386 constant 3911
System.CPUi486 3912
System.CPUi486 constant 3912
System.CPUPentium 3912
System.CPUPentium constant 3912
System.CurrencyToComp 3651
System.CurrencyToComp function 3651
System.DebugHook 3881
System.DebugHook variable 3881
System.Dec 3652
System.Dec function 3652
System.Default8087CW 3881
System.Default8087CW variable 3881
System.DefaultSystemCodePage 3883
System.DefaultSystemCodePage variable 3883
System.DefaultTextLineBreakStyle 3883
System.DefaultTextLineBreakStyle variable 3883
System.DefaultUnicodeCodePage 3884
System.DefaultUnicodeCodePage variable 3884
System.Delete 3653
System.Delete function 3653
System.DispCallByIDProc 3884
System.DispCallByIDProc variable 3884
System.Dispose 3654
System.Dispose function 3654
System.DLLProc 3884
System.DLLProc variable 3884
System.DoubleToComp 3656
System.DoubleToComp function 3656
System.DynArrayClear 3656
System.DynArrayClear function 3656
System.DynArrayDim 3657
System.DynArrayDim function 3657
System.DynArraySetLength 3657
System.DynArraySetLength function 3657
System.E_NOINTERFACE 3912
System.E_NOINTERFACE constant 3912
System.E_NOTIMPL 3913
System.E_NOTIMPL constant 3913
System.E_UNEXPECTED 3913
System.E_UNEXPECTED constant 3913
System.EndThread 3657
System.EndThread function 3657
System.EnumModules 3658
System.EnumModules function 3658
System.EnumResourceModules 3658
System.EnumResourceModules function 3658
System.envp 3885
System.envp variable 3885
System.Eof 3659
System.Eof function 3659
System.Eoln 3660
System.Eoln function 3660
System.Equals 3661
System.Equals function 3661
System.Erase 3661

System.Erase function 3661
System.Error 3662
System.Error function 3662
System.ErrorAddr 3885
System.ErrorAddr variable 3885
System.ErrorProc 3886
System.ErrorProc variable 3886
System.ErrOutput 3888
System.ErrOutput variable 3888
System.ExceptAddr 3663
System.ExceptAddr function 3663
System.ExceptClsProc 3888
System.ExceptClsProc variable 3888
System.ExceptionAcquired 3889
System.ExceptionAcquired variable 3889
System.ExceptionClass 3889
System.ExceptionClass variable 3889
System.ExceptObject 3664
System.ExceptObject function 3664
System.ExceptObjProc 3889
System.ExceptObjProc variable 3889
System.ExceptProc 3890
System.ExceptProc variable 3890
System.Exclude 3664
System.Exclude function 3664
System.Exit 3665
System.Exit function 3665
System.ExitCode 3891
System.ExitCode variable 3891
System.ExitProc 3891
System.ExitProc variable 3891
System.ExitProcessProc 3892
System.ExitProcessProc variable 3892
System.Exp 3667
System.Exp function 3667
System.FileMode 3892
System.FileMode variable 3892
System.FilePos 3668
System.FilePos function 3668
System.FileSize 3669
System.FileSize function 3669
System.FillChar 3670
System.FillChar function 3670
System.Finalize 3672
System.Finalize function 3672
System.FindClassHInstance 3672
System.FindClassHInstance function 3672
System.FindHInstance 3673
System.FindHInstance function 3673
System.FindResourceHInstance 3674
System.FindResourceHInstance function 3674
System.Flush 3674
System.Flush function 3674
System.fmClosed 3913
System.fmClosed constant 3913
System.fmInOut 3914
System.fmInOut constant 3914
System.fmInput 3914
System.fmInput constant 3914
System.fmOutput 3914
System.fmOutput constant 3914
System.FPower10 3675
System.FPower10 function 3675
System.Frac 3675
System.Frac function 3675
System.FreeMem 3676
System.FreeMem function 3676
System.FreeMemory 3678
System.FreeMemory function 3678
System.Get8087CW 3679
System.Get8087CW function 3679
System.GetCurrent 3679
System.GetCurrent function 3679
System.GetDir 3679
System.GetDir function 3679
System.GetEnumerator 3681
System.GetEnumerator function 3681
System.GetHeapStatus 3681

System.GetHeapStatus function 3681
System.GetLastError 3681
System.GetLastError function 3681
System.GetMem 3683
System.GetMem function 3683
System.GetMemory 3684
System.GetMemory function 3684
System.GetMemoryManager 3686
System.GetMemoryManager function 3686
System.GetMemoryManagerState 3690
System.GetMemoryManagerState function 3690
System.GetMemoryMap 3691
System.GetMemoryMap function 3691
System.GetMinimumBlockAlignment 3691
System.GetMinimumBlockAlignment function 3691
System.GetModuleFileName 3692
System.GetModuleFileName function 3692
System.GetPackageInfoTable 3825
System.GetPackageInfoTable type 3825
System.GetVariantManager 3692
System.GetVariantManager function 3692
System.Halt 3693
System.Halt function 3693
System.HeapAllocFlags 3893
System.HeapAllocFlags variable 3893
System.hfFieldSize 3915
System.hfFieldSize constant 3915
System.hfMonitorOffset 3915
System.hfMonitorOffset constant 3915
System.HGLOBAL 3825
System.HGLOBAL type 3825
System.Hi 3694
System.Hi function 3694
System.High 3694
System.High function 3694
System.HINST 3826
System.HINST type 3826
System.HMODULE 3826
System.HMODULE type 3826
System.HPrevInst 3894
System.HPrevInst variable 3894
System.HRESULT 3826
System.HRESULT type 3826
System.HRSRC 3827
System.HRSRC type 3827
System.IComparable 3497
System.IComparable.CompareTo 3498
System.IDispatch 3498
System.IDispatch.GetIDsOfNames 3499
System.IDispatch.GetTypeInfo 3499
System.IDispatch.GetTypeInfoCount 3499
System.IDispatch.Invoke 3499
System.IEnumerable 3500
System.IEnumerable.GetEnumerator 3500
System.IEnumerator 3501
System.IEnumerator.Current 3502
System.IEnumerator.GetCurrent 3501
System.IEnumerator.MoveNext 3502
System.IEnumerator.Reset 3502
System.IInterface 3503
System.IInterface._AddRef 3504
System.IInterface._Release 3504
System.IInterface.QueryInterface 3505
System.IInvokable 3505
System.Inc 3696
System.Inc function 3696
System.Include 3697
System.Include function 3697
System.INFINITE 3915
System.INFINITE constant 3915
System.Initialize 3698
System.Initialize function 3698
System.InitProc 3894
System.InitProc variable 3894
System.Input 3895
System.Input variable 3895
System.Insert 3698
System.Insert function 3698

System.Int 3699
System.Int function 3699
System.Int16 3827
System.Int16 type 3827
System.Int32 3827
System.Int32 type 3827
System.Int8 3828
System.Int8 type 3828
System.IntegerArray 3828
System.IntegerArray type 3828
System.IOResult 3700
System.IOResult function 3700
System.IsConsole 3896
System.IsConsole variable 3896
System.IsLibrary 3896
System.IsLibrary variable 3896
System.IsMemoryManagerSet 3701
System.IsMemoryManagerSet function 3701
System.IsMultiThread 3897
System.IsMultiThread variable 3897
System.IsVariantManagerSet 3701
System.IsVariantManagerSet function 3701
System.IUnknown 3828
System.IUnknown type 3828
System.JITEnable 3897
System.JITEnable variable 3897
System.Length 3702
System.Length function 3702
System.LibModuleList 3898
System.LibModuleList variable 3898
System.Ln 3703
System.Ln function 3703
System.Lo 3704
System.Lo function 3704
System.LoadResourceModule 3705
System.LoadResourceModule function 3705
System.LoadResString 3705
System.LoadResString function 3705
System.Low 3706
System.Low function 3706
System.MainInstance 3898
System.MainInstance variable 3898
System.MainThreadID 3899
System.MainThreadID variable 3899
System.Mark 3707
System.Mark function 3707
System.MaxInt 3916
System.MaxInt constant 3916
System.MaxLongint 3916
System.MaxLongint constant 3916
System.MemoryBarrier 3707
System.MemoryBarrier function 3707
System.MkDir 3708
System.MkDir function 3708
System.ModuleUnloadList 3899
System.ModuleUnloadList variable 3899
System.MonitorEnter 3709
System.MonitorEnter function 3709
System.MonitorExit 3709
System.MonitorExit function 3709
System.MonitorPulse 3710
System.MonitorPulse function 3710
System.MonitorPulseAll 3710
System.MonitorPulseAll function 3710
System.MonitorSupport 3900
System.MonitorSupport variable 3900
System.MonitorTryEnter 3710
System.MonitorTryEnter function 3710
System.MonitorWait 3711
System.MonitorWait function 3711
System.Move 3711
System.Move function 3711
System.MoveChars 3712
System.MoveChars function 3712
System.NeverSleepOnMMThreadContention 3900
System.NeverSleepOnMMThreadContention variable 3900
System.New 3712
System.New function 3712

System.NoErrMsg 3901
System.NoErrMsg variable 3901
System.NumSmallBlockTypes 3916
System.NumSmallBlockTypes constant 3916
System.Odd 3714
System.Odd function 3714
System.OleStrToString 3715
System.OleStrToString function 3715
System.OleStrToStrVar 3715
System.OleStrToStrVar function 3715
System.opAdd 3917
System.opAdd constant 3917
System.opAnd 3917
System.opAnd constant 3917
System.opCmpEQ 3918
System.opCmpEQ constant 3918
System.opCmpGE 3918
System.opCmpGE constant 3918
System.opCmpGT 3919
System.opCmpGT constant 3919
System.opCmpLE 3919
System.opCmpLE constant 3919
System.opCmpLT 3920
System.opCmpLT constant 3920
System.opCmpNE 3920
System.opCmpNE constant 3920
System.opCompare 3921
System.opCompare constant 3921
System.opDivide 3921
System.opDivide constant 3921
System.opIntDivide 3922
System.opIntDivide constant 3922
System.opModulus 3922
System.opModulus constant 3922
System.opMultiply 3923
System.opMultiply constant 3923
System.opNegate 3923
System.opNegate constant 3923
System.opNot 3924
System.opNot constant 3924
System.opOr 3924
System.opOr constant 3924
System.opShiftLeft 3925
System.opShiftLeft constant 3925
System.opShiftRight 3925
System.opShiftRight constant 3925
System.opSubtract 3926
System.opSubtract constant 3926
System.opXor 3926
System.opXor constant 3926
System.Ord 3716
System.Ord function 3716
System.Output 3901
System.Output variable 3901
System.PackageInfo 3829
System.PackageInfo type 3829
System.PackageInfoTable 3795
System.PackageInfoTable record 3795
System.PackageUnitEntry 3795
System.PackageUnitEntry record 3795
System.PAnsiString 3829
System.PAnsiString type 3829
System.ParamCount 3716
System.ParamCount function 3716
System.ParamStr 3718
System.ParamStr function 3718
System.PBoolean 3829
System.PBoolean type 3829
System.PByte 3830
System.PByte type 3830
System.PCardinal 3831
System.PCardinal type 3831
System.PComp 3831
System.PComp type 3831
System.PCurrency 3831
System.PCurrency type 3831
System.PCVModInfo 3832
System.PCVModInfo type 3832

System.PDate 3832
System.PDate type 3832
System.PDateTime 3833
System.PDateTime type 3833
System.PDispatch 3833
System.PDispatch type 3833
System.PDouble 3834
System.PDouble type 3834
System.PDynArrayTypeInfo 3834
System.PDynArrayTypeInfo type 3834
System.PError 3834
System.PError type 3834
System.PExtended 3835
System.PExtended type 3835
System.PGUID 3835
System.PGUID type 3835
System.Pi 3719
System.Pi function 3719
System.PInitContext 3837
System.PInitContext type 3837
System.PInt64 3837
System.PInt64 type 3837
System.PInteger 3837
System.PInteger type 3837
System.PIntegerArray 3838
System.PIntegerArray type 3838
System.PInterfaceEntry 3838
System.PInterfaceEntry type 3838
System.PInterfaceTable 3839
System.PInterfaceTable type 3839
System.PLibModule 3839
System.PLibModule type 3839
System.PLongint 3839
System.PLongint type 3839
System.PLongWord 3840
System.PLongWord type 3840
System.PMonitor 3841
System.PMonitor type 3841
System.PMonitorSupport 3841
System.PMonitorSupport type 3841
System.PointerArray 3842
System.PointerArray type 3842
System.POleVariant 3842
System.POleVariant type 3842
System.Pos 3720
System.Pos function 3720
System.PPAnsiChar 3842
System.PPAnsiChar type 3842
System.PPCharArray 3843
System.PPCharArray type 3843
System.PPDispatch 3843
System.PPDispatch type 3843
System.PPMonitor 3843
System.PPMonitor type 3843
System.PPointer 3844
System.PPointer type 3844
System.PPointerArray 3844
System.PPointerArray type 3844
System.PPUnknown 3844
System.PPUnknown type 3844
System.PPWideChar 3845
System.PPWideChar type 3845
System.PRawByteString 3845
System.PRawByteString type 3845
System.Pred 3721
System.Pred function 3721
System.PShortInt 3846
System.PShortInt type 3846
System.PShortString 3846
System.PShortString type 3846
System.PSingle 3846
System.PSingle type 3846
System.PSmallInt 3847
System.PSmallInt type 3847
System.PTextBuf 3848
System.PTextBuf type 3848
System.Ptr 3723
System.Ptr function 3723

System.PUCS4Char 3848
System.PUCS4Char type 3848
System.PUCS4CharArray 3848
System.PUCS4CharArray type 3848
System.PUCS4Chars 3724
System.PUCS4Chars function 3724
System.PUInt64 3849
System.PUInt64 type 3849
System.PUnicodeString 3849
System.PUnicodeString type 3849
System.PUnitEntryTable 3849
System.PUnitEntryTable type 3849
System.PUnknown 3850
System.PUnknown type 3850
System.PUTF8String 3850
System.PUTF8String type 3850
System.PVarArray 3850
System.PVarArray type 3850
System.PVarArrayBoundArray 3851
System.PVarArrayBoundArray type 3851
System.PVarArrayCoorArray 3851
System.PVarArrayCoorArray type 3851
System.PVarData 3851
System.PVarData type 3851
System.PVariant 3852
System.PVariant type 3852
System.PWideString 3853
System.PWideString type 3853
System.PWord 3853
System.PWord type 3853
System.PWordBool 3853
System.PWordBool type 3853
System.RaiseExceptionProc 3902
System.RaiseExceptionProc variable 3902
System.RaiseExceptObjProc 3902
System.RaiseExceptObjProc variable 3902
System.RaiseList 3724
System.RaiseList function 3724
System.Random 3725
System.Random function 3725
System.Randomize 3726
System.Randomize function 3726
System.RandSeed 3902
System.RandSeed variable 3902
System.RawByteString 3854
System.RawByteString type 3854
System.Read 3727
System.Read function 3727
System.ReadLn 3728
System.ReadLn function 3728
System.ReallocMem 3729
System.ReallocMem function 3729
System.ReallocMemory 3730
System.ReallocMemory function 3730
System.RegisterExpectedMemoryLeak 3731
System.RegisterExpectedMemoryLeak function 3731
System.RegisterModule 3731
System.RegisterModule function 3731
System.Release 3732
System.Release function 3732
System.ReleaseExceptionObject 3732
System.ReleaseExceptionObject function 3732
System.RemoveModuleUnloadProc 3732
System.RemoveModuleUnloadProc function 3732
System.Rename 3733
System.Rename function 3733
System.ReportMemoryLeaksOnShutdown 3903
System.ReportMemoryLeaksOnShutdown variable 3903
System.Reset 3734
System.Reset function 3734
System.Rewrite 3736
System.Rewrite function 3736
System.RmDir 3738
System.RmDir function 3738
System.Round 3739
System.Round function 3739
System.RTLUnwindProc 3903
System.RTLUnwindProc variable 3903

System.RTLVersion 3927
System.RTLVersion constant 3927
System.RunError 3740
System.RunError function 3740
System.S_FALSE 3927
System.S_FALSE constant 3927
System.S_OK 3927
System.S_OK constant 3927
System.SafeCallErrorProc 3904
System.SafeCallErrorProc variable 3904
System.Seek 3741
System.Seek function 3741
System.SeekEof 3742
System.SeekEof function 3742
System.SeekEoln 3743
System.SeekEoln function 3743
System.Set8087CW 3744
System.Set8087CW function 3744
System.SetCodePage 3746
System.SetCodePage function 3746
System.SetInOutRes 3746
System.SetInOutRes function 3746
System.SetLastError 3747
System.SetLastError function 3747
System.SetLength 3748
System.SetLength function 3748
System.SetLineBreakStyle 3750
System.SetLineBreakStyle function 3750
System.SetMemoryManager 3751
System.SetMemoryManager function 3751
System.SetMinimumBlockAlignment 3756
System.SetMinimumBlockAlignment function 3756
System.SetMultiByteConversionCodePage 3756
System.SetMultiByteConversionCodePage function 3756
System.SetRaiseList 3757
System.SetRaiseList function 3757
System.SetString 3757
System.SetString function 3757
System.SetTextBuf 3758
System.SetTextBuf function 3758
System.SetVariantManager 3759
System.SetVariantManager function 3759
System.ShareMemoryManager 3759
System.ShareMemoryManager function 3759
System.Sin 3760
System.Sin function 3760
System.SizeOf 3761
System.SizeOf function 3761
System.Slice 3762
System.Slice function 3762
System.sLineBreak 3928
System.sLineBreak constant 3928
System.Sqr 3762
System.Sqr function 3762
System.Sqrt 3763
System.Sqrt function 3763
System.Str 3764
System.Str function 3764
System.StringCodePage 3765
System.StringCodePage function 3765
System.StringElementSize 3765, 3766
System.StringElementSize function 3765, 3766
System.StringOfChar 3766
System.StringOfChar function 3766
System.StringRefCount 3766, 3767
System.StringRefCount function 3766, 3767
System.StringToOleStr 3767
System.StringToOleStr function 3767
System.StringToWideChar 3768
System.StringToWideChar function 3768
System.Succ 3768
System.Succ function 3768
System.Swap 3769
System.Swap function 3769
System.SysAllocMem 3769
System.SysAllocMem function 3769
System.SysFreeMem 3770
System.SysFreeMem function 3770

System.SysGetMem 3770
System.SysGetMem function 3770
System.SysReallocMem 3771
System.SysReallocMem function 3771
System.SysRegisterExpectedMemoryLeak 3772
System.SysRegisterExpectedMemoryLeak function 3772
System.SystemThreadEndProc 3904
System.SystemThreadEndProc variable 3904
System.SystemThreadFuncProc 3905
System.SystemThreadFuncProc variable 3905
System.SysUnregisterExpectedMemoryLeak 3772
System.SysUnregisterExpectedMemoryLeak function 3772
System.TAssertErrorProc 3855
System.TAssertErrorProc type 3855
System.TBoundArray 3855
System.TBoundArray type 3855
System.TCallDesc 3795
System.TCallDesc record 3795
System.TChunkStatus 3796
System.TChunkStatus enumeration 3796
System.TClass 3856
System.TClass type 3856
System.TClassHelperBase 3331
System.TClassHelperBase._Create 3333
System.TClassHelperBase.FInstance 3333
System.TClassHelperBaseClass 3857
System.TClassHelperBaseClass type 3857
System.TCVModInfo 3797
System.TCVModInfo record 3797
System.TDate 3857
System.TDate type 3857
System.TDateTime 3858
System.TDateTime type 3858
System.TDispatchMessage 3797
System.TDispatchMessage record 3797
System.TDispDesc 3798
System.TDispDesc record 3798
System.TDLLProc 3859
System.TDLLProc type 3859
System.TDLLProcEx 3859
System.TDLLProcEx type 3859
System.TDynArrayTypeInfo 3798
System.TDynArrayTypeInfo record 3798
System.TEnumModuleFunc 3860
System.TEnumModuleFunc type 3860
System.TEnumModuleFuncLW 3860
System.TEnumModuleFuncLW type 3860
System.Test8086 3905
System.Test8086 variable 3905
System.Test8087 3905
System.Test8087 variable 3905
System.TestFDIV 3906
System.TestFDIV variable 3906
System.TextStart 3772
System.TextStart function 3772
System.tfCRLF 3928
System.tfCRLF constant 3928
System.TFileIOFunc 3860
System.TFileIOFunc type 3860
System.TFileRec 3799
System.TFileRec record 3799
System.TGUID 3800
System.TGUID record 3800
System.TInitContext 3803
System.TInitContext record 3803
System.TInterfacedClass 3861
System.TInterfacedClass type 3861
System.TInterfaceEntry 3803
System.TInterfaceEntry record 3803
System.TInterfaceTable 3804
System.TInterfaceTable record 3804
System.TMemoryMap 3862
System.TMemoryMap type 3862
System.TMethod 3806
System.TMethod record 3806
System.TMinimumBlockAlignment 3807
System.TMinimumBlockAlignment enumeration 3807
System.TModuleUnloadProc 3863

System.TModuleUnloadProc type 3863
System.TModuleUnloadProcLW 3863
System.TModuleUnloadProcLW type 3863
System.TMonitor 3361
System.TMonitor.Enter 3362, 3363
System.TMonitor.Exit 3363
System.TMonitor.Pulse 3363
System.TMonitor.PulseAll 3363
System.TMonitor.SetSpinCount 3363
System.TMonitor.TryEnter 3364
System.TMonitor.Wait 3364
System.TMonitorSupport 3808
System.TMonitorSupport record 3808
System.TPCharArray 3863
System.TPCharArray type 3863
System.TResourceHandle 3864
System.TResourceHandle type 3864
System.Trunc 3773
System.Trunc function 3773
System.Truncate 3774
System.Truncate function 3774
System.TRuntimeError 3809
System.TRuntimeError enumeration 3809
System.TSafeCallErrorProc 3864
System.TSafeCallErrorProc type 3864
System.TSystemThreadEndProc 3866
System.TSystemThreadEndProc type 3866
System.TSystemThreadFuncProc 3866
System.TSystemThreadFuncProc type 3866
System.TTextBuf 3867
System.TTextBuf type 3867
System.TTextIOFunc 3867
System.TTextIOFunc type 3867
System.TTextLineBreakStyle 3811
System.TTextLineBreakStyle enumeration 3811
System.TTextRec 3811
System.TTextRec record 3811
System.TTime 3868
System.TTime type 3868
System.TUCS4CharArray 3868
System.TUCS4CharArray type 3868
System.TVarArrayBoundArray 3868
System.TVarArrayBoundArray type 3868
System.TVarArrayCoorArray 3869
System.TVarArrayCoorArray type 3869
System.TVarData 3813
System.TVarData record 3813
System.TVarOp 3869
System.TVarOp type 3869
System.UCS4Char 3871
System.UCS4Char type 3871
System.UCS4String 3871
System.UCS4String type 3871
System.UCS4StringToUnicodeString 3775
System.UCS4StringToUnicodeString function 3775
System.UCS4StringToWideString 3775
System.UCS4StringToWideString function 3775
System.UInt16 3872
System.UInt16 type 3872
System.UInt32 3872
System.UInt32 type 3872
System.UInt8 3872
System.UInt8 type 3872
System.UnicodeStringToUCS4String 3775
System.UnicodeStringToUCS4String function 3775
System.UnicodeToUtf8 3776
System.UnicodeToUtf8 function 3776
System.UniqueString 3777
System.UniqueString function 3777
System.UnitEntryTable 3873
System.UnitEntryTable type 3873
System.UnregisterExpectedMemoryLeak 3777
System.UnregisterExpectedMemoryLeak function 3777
System.UnregisterModule 3778
System.UnregisterModule function 3778
System.UpCase 3778
System.UpCase function 3778
System.UTF8CompareLocale 3906

System.UTF8CompareLocale variable 3906
System.UTF8Decode 3779
System.UTF8Decode function 3779
System.UTF8Encode 3780
System.UTF8Encode function 3780
System.UTF8EncodeToShortString 3780, 3781
System.UTF8EncodeToShortString function 3780, 3781
System.UTF8String 3873
System.UTF8String type 3873
System.Utf8ToAnsi 3781
System.Utf8ToAnsi function 3781
System.UTF8ToString 3782, 3783
System.UTF8ToString function 3782, 3783
System.Utf8ToUnicode 3783
System.Utf8ToUnicode function 3783
System.UTF8ToUnicodeString 3784
System.UTF8ToUnicodeString function 3784
System.UTF8ToWideString 3785
System.UTF8ToWideString function 3785
System.Val 3785
System.Val function 3785
System.VarAddRefProc 3906
System.VarAddRefProc variable 3906
System.varAny 3928
System.varAny constant 3928
System.varArray 3929
System.varArray constant 3929
System.VarArrayRedim 3786
System.VarArrayRedim function 3786
System.varBoolean 3930
System.varBoolean constant 3930
System.varByRef 3930
System.varByRef constant 3930
System.varByte 3931
System.varByte constant 3931
System.VarClear 3786
System.VarClear function 3786
System.VarClearProc 3907
System.VarClearProc variable 3907
System.VarCopyProc 3907
System.VarCopyProc variable 3907
System.varCurrency 3931
System.varCurrency constant 3931
System.varDate 3932
System.varDate constant 3932
System.varDispatch 3932
System.varDispatch constant 3932
System.varDouble 3933
System.varDouble constant 3933
System.varEmpty 3933
System.varEmpty constant 3933
System.varError 3934
System.varError constant 3934
System.varInt64 3934
System.varInt64 constant 3934
System.varInteger 3935
System.varInteger constant 3935
System.varLongWord 3935
System.varLongWord constant 3935
System.varNull 3936
System.varNull constant 3936
System.varOleStr 3936
System.varOleStr constant 3936
System.varShortInt 3937
System.varShortInt constant 3937
System.varSingle 3937
System.varSingle constant 3937
System.varSmallint 3938
System.varSmallint constant 3938
System.varStrArg 3938
System.varStrArg constant 3938
System.varString 3938
System.varString constant 3938
System.VarToLStrProc 3907
System.VarToLStrProc variable 3907
System.VarToWStrProc 3908
System.VarToWStrProc variable 3908
System.varTypeMask 3939

System.varTypeMask constant 3939
System.varUInt64 3940
System.varUInt64 constant 3940
System.varUnknown 3940
System.varUnknown constant 3940
System.varUString 3940
System.varUString constant 3940
System.varVariant 3941
System.varVariant constant 3941
System.varWord 3941
System.varWord constant 3941
System.vmtAddRef 3942
System.vmtAddRef constant 3942
System.vmtAfterConstruction 3942
System.vmtAfterConstruction constant 3942
System.vmtAutoTable 3942
System.vmtAutoTable constant 3942
System.vmtBeforeDestruction 3943
System.vmtBeforeDestruction constant 3943
System.vmtClassName 3943
System.vmtClassName constant 3943
System.vmtCreateObject 3943
System.vmtCreateObject constant 3943
System.vmtDefaultHandler 3944
System.vmtDefaultHandler constant 3944
System.vmtDestroy 3944
System.vmtDestroy constant 3944
System.vmtDispatch 3944
System.vmtDispatch constant 3944
System.vmtDynamicTable 3945
System.vmtDynamicTable constant 3945
System.vmtEquals 3945
System.vmtEquals constant 3945
System.vmtFieldTable 3945
System.vmtFieldTable constant 3945
System.vmtFreeInstance 3946
System.vmtFreeInstance constant 3946
System.vmtGetHashCode 3946
System.vmtGetHashCode constant 3946
System.vmtInitTable 3946
System.vmtInitTable constant 3946
System.vmtInstanceSize 3947
System.vmtInstanceSize constant 3947
System.vmtIntfTable 3947
System.vmtIntfTable constant 3947
System.vmtMethodTable 3947
System.vmtMethodTable constant 3947
System.vmtNewInstance 3948
System.vmtNewInstance constant 3948
System.vmtParent 3948
System.vmtParent constant 3948
System.vmtQueryInterface 3948
System.vmtQueryInterface constant 3948
System.vmtRelease 3949
System.vmtRelease constant 3949
System.vmtSafeCallException 3949
System.vmtSafeCallException constant 3949
System.vmtSelfPtr 3949
System.vmtSelfPtr constant 3949
System.vmtToString 3950
System.vmtToString constant 3950
System.vmtTypeInfo 3950
System.vmtTypeInfo constant 3950
System.vtAnsiString 3950
System.vtAnsiString constant 3950
System.vtBoolean 3951
System.vtBoolean constant 3951
System.vtChar 3951
System.vtChar constant 3951
System.vtClass 3952
System.vtClass constant 3952
System.vtCurrency 3952
System.vtCurrency constant 3952
System.vtExtended 3953
System.vtExtended constant 3953
System.vtInt64 3953
System.vtInt64 constant 3953
System.vtInteger 3954

System.vtInteger constant 3954
System.vtInterface 3954
System.vtInterface constant 3954
System.vtObject 3955
System.vtObject constant 3955
System.vtPChar 3955
System.vtPChar constant 3955
System.vtPointer 3956
System.vtPointer constant 3956
System.vtPWideChar 3956
System.vtPWideChar constant 3956
System.vtString 3957
System.vtString constant 3957
System.vtUnicodeString 3957
System.vtUnicodeString constant 3957
System.vtVariant 3957
System.vtVariant constant 3957
System.vtWideChar 3958
System.vtWideChar constant 3958
System.vtWideString 3958
System.vtWideString constant 3958
System.WideCharLenToString 3787
System.WideCharLenToString function 3787
System.WideCharLenToStrVar 3788
System.WideCharLenToStrVar function 3788
System.WideCharToString 3788
System.WideCharToString function 3788
System.WideCharToStrVar 3788
System.WideCharToStrVar function 3788
System.WideStringToUCS4String 3789
System.WideStringToUCS4String function 3789
System.Write 3789
System.Write function 3789
System.WriteLn 3792
System.WriteLn function 3792
System.YieldProcessor 3793
System.YieldProcessor function 3793
System::__currMul 3519
System::__currMul function 3519
System::_roundToInt64 3574
System::_roundToInt64 function 3574
System::+ 3618, 3619
System::+ function 3618, 3619
System::AnsiStringBase 3185
System::AnsiStringBase::!= 3215
System::AnsiStringBase::[] 3216
System::AnsiStringBase::_AnsiCat 3196
System::AnsiStringBase::_AnsiFromPWChar 3196
System::AnsiStringBase::~AnsiStringBase 3189
System::AnsiStringBase::+ 3216
System::AnsiStringBase::< 3216
System::AnsiStringBase::<= 3217
System::AnsiStringBase::= 3217
System::AnsiStringBase::== 3218
System::AnsiStringBase::> 3218
System::AnsiStringBase::>= 3218
System::AnsiStringBase::AnsiCompare 3196
System::AnsiStringBase::AnsiCompareIC 3197
System::AnsiStringBase::AnsiLastChar 3197
System::AnsiStringBase::AnsiPos 3197
System::AnsiStringBase::AnsiStringBase 3190, 3191, 3192
System::AnsiStringBase::BytesOf 3198
System::AnsiStringBase::ByteType 3198
System::AnsiStringBase::cat_vprintf 3199
System::AnsiStringBase::CodePage 3199
System::AnsiStringBase::CurrToStr 3199
System::AnsiStringBase::CurrToStrF 3200
System::AnsiStringBase::Data 3194
System::AnsiStringBase::Delete 3200
System::AnsiStringBase::ElementSize 3200
System::AnsiStringBase::EnsureAnsi 3201
System::AnsiStringBase::FloatToStrF 3201
System::AnsiStringBase::FmtLoadStr 3202
System::AnsiStringBase::Format 3202
System::AnsiStringBase::FormatFloat 3203
System::AnsiStringBase::GetRec 3204
System::AnsiStringBase::Insert 3205
System::AnsiStringBase::IntToHex 3205

System::AnsiStringBase::IsDelimiter 3206
System::AnsiStringBase::IsEmpty 3206
System::AnsiStringBase::IsLeadByte 3206
System::AnsiStringBase::IsPathDelimiter 3206
System::AnsiStringBase::IsTrailByte 3207
System::AnsiStringBase::LastChar 3207
System::AnsiStringBase::LastDelimiter 3207
System::AnsiStringBase::Length 3208
System::AnsiStringBase::LoadStr 3208
System::AnsiStringBase::LoadString 3209
System::AnsiStringBase::LowerCase 3209
System::AnsiStringBase::Pos 3209
System::AnsiStringBase::RefCount 3210
System::AnsiStringBase::SetLength 3210
System::AnsiStringBase::StringOfChar 3210
System::AnsiStringBase::StrRec 3192
System::AnsiStringBase::StrRec structure 3192
System::AnsiStringBase::SubString 3211
System::AnsiStringBase::swap 3211
System::AnsiStringBase::ThrowIfOutOfRange 3211
System::AnsiStringBase::ToDouble 3211
System::AnsiStringBase::ToInt 3212
System::AnsiStringBase::ToIntDef 3212
System::AnsiStringBase::Trim 3212
System::AnsiStringBase::TrimLeft 3212
System::AnsiStringBase::TrimRight 3213
System::AnsiStringBase::TStringFloatFormat 3193
System::AnsiStringBase::TStringFloatFormat enumeration 3193
System::AnsiStringBase::TStringMbcsByteType 3193
System::AnsiStringBase::TStringMbcsByteType enumeration 3193
System::AnsiStringBase::Unique 3213
System::AnsiStringBase::UpperCase 3213
System::AnsiStringBase::vprintf 3214
System::AnsiStringBase::WideChar 3214
System::AnsiStringBase::WideCharBufSize 3214
System::AnsiStringT 3219
System::AnsiStringT::!= 3240
System::AnsiStringT::+ 3240
System::AnsiStringT::+= 3240
System::AnsiStringT::< 3240
System::AnsiStringT::<= 3241
System::AnsiStringT::= 3241
System::AnsiStringT::== 3241
System::AnsiStringT::> 3241
System::AnsiStringT::>= 3241
System::AnsiStringT::AnsiCompare 3229
System::AnsiStringT::AnsiCompareIC 3229
System::AnsiStringT::AnsiPos 3229
System::AnsiStringT::AnsiStringT 3223, 3224, 3225, 3226, 3227, 3228
System::AnsiStringT::c_str 3229
System::AnsiStringT::cat_printf 3230
System::AnsiStringT::cat_sprintf 3230
System::AnsiStringT::cat_vprintf 3230
System::AnsiStringT::CodePage 3230
System::AnsiStringT::CurrToStr 3231
System::AnsiStringT::CurrToStrF 3231
System::AnsiStringT::data 3231
System::AnsiStringT::Delete 3232
System::AnsiStringT::ElementSize 3232
System::AnsiStringT::EnsureAnsi 3232
System::AnsiStringT::FloatToStrF 3233
System::AnsiStringT::FmtLoadStr 3233
System::AnsiStringT::Format 3233
System::AnsiStringT::FormatFloat 3233
System::AnsiStringT::Insert 3234
System::AnsiStringT::IntToHex 3234
System::AnsiStringT::IsDelimiter 3234
System::AnsiStringT::LastDelimiter 3234
System::AnsiStringT::LoadStr 3235
System::AnsiStringT::LoadString 3235
System::AnsiStringT::LowerCase 3235
System::AnsiStringT::Pos 3236
System::AnsiStringT::printf 3236
System::AnsiStringT::RefCount 3236
System::AnsiStringT::SetLength 3236
System::AnsiStringT::sprintf 3237
System::AnsiStringT::StringOfChar 3237
System::AnsiStringT::SubString 3237

System::AnsiStringT::swap 3237
System::AnsiStringT::Trim 3238
System::AnsiStringT::TrimLeft 3238
System::AnsiStringT::TrimRight 3238
System::AnsiStringT::Unique 3238
System::AnsiStringT::UpperCase 3238
System::AnsiStringT::vprintf 3239
System::AnsiStringT::WideChar 3239
System::AnsiStringT::WideCharBufSize 3239
System::CheckSafecallResult 3640
System::CheckSafecallResult function 3640
System::Currency 3242
System::Currency::- 3246
System::Currency::-- 3247
System::Currency::! 3247
System::Currency::!= 3248
System::Currency::% 3248
System::Currency::%= 3249
System::Currency::* 3249
System::Currency::*= 3250
System::Currency::/ 3251
System::Currency::/= 3251
System::Currency::+ 3252
System::Currency::++ 3253
System::Currency::+= 3253
System::Currency::< 3254
System::Currency::<= 3255
System::Currency::= 3256
System::Currency::-= 3256
System::Currency::== 3257
System::Currency::> 3258
System::Currency::>= 3258
System::Currency::Currency 3244
System::Currency::double 3259
System::Currency::int 3259
System::Currency::System::String 3260
System::CurrencyBase 3262
System::CurrencyBase::Val 3263
System::DelphiInterface 3263
System::DelphiInterface::! 3266
System::DelphiInterface::& 3266
System::DelphiInterface::* 3267
System::DelphiInterface::~DelphiInterface 3264
System::DelphiInterface::= 3267
System::DelphiInterface::-> 3268
System::DelphiInterface::DelphiInterface 3265
System::DelphiInterface::Release 3266
System::DelphiInterface::T* 3268
System::DynamicArray 3268
System::DynamicArray::[] 3276
System::DynamicArray::~DynamicArray 3271
System::DynamicArray::= 3276
System::DynamicArray::== 3277
System::DynamicArray::AllocData 3272
System::DynamicArray::Copy 3273
System::DynamicArray::CopyRange 3273
System::DynamicArray::DecRefCount 3274
System::DynamicArray::DimCount 3274
System::DynamicArray::DynamicArray 3271
System::DynamicArray::FreeData 3274
System::DynamicArray::get_high 3274
System::DynamicArray::get_length 3275
System::DynamicArray::get_low 3275
System::DynamicArray::High 3277
System::DynamicArray::IncRefCount 3275
System::DynamicArray::Length 3278
System::DynamicArray::Low 3278
System::DynamicArray::set_length 3275
System::DynamicArray::SetData 3275
System::DynArrayException 3279
System::DynArrayNullData 3279
System::DynArrayOutOfRange 3279
System::DynArrayOutOfRange::DynArrayOutOfRange 3280
System::DynArrayOutOfRange::m_Count 3280
System::DynArrayOutOfRange::m_Index 3281
System::FindDynaInst 3673
System::FindDynaInst function 3673
System::LoadResourceString 3705

System::LoadResourceString function 3705
System::OleVariant 3281
System::OleVariant::!= 3316
System::OleVariant::~OleVariant 3286
System::OleVariant::< 3316, 3317
System::OleVariant::<= 3317
System::OleVariant::= 3319
System::OleVariant::== 3320
System::OleVariant::> 3321
System::OleVariant::>= 3321
System::OleVariant::OleVariant 3288, 3289, 3290, 3291, 3292, 3293, 3294, 3295, 3296
System::OleVariant::TVarData::Reserved1 3298
System::OleVariant::TVarData::Reserved2 3298
System::OleVariant::TVarData::Reserved3 3298
System::OleVariant::TVarData::VArray 3298
System::OleVariant::TVarData::VBoolean 3299
System::OleVariant::TVarData::VByte 3299
System::OleVariant::TVarData::VCurrency 3299
System::OleVariant::TVarData::VDate 3299
System::OleVariant::TVarData::VDispatch 3299
System::OleVariant::TVarData::VDouble 3300
System::OleVariant::TVarData::VError 3300
System::OleVariant::TVarData::VInteger 3300
System::OleVariant::TVarData::VOleStr 3300
System::OleVariant::TVarData::VPointer 3301
System::OleVariant::TVarData::VSingle 3301
System::OleVariant::TVarData::VSmallint 3301
System::OleVariant::TVarData::VType 3301
System::OleVariant::TVarData::VUnknown 3301
System::OleVariant::Variant::- 3302
System::OleVariant::Variant::! 3302
System::OleVariant::Variant::%= 3302
System::OleVariant::Variant::&= 3302
System::OleVariant::Variant::*= 3303
System::OleVariant::Variant::/= 3303
System::OleVariant::Variant::^= 3303
System::OleVariant::Variant::__int64 3303
System::OleVariant::Variant::|= 3303
System::OleVariant::Variant::+= 3304
System::OleVariant::Variant::<<= 3304
System::OleVariant::Variant::-= 3304
System::OleVariant::Variant::>>= 3304
System::OleVariant::Variant::AnsiString 3305
System::OleVariant::Variant::ArrayDimCount 3305
System::OleVariant::Variant::ArrayHighBound 3305
System::OleVariant::Variant::ArrayLock 3305
System::OleVariant::Variant::ArrayLowBound 3305
System::OleVariant::Variant::ArrayRedim 3306
System::OleVariant::Variant::ArrayUnlock 3306
System::OleVariant::Variant::AsType 3306
System::OleVariant::Variant::bool 3306
System::OleVariant::Variant::Byte* 3307
System::OleVariant::Variant::ChangeType 3307
System::OleVariant::Variant::char 3307
System::OleVariant::Variant::Clear 3307
System::OleVariant::Variant::Currency 3307
System::OleVariant::Variant::Currency* 3308
System::OleVariant::Variant::double 3308
System::OleVariant::Variant::double* 3308
System::OleVariant::Variant::float 3308
System::OleVariant::Variant::float* 3309
System::OleVariant::Variant::GetElement 3309
System::OleVariant::Variant::GetError 3309
System::OleVariant::Variant::IDispatch* 3309
System::OleVariant::Variant::IInterface* 3309
System::OleVariant::Variant::int 3310
System::OleVariant::Variant::int* 3310
System::OleVariant::Variant::IsArray 3310
System::OleVariant::Variant::IsEmpty 3310
System::OleVariant::Variant::IsNull 3311
System::OleVariant::Variant::IUnknown* 3311
System::OleVariant::Variant::long 3311
System::OleVariant::Variant::long* 3311
System::OleVariant::Variant::PutElement 3311
System::OleVariant::Variant::SetError 3312
System::OleVariant::Variant::short 3312
System::OleVariant::Variant::short* 3312
System::OleVariant::Variant::signed char 3312

System::OleVariant::Variant::signed char* 3313
System::OleVariant::Variant::TDateTime 3313
System::OleVariant::Variant::TDateTime* 3313
System::OleVariant::Variant::Type 3313
System::OleVariant::Variant::UnicodeString 3313
System::OleVariant::Variant::unsigned char 3314
System::OleVariant::Variant::unsigned int 3314
System::OleVariant::Variant::unsigned int* 3314
System::OleVariant::Variant::unsigned long 3314
System::OleVariant::Variant::unsigned short 3315
System::OleVariant::Variant::unsigned short* 3315
System::OleVariant::Variant::WideString 3315
System::PCallDesc 3830
System::PCallDesc type 3830
System::PDispDesc 3833
System::PDispDesc type 3833
System::PMemoryManager 3840
System::PMemoryManager type 3840
System::PMemoryManagerEx 3840
System::PMemoryManagerEx type 3840
System::PModuleUnloadRec 3841
System::PModuleUnloadRec type 3841
System::PResStringRec 3845
System::PResStringRec type 3845
System::PSmallBlockTypeState 3847
System::PSmallBlockTypeState type 3847
System::PVariantManager 3852
System::PVariantManager type 3852
System::PVarRec 3852
System::PVarRec type 3852
System::ResourceString 3854
System::ResourceString type 3854
System::SmallString 3322
System::SmallString::[] 3324
System::SmallString::= 3324
System::SmallString::AnsiString 3324
System::SmallString::SmallString 3323
System::SmallStringBase 3324
System::SmallStringBase::Data 3325
System::SmallStringBase::Len 3325
System::sPointer 3326
System::sPointer::& 3327
System::sPointer::~sPointer 3326
System::sPointer::-> 3328
System::sPointer::sPointer 3327
System::TAggregatedObject 3328
System::TAggregatedObject::_AddRef 3330
System::TAggregatedObject::_Release 3330
System::TAggregatedObject::Controller 3331
System::TAggregatedObject::Create 3329
System::TAggregatedObject::QueryInterface 3330
System::TBooleanDynArray 3855
System::TBooleanDynArray type 3855
System::TByteDynArray 3856
System::TByteDynArray type 3856
System::TCallDesc 3820
System::TCallDesc structure 3820
System::TCardinalDynArray 3856
System::TCardinalDynArray type 3856
System::TContainedObject 3333
System::TContainedObject::QueryInterface 3335
System::TDateTime 3335
System::TDateTime::- 3347
System::TDateTime::-- 3347
System::TDateTime::!= 3348
System::TDateTime::+ 3348
System::TDateTime::++ 3349
System::TDateTime::+= 3349
System::TDateTime::< 3350
System::TDateTime::<= 3350
System::TDateTime::= 3351
System::TDateTime::-= 3352
System::TDateTime::== 3352
System::TDateTime::> 3352
System::TDateTime::>= 3353
System::TDateTime::CurrentDate 3340
System::TDateTime::CurrentDateTime 3341
System::TDateTime::CurrentTime 3341

System::TDateTime::DateString 3342
System::TDateTime::DateTimeString 3342
System::TDateTime::DayOfWeek 3342
System::TDateTime::DecodeDate 3343
System::TDateTime::DecodeTime 3343
System::TDateTime::double 3353
System::TDateTime::FileDate 3343
System::TDateTime::FileDateToDateTime 3344
System::TDateTime::FormatString 3344
System::TDateTime::int 3353
System::TDateTime::System::String 3353
System::TDateTime::TDateTime 3338
System::TDateTime::TDateTimeFlag 3340
System::TDateTime::TDateTimeFlag enumeration 3340
System::TDateTime::TimeString 3345
System::TDateTimeBase 3354
System::TDateTimeBase::Val 3354
System::TDispDesc 3820
System::TDispDesc structure 3820
System::TDoubleDynArray 3859
System::TDoubleDynArray type 3859
System::THeapBlock 3802
System::THeapBlock structure 3802
System::THeapStatus 3802
System::THeapStatus record 3802
System::TInt64DynArray 3861
System::TInt64DynArray type 3861
System::TIntegerDynArray 3861
System::TIntegerDynArray type 3861
System::TInterfacedObject 3355
System::TInterfacedObject::_AddRef 3357
System::TInterfacedObject::_Release 3357
System::TInterfacedObject::AfterConstruction 3358
System::TInterfacedObject::BeforeDestruction 3358
System::TInterfacedObject::FRefCount 3356
System::TInterfacedObject::NewInstance 3358
System::TInterfacedObject::QueryInterface 3359
System::TInterfacedObject::RefCount 3360
System::TLibModule 3804
System::TLibModule record 3804
System::TLongWordDynArray 3862
System::TLongWordDynArray type 3862
System::TMemoryManager 3805
System::TMemoryManager record 3805
System::TMemoryManagerEx 3805
System::TMemoryManagerEx record 3805
System::TMemoryManagerState 3806
System::TMemoryManagerState record 3806
System::TMetaClass 3360
System::TModuleUnloadRec 3808
System::TModuleUnloadRec record 3808
System::TObject 3364
System::TObject::AfterConstruction 3367
System::TObject::BeforeDestruction 3368
System::TObject::ClassInfo 3368
System::TObject::ClassName 3369
System::TObject::ClassNameIs 3371
System::TObject::ClassParent 3372
System::TObject::ClassType 3374
System::TObject::CleanupInstance 3377
System::TObject::Create 3366
System::TObject::DefaultHandler 3377
System::TObject::Destroy 3366
System::TObject::Dispatch 3379
System::TObject::Equals 3381
System::TObject::FieldAddress 3382
System::TObject::Free 3382
System::TObject::FreeInstance 3386
System::TObject::GetHashCode 3386
System::TObject::GetInterface 3387
System::TObject::GetInterfaceEntry 3387
System::TObject::GetInterfaceTable 3388
System::TObject::InheritsFrom 3388
System::TObject::InitInstance 3389
System::TObject::InstanceSize 3389
System::TObject::MethodAddress 3390
System::TObject::MethodName 3390
System::TObject::NewInstance 3391

System::TObject::SafeCallException 3391
System::TObject::ToString 3391
System::TObject::UnitName 3392
System::TResStringRec 3809
System::TResStringRec record 3809
System::TShortIntDynArray 3864
System::TShortIntDynArray type 3864
System::TSingleDynArray 3865
System::TSingleDynArray type 3865
System::TSmallBlockTypeState 3810
System::TSmallBlockTypeState record 3810
System::TSmallBlockTypeStates 3865
System::TSmallBlockTypeStates type 3865
System::TSmallIntDynArray 3865
System::TSmallIntDynArray type 3865
System::TStringDynArray 3866
System::TStringDynArray type 3866
System::TThreadFunc 3867
System::TThreadFunc type 3867
System::TVarArray 3812
System::TVarArray record 3812
System::TVarArrayBound 3813
System::TVarArrayBound record 3813
System::TVariantManager 3817
System::TVariantManager record 3817
System::TVarRec 3819
System::TVarRec record 3819
System::TWideStringDynArray 3870
System::TWideStringDynArray type 3870
System::TWordDynArray 3870
System::TWordDynArray type 3870
System::UnicodeString 3393
System::UnicodeString::!= 3423
System::UnicodeString::[] 3424
System::UnicodeString::~UnicodeString 3396
System::UnicodeString::+ 3424
System::UnicodeString::+= 3425
System::UnicodeString::< 3426
System::UnicodeString::<= 3426
System::UnicodeString::= 3427
System::UnicodeString::== 3427
System::UnicodeString::> 3428
System::UnicodeString::>= 3428
System::UnicodeString::BytesOf 3401
System::UnicodeString::ByteType 3401
System::UnicodeString::c_str 3402
System::UnicodeString::cat_printf 3402
System::UnicodeString::cat_sprintf 3403
System::UnicodeString::cat_vprintf 3403
System::UnicodeString::CodePage 3404
System::UnicodeString::Compare 3404
System::UnicodeString::CompareIC 3405
System::UnicodeString::CurrToStr 3405
System::UnicodeString::CurrToStrF 3405
System::UnicodeString::data 3406
System::UnicodeString::Delete 3406
System::UnicodeString::ElementSize 3407
System::UnicodeString::EnsureUnicode 3407
System::UnicodeString::FloatToStrF 3407
System::UnicodeString::FmtLoadStr 3408
System::UnicodeString::Format 3409
System::UnicodeString::FormatFloat 3409
System::UnicodeString::GetRec 3411
System::UnicodeString::Insert 3411
System::UnicodeString::IntToHex 3412
System::UnicodeString::IsDelimiter 3412
System::UnicodeString::IsEmpty 3412
System::UnicodeString::IsLeadSurrogate 3412
System::UnicodeString::IsPathDelimiter 3413
System::UnicodeString::IsTrailSurrogate 3413
System::UnicodeString::LastChar 3413
System::UnicodeString::LastDelimiter 3413
System::UnicodeString::Length 3414
System::UnicodeString::LoadStr 3414
System::UnicodeString::LoadString 3415
System::UnicodeString::LowerCase 3415
System::UnicodeString::Pos 3416
System::UnicodeString::printf 3416

System::UnicodeString::RefCount 3416
System::UnicodeString::SetLength 3417
System::UnicodeString::sprintf 3417
System::UnicodeString::StringOfChar 3417
System::UnicodeString::StrRec 3398
System::UnicodeString::StrRec structure 3398
System::UnicodeString::SubString 3418
System::UnicodeString::swap 3418
System::UnicodeString::t_str 3418
System::UnicodeString::ThrowIfOutOfRange 3419
System::UnicodeString::ToDouble 3419
System::UnicodeString::ToInt 3419
System::UnicodeString::ToIntDef 3420
System::UnicodeString::Trim 3420
System::UnicodeString::TrimLeft 3420
System::UnicodeString::TrimRight 3421
System::UnicodeString::TStringFloatFormat 3399
System::UnicodeString::TStringFloatFormat enumeration 3399
System::UnicodeString::TStringLeadCharType 3399
System::UnicodeString::TStringLeadCharType enumeration 3399
System::UnicodeString::UnicodeString 3397
System::UnicodeString::Unique 3421
System::UnicodeString::UpperCase 3421
System::UnicodeString::vprintf 3422
System::UnicodeString::w_str 3422
System::USE 3960
System::USE macro 3960
System::USEADDITIONALFILES 3960
System::USEADDITIONALFILES macro 3960
System::USEASM 3961
System::USEASM macro 3961
System::USEDATAMODULE 3961
System::USEDATAMODULE macro 3961
System::USEDATAMODULENS 3961
System::USEDATAMODULENS macro 3961
System::USEDEF 3962
System::USEDEF macro 3962
System::USEFILE 3962
System::USEFILE macro 3962
System::USEFORM 3963
System::USEFORM macro 3963
System::USEFORMNS 3963
System::USEFORMNS macro 3963
System::USEFORMRES 3963
System::USEFORMRES macro 3963
System::USEIDL 3964
System::USEIDL macro 3964
System::USELIB 3964
System::USELIB macro 3964
System::USEOBJ 3964
System::USEOBJ macro 3964
System::USEPACKAGE 3965
System::USEPACKAGE macro 3965
System::USERC 3965
System::USERC macro 3965
System::USERES 3965
System::USERES macro 3965
System::USETLB 3966
System::USETLB macro 3966
System::USEUNIT 3966
System::USEUNIT macro 3966
System::Variant 3430
System::Variant::- 3446, 3447
System::Variant::! 3447
System::Variant::!= 3448
System::Variant::% 3449
System::Variant::%= 3449
System::Variant::& 3450
System::Variant::&= 3450
System::Variant::* 3451
System::Variant::*= 3451
System::Variant::/ 3452
System::Variant::/= 3452
System::Variant::^ 3453
System::Variant::^= 3453
System::Variant::__int64 3454
System::Variant::| 3454
System::Variant::|= 3454

System::Variant::~Variant 3433
System::Variant::+ 3455
System::Variant::+= 3455
System::Variant::< 3456
System::Variant::<< 3457
System::Variant::<<= 3458
System::Variant::<= 3458
System::Variant::= 3461
System::Variant::-= 3463
System::Variant::== 3463
System::Variant::> 3464, 3465
System::Variant::>= 3466
System::Variant::>> 3467
System::Variant::>>= 3468
System::Variant::AnsiString 3468
System::Variant::ArrayDimCount 3437
System::Variant::ArrayHighBound 3438
System::Variant::ArrayLock 3438
System::Variant::ArrayLowBound 3438
System::Variant::ArrayRedim 3439
System::Variant::ArrayUnlock 3439
System::Variant::AsType 3439
System::Variant::bool 3468
System::Variant::ChangeType 3439
System::Variant::char 3468
System::Variant::Clear 3440
System::Variant::Currency 3469
System::Variant::Currency* 3469
System::Variant::double 3469
System::Variant::double* 3470
System::Variant::Empty 3440
System::Variant::float 3470
System::Variant::float* 3470
System::Variant::GetBaseVariant 3440
System::Variant::GetElement 3441
System::Variant::GetError 3441
System::Variant::IDispatch* 3470
System::Variant::IInterface* 3471
System::Variant::int 3471
System::Variant::int* 3471
System::Variant::IsArray 3442
System::Variant::IsEmpty 3442
System::Variant::IsNull 3442
System::Variant::IUnknown* 3471
System::Variant::long 3472
System::Variant::long* 3472
System::Variant::NoParam 3443
System::Variant::PutElement 3443
System::Variant::Sanitized 3444
System::Variant::SetError 3444
System::Variant::short 3472
System::Variant::short* 3472
System::Variant::signed char 3473
System::Variant::signed char* 3473
System::Variant::TDateTime 3473
System::Variant::TDateTime* 3473
System::Variant::Type 3444
System::Variant::UnicodeString 3474
System::Variant::unsigned char 3474
System::Variant::unsigned char* 3474
System::Variant::unsigned int 3474
System::Variant::unsigned int* 3474
System::Variant::unsigned long 3475
System::Variant::unsigned long* 3475
System::Variant::unsigned short 3475
System::Variant::unsigned short* 3475
System::Variant::Variant 3435
System::Variant::WideString 3476
System::WideString 3476
System::WideString::!= 3491
System::WideString::[] 3492
System::WideString::~WideString 3478
System::WideString::+ 3492
System::WideString::+= 3493
System::WideString::< 3493
System::WideString::<= 3494
System::WideString::= 3494
System::WideString::== 3495

System::WideString::> 3495
System::WideString::>= 3496
System::WideString::c_bstr 3481
System::WideString::cat_printf 3482
System::WideString::cat_sprintf 3482
System::WideString::cat_vprintf 3482
System::WideString::Copy 3482
System::WideString::CurrToStr 3483
System::WideString::CurrToStrF 3483
System::WideString::data 3483
System::WideString::Delete 3484
System::WideString::FloatToStrF 3484
System::WideString::Format 3484
System::WideString::FormatFloat 3484
System::WideString::Insert 3485
System::WideString::IntToHex 3485
System::WideString::IsDelimiter 3485
System::WideString::IsEmpty 3485
System::WideString::IsPathDelimiter 3486
System::WideString::LastChar 3486
System::WideString::LastDelimiter 3486
System::WideString::Length 3486
System::WideString::LowerCase 3487
System::WideString::Pos 3487
System::WideString::printf 3487
System::WideString::SetLength 3487
System::WideString::sprintf 3488
System::WideString::StringOfChar 3488
System::WideString::SubString 3488
System::WideString::swap 3489
System::WideString::ToDouble 3489
System::WideString::ToInt 3489
System::WideString::ToIntDef 3489
System::WideString::Trim 3490
System::WideString::TrimLeft 3490
System::WideString::TrimRight 3490
System::WideString::TStringFloatFormat 3480
System::WideString::TStringFloatFormat enumeration 3480
System::WideString::UpperCase 3490
System::WideString::vprintf 3491
System::WideString::WideString 3479
SysUtils 3967
SysUtils namespace 3967
    Classes 3981
    Constants 4408
    Functions 4103
    Interfaces 4101
    Structs, Records, Enums 4361
    Types 4377
    Variables 4387
SysUtils.Abort 4111
SysUtils.Abort function 4111
SysUtils.AddExitProc 4112
SysUtils.AddExitProc function 4112
SysUtils.AddTerminateProc 4112
SysUtils.AddTerminateProc function 4112
SysUtils.AdjustLineBreaks 4113
SysUtils.AdjustLineBreaks function 4113
SysUtils.AnsiCompareFileName 4113
SysUtils.AnsiCompareFileName function 4113
SysUtils.AnsiCompareStr 4114
SysUtils.AnsiCompareStr function 4114
SysUtils.AnsiCompareText 4115
SysUtils.AnsiCompareText function 4115
SysUtils.AnsiDequotedStr 4115
SysUtils.AnsiDequotedStr function 4115
SysUtils.AnsiExtractQuotedStr 4116
SysUtils.AnsiExtractQuotedStr function 4116
SysUtils.AnsiLastChar 4116
SysUtils.AnsiLastChar function 4116
SysUtils.AnsiLowerCase 4117
SysUtils.AnsiLowerCase function 4117
SysUtils.AnsiLowerCaseFileName 4117
SysUtils.AnsiLowerCaseFileName function 4117
SysUtils.AnsiPos 4118
SysUtils.AnsiPos function 4118
SysUtils.AnsiQuotedStr 4118
SysUtils.AnsiQuotedStr function 4118

SysUtils.AnsiSameStr 4119
SysUtils.AnsiSameStr function 4119
SysUtils.AnsiSameText 4119
SysUtils.AnsiSameText function 4119
SysUtils.AnsiStrAlloc 4120
SysUtils.AnsiStrAlloc function 4120
SysUtils.AnsiStrComp 4120
SysUtils.AnsiStrComp function 4120
SysUtils.AnsiStrIComp 4121
SysUtils.AnsiStrIComp function 4121
SysUtils.AnsiStrLastChar 4121
SysUtils.AnsiStrLastChar function 4121
SysUtils.AnsiStrLComp 4122
SysUtils.AnsiStrLComp function 4122
SysUtils.AnsiStrLIComp 4122
SysUtils.AnsiStrLIComp function 4122
SysUtils.AnsiStrLower 4123
SysUtils.AnsiStrLower function 4123
SysUtils.AnsiStrPos 4124
SysUtils.AnsiStrPos function 4124
SysUtils.AnsiStrRScan 4124
SysUtils.AnsiStrRScan function 4124
SysUtils.AnsiStrScan 4125
SysUtils.AnsiStrScan function 4125
SysUtils.AnsiStrUpper 4125
SysUtils.AnsiStrUpper function 4125
SysUtils.AnsiUpperCase 4126
SysUtils.AnsiUpperCase function 4126
SysUtils.AnsiUpperCaseFileName 4126
SysUtils.AnsiUpperCaseFileName function 4126
SysUtils.AppendStr 4127
SysUtils.AppendStr function 4127
SysUtils.AssignStr 4127
SysUtils.AssignStr function 4127
SysUtils.Beep 4127
SysUtils.Beep function 4127
SysUtils.BoolToStr 4130
SysUtils.BoolToStr function 4130
SysUtils.ByteLength 4132
SysUtils.ByteLength function 4132
SysUtils.BytesOf 4132, 4133
SysUtils.BytesOf function 4132, 4133
SysUtils.ByteToCharIndex 4134
SysUtils.ByteToCharIndex function 4134
SysUtils.ByteToCharLen 4134
SysUtils.ByteToCharLen function 4134
SysUtils.ByteType 4135
SysUtils.ByteType function 4135
SysUtils.CallTerminateProcs 4136
SysUtils.CallTerminateProcs function 4136
SysUtils.ChangeFileExt 4136
SysUtils.ChangeFileExt function 4136
SysUtils.ChangeFilePath 4137
SysUtils.ChangeFilePath function 4137
SysUtils.CharInSet 4138
SysUtils.CharInSet function 4138
SysUtils.CharLength 4139
SysUtils.CharLength function 4139
SysUtils.CharToByteIndex 4139
SysUtils.CharToByteIndex function 4139
SysUtils.CharToByteLen 4140
SysUtils.CharToByteLen function 4140
SysUtils.CharToElementIndex 4140
SysUtils.CharToElementIndex function 4140
SysUtils.CharToElementLen 4141
SysUtils.CharToElementLen function 4141
SysUtils.CheckWin32Version 4141
SysUtils.CheckWin32Version function 4141
SysUtils.CompareMem 4142
SysUtils.CompareMem function 4142
SysUtils.CompareStr 4142
SysUtils.CompareStr function 4142
SysUtils.CompareText 4143
SysUtils.CompareText function 4143
SysUtils.CreateDir 4147
SysUtils.CreateDir function 4147
SysUtils.CreateGUID 4148
SysUtils.CreateGUID function 4148

SysUtils.CurrencyDecimals 4388
SysUtils.CurrencyDecimals variable 4388
SysUtils.CurrencyFormat 4388
SysUtils.CurrencyFormat variable 4388
SysUtils.CurrencyString 4389
SysUtils.CurrencyString variable 4389
SysUtils.CurrentYear 4150
SysUtils.CurrentYear function 4150
SysUtils.CurrToStr 4150
SysUtils.CurrToStr function 4150
SysUtils.CurrToStrF 4151
SysUtils.CurrToStrF function 4151
SysUtils.Date 4151
SysUtils.Date function 4151
SysUtils.DateDelta 4409
SysUtils.DateDelta constant 4409
SysUtils.DateSeparator 4389
SysUtils.DateSeparator variable 4389
SysUtils.DateTimeToFileDate 4152
SysUtils.DateTimeToFileDate function 4152
SysUtils.DateTimeToStr 4153
SysUtils.DateTimeToStr function 4153
SysUtils.DateTimeToString 4153
SysUtils.DateTimeToString function 4153
SysUtils.DateTimeToSystemTime 4155
SysUtils.DateTimeToSystemTime function 4155
SysUtils.DateTimeToTimeStamp 4156
SysUtils.DateTimeToTimeStamp function 4156
SysUtils.DateToStr 4156
SysUtils.DateToStr function 4156
SysUtils.DayOfWeek 4158
SysUtils.DayOfWeek function 4158
SysUtils.DecimalSeparator 4390
SysUtils.DecimalSeparator variable 4390
SysUtils.DecodeDate 4159
SysUtils.DecodeDate function 4159
SysUtils.DecodeDateFully 4160
SysUtils.DecodeDateFully function 4160
SysUtils.DecodeTime 4161
SysUtils.DecodeTime function 4161
SysUtils.DefaultFalseBoolStr 4410
SysUtils.DefaultFalseBoolStr constant 4410
SysUtils.DefaultTrueBoolStr 4410
SysUtils.DefaultTrueBoolStr constant 4410
SysUtils.DelegatesEqual 4162
SysUtils.DelegatesEqual function 4162
SysUtils.DeleteFile 4162
SysUtils.DeleteFile function 4162
SysUtils.DirectoryExists 4164
SysUtils.DirectoryExists function 4164
SysUtils.DiskFree 4165
SysUtils.DiskFree function 4165
SysUtils.DiskSize 4166
SysUtils.DiskSize function 4166
SysUtils.DisposeStr 4167
SysUtils.DisposeStr function 4167
SysUtils.DriveDelim 4411
SysUtils.DriveDelim constant 4411
SysUtils.EAbort 3982
SysUtils.EAbstractError 3982
SysUtils.EAccessViolation 3983
SysUtils.EArgumentException 3984
SysUtils.EArgumentOutOfRangeException 3984
SysUtils.EAssertionFailed 3984
SysUtils.EControlC 3985
SysUtils.EConvertError 3985
SysUtils.EDivByZero 3987
SysUtils.EEncodingError 3988
SysUtils.EExternal 3988
SysUtils.EExternal.ExceptionRecord 3991
SysUtils.EExternalException 3991
SysUtils.EHeapException 3992
SysUtils.EHeapException.FreeInstance 3994
SysUtils.EHeapException.RaisingException 3995
SysUtils.EInOutError 3995
SysUtils.EInOutError.ErrorCode 3998
SysUtils.EIntError 3999
SysUtils.EIntfCastError 3999

SysUtils.EIntOverflow 4000
SysUtils.EInvalidCast 4000
SysUtils.EInvalidContainer 4001
SysUtils.EInvalidInsert 4001
SysUtils.EInvalidOp 4002
SysUtils.EInvalidPointer 4002
SysUtils.ElementToCharIndex 4168
SysUtils.ElementToCharIndex function 4168
SysUtils.ElementToCharLen 4168, 4169
SysUtils.ElementToCharLen function 4168, 4169
SysUtils.EMathError 4003
SysUtils.EMonitor 4004
SysUtils.EMonitorLockException 4004
SysUtils.EmptyAnsiStr 4391
SysUtils.EmptyAnsiStr variable 4391
SysUtils.EmptyStr 4391
SysUtils.EmptyStr variable 4391
SysUtils.EmptyWideStr 4391
SysUtils.EmptyWideStr variable 4391
SysUtils.EncodeDate 4169
SysUtils.EncodeDate function 4169
SysUtils.EncodeTime 4170
SysUtils.EncodeTime function 4170
SysUtils.ENoConstructException 4004
SysUtils.ENoMonitorSupportException 4005
SysUtils.EOSError 4005
SysUtils.EOSError.ErrorCode 4007
SysUtils.EOutOfMemory 4008
SysUtils.EOverflow 4008
SysUtils.EPackageError 4009
SysUtils.EPrivilege 4009
SysUtils.EProgrammerNotFound 4010
SysUtils.EPropReadOnly 4010
SysUtils.EPropWriteOnly 4011
SysUtils.EraNames 4392
SysUtils.EraNames variable 4392
SysUtils.ERangeError 4011
SysUtils.EraYearOffsets 4392
SysUtils.EraYearOffsets variable 4392
SysUtils.ESafecallException 4012
SysUtils.EStackOverflow 4013
SysUtils.EUnderflow 4013
SysUtils.EVariantError 4014
SysUtils.EWin32Error 4015
SysUtils.ExceptClass 4377
SysUtils.ExceptClass type 4377
SysUtils.Exception 4015
SysUtils.Exception.BaseException 4030
SysUtils.Exception.CleanUpStackInfoProc 4026
SysUtils.Exception.Create 4018
SysUtils.Exception.CreateFmt 4020
SysUtils.Exception.CreateFmtHelp 4021
SysUtils.Exception.CreateHelp 4021
SysUtils.Exception.CreateRes 4022
SysUtils.Exception.CreateResFmt 4023
SysUtils.Exception.CreateResFmtHelp 4024
SysUtils.Exception.CreateResHelp 4025
SysUtils.Exception.Destroy 4026
SysUtils.Exception.GetBaseException 4027
SysUtils.Exception.GetExceptionStackInfoProc 4027
SysUtils.Exception.GetStackInfoStringProc 4027
SysUtils.Exception.GetStackTrace 4028
SysUtils.Exception.HelpContext 4030
SysUtils.Exception.InnerException 4031
SysUtils.Exception.Message 4031
SysUtils.Exception.RaiseOuterException 4028
SysUtils.Exception.RaisingException 4028
SysUtils.Exception.SetInnerException 4028
SysUtils.Exception.SetStackInfo 4029
SysUtils.Exception.StackInfo 4031
SysUtils.Exception.StackTrace 4032
SysUtils.Exception.ThrowOuterException 4029
SysUtils.Exception.ToString 4029
SysUtils.ExceptionErrorMessage 4171
SysUtils.ExceptionErrorMessage function 4171
SysUtils.ExcludeTrailingBackslash 4172
SysUtils.ExcludeTrailingBackslash function 4172
SysUtils.ExcludeTrailingPathDelimiter 4173

SysUtils.ExcludeTrailingPathDelimiter function 4173
SysUtils.ExpandFileName 4173
SysUtils.ExpandFileName function 4173
SysUtils.ExpandFileNameCase 4174
SysUtils.ExpandFileNameCase function 4174
SysUtils.ExpandUNCFileName 4175
SysUtils.ExpandUNCFileName function 4175
SysUtils.ExtractFileDir 4176
SysUtils.ExtractFileDir function 4176
SysUtils.ExtractFileDrive 4176
SysUtils.ExtractFileDrive function 4176
SysUtils.ExtractFileExt 4177
SysUtils.ExtractFileExt function 4177
SysUtils.ExtractFileName 4177
SysUtils.ExtractFileName function 4177
SysUtils.ExtractFilePath 4179
SysUtils.ExtractFilePath function 4179
SysUtils.ExtractRelativePath 4181
SysUtils.ExtractRelativePath function 4181
SysUtils.ExtractShortPathName 4182
SysUtils.ExtractShortPathName function 4182
SysUtils.EZeroDivide 4032
SysUtils.faAnyFile 4411
SysUtils.faAnyFile constant 4411
SysUtils.faArchive 4412
SysUtils.faArchive constant 4412
SysUtils.faDirectory 4412
SysUtils.faDirectory constant 4412
SysUtils.faHidden 4413
SysUtils.faHidden constant 4413
SysUtils.FalseBoolStrs 4392
SysUtils.FalseBoolStrs variable 4392
SysUtils.faNormal 4414
SysUtils.faNormal constant 4414
SysUtils.faReadOnly 4414
SysUtils.faReadOnly constant 4414
SysUtils.faSymLink 4415
SysUtils.faSymLink constant 4415
SysUtils.faSysFile 4415
SysUtils.faSysFile constant 4415
SysUtils.faTemporary 4416
SysUtils.faTemporary constant 4416
SysUtils.faVolumeID 4416
SysUtils.faVolumeID constant 4416
SysUtils.FileAge 4183
SysUtils.FileAge function 4183
SysUtils.FileClose 4185
SysUtils.FileClose function 4185
SysUtils.FileCreate 4188
SysUtils.FileCreate function 4188
SysUtils.FileDateToDateTime 4191
SysUtils.FileDateToDateTime function 4191
SysUtils.FileExists 4193
SysUtils.FileExists function 4193
SysUtils.FileGetAttr 4197
SysUtils.FileGetAttr function 4197
SysUtils.FileGetDate 4199
SysUtils.FileGetDate function 4199
SysUtils.FileIsReadOnly 4200
SysUtils.FileIsReadOnly function 4200
SysUtils.FileOpen 4200
SysUtils.FileOpen function 4200
SysUtils.FileRead 4204
SysUtils.FileRead function 4204
SysUtils.FileSearch 4206
SysUtils.FileSearch function 4206
SysUtils.FileSeek 4207
SysUtils.FileSeek function 4207
SysUtils.FileSetAttr 4209
SysUtils.FileSetAttr function 4209
SysUtils.FileSetDate 4212
SysUtils.FileSetDate function 4212
SysUtils.FileSetReadOnly 4212
SysUtils.FileSetReadOnly function 4212
SysUtils.FileWrite 4213
SysUtils.FileWrite function 4213
SysUtils.FinalizePackage 4216
SysUtils.FinalizePackage function 4216

SysUtils.FindClose 4216
SysUtils.FindClose function 4216
SysUtils.FindCmdLineSwitch 4218
SysUtils.FindCmdLineSwitch function 4218
SysUtils.FindFirst 4218
SysUtils.FindFirst function 4218
SysUtils.FindNext 4221
SysUtils.FindNext function 4221
SysUtils.FloatToCurr 4223
SysUtils.FloatToCurr function 4223
SysUtils.FloatToDateTime 4223
SysUtils.FloatToDateTime function 4223
SysUtils.FloatToDecimal 4224
SysUtils.FloatToDecimal function 4224
SysUtils.FloatToStr 4224
SysUtils.FloatToStr function 4224
SysUtils.FloatToStrF 4226
SysUtils.FloatToStrF function 4226
SysUtils.FloatToText 4227
SysUtils.FloatToText function 4227
SysUtils.FloatToTextFmt 4228
SysUtils.FloatToTextFmt function 4228
SysUtils.fmOpenRead 4417
SysUtils.fmOpenRead constant 4417
SysUtils.fmOpenReadWrite 4417
SysUtils.fmOpenReadWrite constant 4417
SysUtils.fmOpenWrite 4418
SysUtils.fmOpenWrite constant 4418
SysUtils.fmShareCompat 4419
SysUtils.fmShareCompat constant 4419
SysUtils.fmShareDenyNone 4419
SysUtils.fmShareDenyNone constant 4419
SysUtils.fmShareDenyRead 4419
SysUtils.fmShareDenyRead constant 4419
SysUtils.fmShareDenyWrite 4420
SysUtils.fmShareDenyWrite constant 4420
SysUtils.fmShareExclusive 4420
SysUtils.fmShareExclusive constant 4420
SysUtils.FmtLoadStr 4229
SysUtils.FmtLoadStr function 4229
SysUtils.FmtStr 4229
SysUtils.FmtStr function 4229
SysUtils.ForceDirectories 4230
SysUtils.ForceDirectories function 4230
SysUtils.Format 4232
SysUtils.Format function 4232
SysUtils.FormatBuf 4237
SysUtils.FormatBuf function 4237
SysUtils.FormatCurr 4238
SysUtils.FormatCurr function 4238
SysUtils.FormatDateTime 4239
SysUtils.FormatDateTime function 4239
SysUtils.FormatFloat 4241
SysUtils.FormatFloat function 4241
SysUtils.FreeAndNil 4245
SysUtils.FreeAndNil function 4245
SysUtils.GDAL 4246
SysUtils.GDAL function 4246
SysUtils.GetCurrentDir 4246
SysUtils.GetCurrentDir function 4246
SysUtils.GetDiskFreeSpaceEx 4393
SysUtils.GetDiskFreeSpaceEx variable 4393
SysUtils.GetEnvironmentVariable 4248
SysUtils.GetEnvironmentVariable function 4248
SysUtils.GetFileVersion 4249
SysUtils.GetFileVersion function 4249
SysUtils.GetFormatSettings 4250
SysUtils.GetFormatSettings function 4250
SysUtils.GetLocaleChar 4250
SysUtils.GetLocaleChar function 4250
SysUtils.GetLocaleFormatSettings 4250
SysUtils.GetLocaleFormatSettings function 4250
SysUtils.GetLocaleStr 4251
SysUtils.GetLocaleStr function 4251
SysUtils.GetModuleName 4251
SysUtils.GetModuleName function 4251
SysUtils.GetPackageDescription 4252
SysUtils.GetPackageDescription function 4252

SysUtils.GetPackageInfo 4253
SysUtils.GetPackageInfo function 4253
SysUtils.GetTime 4253
SysUtils.GetTime function 4253
SysUtils.GUIDToString 4254
SysUtils.GUIDToString function 4254
SysUtils.HashName 4256
SysUtils.HashName function 4256
SysUtils.HexDisplayPrefix 4393
SysUtils.HexDisplayPrefix variable 4393
SysUtils.HoursPerDay 4420
SysUtils.HoursPerDay constant 4420
SysUtils.IncAMonth 4256
SysUtils.IncAMonth function 4256
SysUtils.IncludeTrailingBackslash 4257
SysUtils.IncludeTrailingBackslash function 4257
SysUtils.IncludeTrailingPathDelimiter 4257
SysUtils.IncludeTrailingPathDelimiter function 4257
SysUtils.IncMonth 4258
SysUtils.IncMonth function 4258
SysUtils.InitializePackage 4259
SysUtils.InitializePackage function 4259
SysUtils.Int64Rec 4361
SysUtils.Int64Rec record 4361
SysUtils.IntToHex 4259
SysUtils.IntToHex function 4259
SysUtils.IntToStr 4260
SysUtils.IntToStr function 4260
SysUtils.IReadWriteSync 4101
SysUtils.IReadWriteSync.BeginRead 4102
SysUtils.IReadWriteSync.BeginWrite 4102
SysUtils.IReadWriteSync.EndRead 4102
SysUtils.IReadWriteSync.EndWrite 4102
SysUtils.IsAssembly 4261
SysUtils.IsAssembly function 4261
SysUtils.IsDelimiter 4262
SysUtils.IsDelimiter function 4262
SysUtils.IsEqualGUID 4262
SysUtils.IsEqualGUID function 4262
SysUtils.IsLeadChar 4264
SysUtils.IsLeadChar function 4264
SysUtils.IsLeapYear 4264
SysUtils.IsLeapYear function 4264
SysUtils.IsPathDelimiter 4266
SysUtils.IsPathDelimiter function 4266
SysUtils.IsValidIdent 4266
SysUtils.IsValidIdent function 4266
SysUtils.Languages 4267
SysUtils.Languages function 4267
SysUtils.LastDelimiter 4267
SysUtils.LastDelimiter function 4267
SysUtils.LeadBytes 4394
SysUtils.LeadBytes variable 4394
SysUtils.ListSeparator 4394
SysUtils.ListSeparator variable 4394
SysUtils.LoadPackage 4268
SysUtils.LoadPackage function 4268
SysUtils.LoadStr 4268
SysUtils.LoadStr function 4268
SysUtils.LongDateFormat 4394
SysUtils.LongDateFormat variable 4394
SysUtils.LongDayNames 4395
SysUtils.LongDayNames variable 4395
SysUtils.LongMonthNames 4395
SysUtils.LongMonthNames variable 4395
SysUtils.LongRec 4362
SysUtils.LongRec record 4362
SysUtils.LongTimeFormat 4396
SysUtils.LongTimeFormat variable 4396
SysUtils.LowerCase 4269
SysUtils.LowerCase function 4269
SysUtils.MaxCurrency 4421
SysUtils.MaxCurrency constant 4421
SysUtils.MaxDateTime 4421
SysUtils.MaxDateTime constant 4421
SysUtils.MaxEraCount 4422
SysUtils.MaxEraCount constant 4422
SysUtils.MinCurrency 4422

SysUtils.MinCurrency constant 4422
SysUtils.MinDateTime 4423
SysUtils.MinDateTime constant 4423
SysUtils.MinsPerDay 4423
SysUtils.MinsPerDay constant 4423
SysUtils.MinsPerHour 4424
SysUtils.MinsPerHour constant 4424
SysUtils.MonthDays 4424
SysUtils.MonthDays constant 4424
SysUtils.MSecsPerDay 4425
SysUtils.MSecsPerDay constant 4425
SysUtils.MSecsPerSec 4425
SysUtils.MSecsPerSec constant 4425
SysUtils.MSecsToTimeStamp 4270
SysUtils.MSecsToTimeStamp function 4270
SysUtils.NegCurrFormat 4396
SysUtils.NegCurrFormat variable 4396
SysUtils.NewStr 4271
SysUtils.NewStr function 4271
SysUtils.NextCharIndex 4271
SysUtils.NextCharIndex function 4271
SysUtils.Now 4271
SysUtils.Now function 4271
SysUtils.NullAnsiStr 4397
SysUtils.NullAnsiStr variable 4397
SysUtils.NullStr 4397
SysUtils.NullStr variable 4397
SysUtils.NullWideStr 4398
SysUtils.NullWideStr variable 4398
SysUtils.OutOfMemoryError 4272
SysUtils.OutOfMemoryError function 4272
SysUtils.PathDelim 4426
SysUtils.PathDelim constant 4426
SysUtils.PathSep 4426
SysUtils.PathSep constant 4426
SysUtils.PByteArray 4378
SysUtils.PByteArray type 4378
SysUtils.PDayTable 4378
SysUtils.PDayTable type 4378
SysUtils.PExceptionRecord 4379
SysUtils.PExceptionRecord type 4379
SysUtils.pfBCB4Produced 4427
SysUtils.pfBCB4Produced constant 4427
SysUtils.pfConsumerBCB 4427
SysUtils.pfConsumerBCB constant 4427
SysUtils.pfConsumerCompat 4428
SysUtils.pfConsumerCompat constant 4428
SysUtils.pfConsumerDelphi 4428
SysUtils.pfConsumerDelphi constant 4428
SysUtils.pfConsumerMask 4428
SysUtils.pfConsumerMask constant 4428
SysUtils.pfDelphi4Produced 4429
SysUtils.pfDelphi4Produced constant 4429
SysUtils.pfDesignOnly 4429
SysUtils.pfDesignOnly constant 4429
SysUtils.pfExeModule 4430
SysUtils.pfExeModule constant 4430
SysUtils.pfIgnoreDupUnits 4430
SysUtils.pfIgnoreDupUnits constant 4430
SysUtils.pfLibraryModule 4431
SysUtils.pfLibraryModule constant 4431
SysUtils.pfModuleTypeMask 4431
SysUtils.pfModuleTypeMask constant 4431
SysUtils.pfNeverBuild 4432
SysUtils.pfNeverBuild constant 4432
SysUtils.pfPackageModule 4432
SysUtils.pfPackageModule constant 4432
SysUtils.pfProducerMask 4433
SysUtils.pfProducerMask constant 4433
SysUtils.pfProducerUndefined 4433
SysUtils.pfProducerUndefined constant 4433
SysUtils.pfRunOnly 4434
SysUtils.pfRunOnly constant 4434
SysUtils.pfV3Produced 4434
SysUtils.pfV3Produced constant 4434
SysUtils.PlatformBytesOf 4273
SysUtils.PlatformBytesOf function 4273
SysUtils.PlatformStringOf 4273

SysUtils.PlatformStringOf function 4273
SysUtils.PThreadInfo 4379
SysUtils.PThreadInfo type 4379
SysUtils.PWordArray 4379
SysUtils.PWordArray type 4379
SysUtils.QuotedStr 4273
SysUtils.QuotedStr function 4273
SysUtils.RaiseLastOSError 4274
SysUtils.RaiseLastOSError function 4274
SysUtils.RaiseLastWin32Error 4275
SysUtils.RaiseLastWin32Error function 4275
SysUtils.RCS 4276
SysUtils.RCS function 4276
SysUtils.RemoveDir 4276
SysUtils.RemoveDir function 4276
SysUtils.RenameFile 4277
SysUtils.RenameFile function 4277
SysUtils.ReplaceDate 4279
SysUtils.ReplaceDate function 4279
SysUtils.ReplaceTime 4280
SysUtils.ReplaceTime function 4280
SysUtils.RPR 4281
SysUtils.RPR function 4281
SysUtils.SafeLoadLibrary 4281
SysUtils.SafeLoadLibrary function 4281
SysUtils.SameFileName 4282
SysUtils.SameFileName function 4282
SysUtils.SameStr 4282
SysUtils.SameStr function 4282
SysUtils.SameText 4283
SysUtils.SameText function 4283
SysUtils.SecsPerDay 4435
SysUtils.SecsPerDay constant 4435
SysUtils.SecsPerHour 4435
SysUtils.SecsPerHour constant 4435
SysUtils.SecsPerMin 4436
SysUtils.SecsPerMin constant 4436
SysUtils.SetCurrentDir 4283
SysUtils.SetCurrentDir function 4283
SysUtils.ShortDateFormat 4398
SysUtils.ShortDateFormat variable 4398
SysUtils.ShortDayNames 4399
SysUtils.ShortDayNames variable 4399
SysUtils.ShortMonthNames 4399
SysUtils.ShortMonthNames variable 4399
SysUtils.ShortTimeFormat 4400
SysUtils.ShortTimeFormat variable 4400
SysUtils.ShowException 4284
SysUtils.ShowException function 4284
SysUtils.Sleep 4284
SysUtils.Sleep function 4284
SysUtils.StrAlloc 4285
SysUtils.StrAlloc function 4285
SysUtils.StrBufSize 4285
SysUtils.StrBufSize function 4285
SysUtils.StrByteType 4286
SysUtils.StrByteType function 4286
SysUtils.StrCat 4286
SysUtils.StrCat function 4286
SysUtils.StrCharLength 4288
SysUtils.StrCharLength function 4288
SysUtils.StrComp 4288
SysUtils.StrComp function 4288
SysUtils.StrCopy 4289
SysUtils.StrCopy function 4289
SysUtils.StrDispose 4291
SysUtils.StrDispose function 4291
SysUtils.StrECopy 4292
SysUtils.StrECopy function 4292
SysUtils.StrEnd 4293
SysUtils.StrEnd function 4293
SysUtils.StrFmt 4294
SysUtils.StrFmt function 4294
SysUtils.StrIComp 4295
SysUtils.StrIComp function 4295
SysUtils.StringOf 4296
SysUtils.StringOf function 4296
SysUtils.StringReplace 4297

SysUtils.StringReplace function 4297
SysUtils.StringToGUID 4297
SysUtils.StringToGUID function 4297
SysUtils.StrLCat 4299
SysUtils.StrLCat function 4299
SysUtils.StrLComp 4300
SysUtils.StrLComp function 4300
SysUtils.StrLCopy 4302
SysUtils.StrLCopy function 4302
SysUtils.StrLen 4303
SysUtils.StrLen function 4303
SysUtils.StrLFmt 4304
SysUtils.StrLFmt function 4304
SysUtils.StrLIComp 4305
SysUtils.StrLIComp function 4305
SysUtils.StrLower 4307
SysUtils.StrLower function 4307
SysUtils.StrMove 4308
SysUtils.StrMove function 4308
SysUtils.StrNew 4310
SysUtils.StrNew function 4310
SysUtils.StrNextChar 4311
SysUtils.StrNextChar function 4311
SysUtils.StrPas 4311
SysUtils.StrPas function 4311
SysUtils.StrPCopy 4313
SysUtils.StrPCopy function 4313
SysUtils.StrPLCopy 4314
SysUtils.StrPLCopy function 4314
SysUtils.StrPos 4314
SysUtils.StrPos function 4314
SysUtils.StrRScan 4315
SysUtils.StrRScan function 4315
SysUtils.StrScan 4317
SysUtils.StrScan function 4317
SysUtils.StrToBool 4318
SysUtils.StrToBool function 4318
SysUtils.StrToBoolDef 4318
SysUtils.StrToBoolDef function 4318
SysUtils.StrToCurr 4319
SysUtils.StrToCurr function 4319
SysUtils.StrToCurrDef 4320
SysUtils.StrToCurrDef function 4320
SysUtils.StrToDate 4320
SysUtils.StrToDate function 4320
SysUtils.StrToDateDef 4322
SysUtils.StrToDateDef function 4322
SysUtils.StrToDateTime 4322
SysUtils.StrToDateTime function 4322
SysUtils.StrToDateTimeDef 4323
SysUtils.StrToDateTimeDef function 4323
SysUtils.StrToFloat 4324
SysUtils.StrToFloat function 4324
SysUtils.StrToFloatDef 4327
SysUtils.StrToFloatDef function 4327
SysUtils.StrToInt 4328
SysUtils.StrToInt function 4328
SysUtils.StrToInt64 4329
SysUtils.StrToInt64 function 4329
SysUtils.StrToInt64Def 4329
SysUtils.StrToInt64Def function 4329
SysUtils.StrToIntDef 4330
SysUtils.StrToIntDef function 4330
SysUtils.StrToTime 4332
SysUtils.StrToTime function 4332
SysUtils.StrToTimeDef 4333
SysUtils.StrToTimeDef function 4333
SysUtils.StrUpper 4334
SysUtils.StrUpper function 4334
SysUtils.Supports 4335
SysUtils.Supports function 4335
SysUtils.SwitchChars 4436
SysUtils.SwitchChars constant 4436
SysUtils.SysErrorMessage 4336
SysUtils.SysErrorMessage function 4336
SysUtils.SysLocale 4400
SysUtils.SysLocale variable 4400
SysUtils.SystemTimeToDateTime 4336

SysUtils.SystemTimeToDateTime function 4336
SysUtils.TBigEndianUnicodeEncoding 4033
SysUtils.TBigEndianUnicodeEncoding.GetBytes 4034
SysUtils.TBigEndianUnicodeEncoding.GetChars 4035
SysUtils.TBigEndianUnicodeEncoding.GetPreamble 4035
SysUtils.TByteArray 4380
SysUtils.TByteArray type 4380
SysUtils.TBytes 4380
SysUtils.TBytes type 4380
SysUtils.TCharArray 4381
SysUtils.TCharArray type 4381
SysUtils.TCharSearch 4362
SysUtils.TCharSearch record 4362
SysUtils.TDayTable 4381
SysUtils.TDayTable type 4381
SysUtils.TEncoding 4035
SysUtils.TEncoding.ASCII 4048
SysUtils.TEncoding.BigEndianUnicode 4049
SysUtils.TEncoding.Convert 4041
SysUtils.TEncoding.Default 4049
SysUtils.TEncoding.FIsSingleByte 4039
SysUtils.TEncoding.FMaxCharSize 4040
SysUtils.TEncoding.FreeEncodings 4041
SysUtils.TEncoding.GetBufferEncoding 4042
SysUtils.TEncoding.GetByteCount 4042
SysUtils.TEncoding.GetBytes 4043
SysUtils.TEncoding.GetCharCount 4044
SysUtils.TEncoding.GetChars 4045
SysUtils.TEncoding.GetEncoding 4046
SysUtils.TEncoding.GetMaxByteCount 4046
SysUtils.TEncoding.GetMaxCharCount 4046
SysUtils.TEncoding.GetPreamble 4047
SysUtils.TEncoding.GetString 4047
SysUtils.TEncoding.IsSingleByte 4049
SysUtils.TEncoding.IsStandardEncoding 4048
SysUtils.TEncoding.Unicode 4050
SysUtils.TEncoding.UTF7 4050
SysUtils.TEncoding.UTF8 4050
SysUtils.TExceptionRecord 4363
SysUtils.TExceptionRecord record 4363
SysUtils.TextPos 4337
SysUtils.TextPos function 4337
SysUtils.TextToFloat 4338
SysUtils.TextToFloat function 4338
SysUtils.TFileName 4381
SysUtils.TFileName type 4381
SysUtils.TFilenameCaseMatch 4364
SysUtils.TFilenameCaseMatch enumeration 4364
SysUtils.TFloatFormat 4364
SysUtils.TFloatFormat enumeration 4364
SysUtils.TFloatRec 4366
SysUtils.TFloatRec record 4366
SysUtils.TFloatValue 4366
SysUtils.TFloatValue enumeration 4366
SysUtils.TFormatSettings 4367
SysUtils.TFormatSettings record 4367
SysUtils.ThousandSeparator 4402
SysUtils.ThousandSeparator variable 4402
SysUtils.Time 4338
SysUtils.Time function 4338
SysUtils.TimeAMString 4402
SysUtils.TimeAMString variable 4402
SysUtils.TimePMString 4402
SysUtils.TimePMString variable 4402
SysUtils.TimeSeparator 4403
SysUtils.TimeSeparator variable 4403
SysUtils.TimeStampToDateTime 4339
SysUtils.TimeStampToDateTime function 4339
SysUtils.TimeStampToMSecs 4340
SysUtils.TimeStampToMSecs function 4340
SysUtils.TimeToStr 4340
SysUtils.TimeToStr function 4340
SysUtils.TIntegerSet 4382
SysUtils.TIntegerSet type 4382
SysUtils.TLangRec 4368
SysUtils.TLangRec record 4368
SysUtils.TLanguages 4051
SysUtils.TLanguages.Count 4053

SysUtils.TLanguages.Create 4052
SysUtils.TLanguages.Ext 4053
SysUtils.TLanguages.ID 4054
SysUtils.TLanguages.IndexOf 4052
SysUtils.TLanguages.LocaleID 4054
SysUtils.TLanguages.Name 4054
SysUtils.TLanguages.NameFromLCID 4055
SysUtils.TLanguages.NameFromLocaleID 4055
SysUtils.TLocaleOptions 4369
SysUtils.TLocaleOptions enumeration 4369
SysUtils.TMbcsByteType 4369
SysUtils.TMbcsByteType enumeration 4369
SysUtils.TMBCSEncoding 4056
SysUtils.TMBCSEncoding.Create 4057, 4058
SysUtils.TMBCSEncoding.GetByteCount 4058
SysUtils.TMBCSEncoding.GetBytes 4059
SysUtils.TMBCSEncoding.GetCharCount 4059
SysUtils.TMBCSEncoding.GetChars 4060
SysUtils.TMBCSEncoding.GetMaxByteCount 4060
SysUtils.TMBCSEncoding.GetMaxCharCount 4060
SysUtils.TMBCSEncoding.GetPreamble 4061
SysUtils.TMREWSync 4382
SysUtils.TMREWSync type 4382
SysUtils.TMultiReadExclusiveWriteSynchronizer 4061
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginRead 4064
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginWrite 4064
SysUtils.TMultiReadExclusiveWriteSynchronizer.Create 4063
SysUtils.TMultiReadExclusiveWriteSynchronizer.Destroy 4063
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndRead 4065
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndWrite 4065
SysUtils.TMultiReadExclusiveWriteSynchronizer.RevisionLevel 4066
SysUtils.TNameType 4370
SysUtils.TNameType enumeration 4370
SysUtils.TPackageInfoProc 4383
SysUtils.TPackageInfoProc type 4383
SysUtils.TProc 4383
SysUtils.TProc type 4383
SysUtils.TProcedure 4384
SysUtils.TProcedure type 4384
SysUtils.TReplaceFlags 4384
SysUtils.TReplaceFlags type 4384
SysUtils.Trim 4342
SysUtils.Trim function 4342
SysUtils.TrimLeft 4343
SysUtils.TrimLeft function 4343
SysUtils.TrimRight 4343
SysUtils.TrimRight function 4343
SysUtils.TrueBoolStrs 4403
SysUtils.TrueBoolStrs variable 4403
SysUtils.TryEncodeDate 4343
SysUtils.TryEncodeDate function 4343
SysUtils.TryEncodeTime 4344
SysUtils.TryEncodeTime function 4344
SysUtils.TryFloatToCurr 4345
SysUtils.TryFloatToCurr function 4345
SysUtils.TryFloatToDateTime 4345
SysUtils.TryFloatToDateTime function 4345
SysUtils.TryStrToBool 4346
SysUtils.TryStrToBool function 4346
SysUtils.TryStrToCurr 4347
SysUtils.TryStrToCurr function 4347
SysUtils.TryStrToDate 4347
SysUtils.TryStrToDate function 4347
SysUtils.TryStrToDateTime 4348
SysUtils.TryStrToDateTime function 4348
SysUtils.TryStrToFloat 4349
SysUtils.TryStrToFloat function 4349
SysUtils.TryStrToInt 4349
SysUtils.TryStrToInt function 4349
SysUtils.TryStrToInt64 4350
SysUtils.TryStrToInt64 function 4350
SysUtils.TryStrToTime 4351
SysUtils.TryStrToTime function 4351
SysUtils.TSearchRec 4371
SysUtils.TSearchRec record 4371
SysUtils.TSimpleRWSync 4066

SysUtils.TSimpleRWSync.BeginRead 4069
SysUtils.TSimpleRWSync.BeginWrite 4069
SysUtils.TSimpleRWSync.Create 4068
SysUtils.TSimpleRWSync.Destroy 4068
SysUtils.TSimpleRWSync.EndRead 4070
SysUtils.TSimpleRWSync.EndWrite 4070
SysUtils.TStringBuilder 4070
SysUtils.TStringBuilder.Append 4078
SysUtils.TStringBuilder.AppendFormat 4079
SysUtils.TStringBuilder.AppendLine 4080
SysUtils.TStringBuilder.Capacity 4085
SysUtils.TStringBuilder.Chars 4085
SysUtils.TStringBuilder.CopyTo 4080
SysUtils.TStringBuilder.Create 4075
SysUtils.TStringBuilder.EnsureCapacity 4080
SysUtils.TStringBuilder.Equals 4081
SysUtils.TStringBuilder.FData 4076
SysUtils.TStringBuilder.FLength 4076
SysUtils.TStringBuilder.FMaxCapacity 4076
SysUtils.TStringBuilder.Insert 4082
SysUtils.TStringBuilder.Length 4086
SysUtils.TStringBuilder.MaxCapacity 4086
SysUtils.TStringBuilder.Remove 4083
SysUtils.TStringBuilder.Replace 4084
SysUtils.TStringBuilder.ToString 4084
SysUtils.TSysCharSet 4384
SysUtils.TSysCharSet type 4384
SysUtils.TSysLocale 4374
SysUtils.TSysLocale record 4374
SysUtils.TTerminateProc 4385
SysUtils.TTerminateProc type 4385
SysUtils.TThreadInfo 4375
SysUtils.TThreadInfo record 4375
SysUtils.TThreadLocalCounter 4087
SysUtils.TThreadLocalCounter.Close 4088
SysUtils.TThreadLocalCounter.Delete 4088
SysUtils.TThreadLocalCounter.Destroy 4087
SysUtils.TThreadLocalCounter.Open 4088
SysUtils.TTimeStamp 4375
SysUtils.TTimeStamp record 4375
SysUtils.TUnicodeEncoding 4088
SysUtils.TUnicodeEncoding.Create 4090
SysUtils.TUnicodeEncoding.GetByteCount 4090
SysUtils.TUnicodeEncoding.GetBytes 4091
SysUtils.TUnicodeEncoding.GetCharCount 4091
SysUtils.TUnicodeEncoding.GetChars 4092
SysUtils.TUnicodeEncoding.GetMaxByteCount 4092
SysUtils.TUnicodeEncoding.GetMaxCharCount 4092
SysUtils.TUnicodeEncoding.GetPreamble 4093
SysUtils.TUTF7Encoding 4093
SysUtils.TUTF7Encoding.Create 4094
SysUtils.TUTF7Encoding.GetByteCount 4095
SysUtils.TUTF7Encoding.GetBytes 4095
SysUtils.TUTF7Encoding.GetCharCount 4096
SysUtils.TUTF7Encoding.GetChars 4096
SysUtils.TUTF7Encoding.GetMaxByteCount 4097
SysUtils.TUTF7Encoding.GetMaxCharCount 4097
SysUtils.TUTF8Encoding 4098
SysUtils.TUTF8Encoding.Create 4099
SysUtils.TUTF8Encoding.GetMaxByteCount 4099
SysUtils.TUTF8Encoding.GetMaxCharCount 4100
SysUtils.TUTF8Encoding.GetPreamble 4100
SysUtils.TValidatePackageProc 4385
SysUtils.TValidatePackageProc type 4385
SysUtils.TwoDigitYearCenturyWindow 4404
SysUtils.TwoDigitYearCenturyWindow variable 4404
SysUtils.TWordArray 4386
SysUtils.TWordArray type 4386
SysUtils.ufImplicitUnit 4437
SysUtils.ufImplicitUnit constant 4437
SysUtils.ufMainUnit 4437
SysUtils.ufMainUnit constant 4437
SysUtils.ufOrgWeakUnit 4438
SysUtils.ufOrgWeakUnit constant 4438
SysUtils.ufPackageUnit 4438
SysUtils.ufPackageUnit constant 4438
SysUtils.ufWeakPackageUnit 4439
SysUtils.ufWeakPackageUnit constant 4439

SysUtils.ufWeakUnit 4439
SysUtils.ufWeakUnit constant 4439
SysUtils.UIntToStr 4351, 4352
SysUtils.UIntToStr function 4351, 4352
SysUtils.UnixDateDelta 4439
SysUtils.UnixDateDelta constant 4439
SysUtils.UnloadPackage 4352
SysUtils.UnloadPackage function 4352
SysUtils.UpperCase 4352
SysUtils.UpperCase function 4352
SysUtils.WideBytesOf 4354
SysUtils.WideBytesOf function 4354
SysUtils.WideCompareStr 4354
SysUtils.WideCompareStr function 4354
SysUtils.WideCompareText 4355
SysUtils.WideCompareText function 4355
SysUtils.WideFmtStr 4355
SysUtils.WideFmtStr function 4355
SysUtils.WideFormat 4356
SysUtils.WideFormat function 4356
SysUtils.WideFormatBuf 4356
SysUtils.WideFormatBuf function 4356
SysUtils.WideLowerCase 4357
SysUtils.WideLowerCase function 4357
SysUtils.WideSameStr 4357
SysUtils.WideSameStr function 4357
SysUtils.WideSameText 4358
SysUtils.WideSameText function 4358
SysUtils.WideStrAlloc 4358
SysUtils.WideStrAlloc function 4358
SysUtils.WideStringOf 4359
SysUtils.WideStringOf function 4359
SysUtils.WideUpperCase 4359
SysUtils.WideUpperCase function 4359
SysUtils.Win32BuildNumber 4404
SysUtils.Win32BuildNumber variable 4404
SysUtils.Win32Check 4359
SysUtils.Win32Check function 4359
SysUtils.Win32CSDVersion 4405
SysUtils.Win32CSDVersion variable 4405
SysUtils.Win32MajorVersion 4405
SysUtils.Win32MajorVersion variable 4405
SysUtils.Win32MinorVersion 4406
SysUtils.Win32MinorVersion variable 4406
SysUtils.Win32Platform 4406
SysUtils.Win32Platform variable 4406
SysUtils.WordRec 4376
SysUtils.WordRec record 4376
SysUtils.WrapText 4360
SysUtils.WrapText function 4360

T

TabNotBk 4441
TabNotBk namespace 4441
  Classes 4442
  Constants 4492
  Types 4491
TabNotBk.CM_TABFONTCHANGED 4492
TabNotBk.CM_TABFONTCHANGED constant 4492
TabNotBk.TPageChangeEvent 4491
TabNotBk.TPageChangeEvent type 4491
TabNotBk.TTabbedNotebook 4442
TabNotBk.TTabbedNotebook.ActivePage 4451
TabNotBk.TTabbedNotebook.Align 4452
TabNotBk.TTabbedNotebook.AlignControls 4445
TabNotBk.TTabbedNotebook.Anchors 4455
TabNotBk.TTabbedNotebook.Change 4445
TabNotBk.TTabbedNotebook.Click 4445
TabNotBk.TTabbedNotebook.CMTabFontChanged 4446
TabNotBk.TTabbedNotebook.Constraints 4456
TabNotBk.TTabbedNotebook.Create 4443
TabNotBk.TTabbedNotebook.CreateHandle 4446
TabNotBk.TTabbedNotebook.CreateParams 4447
TabNotBk.TTabbedNotebook.Destroy 4444
TabNotBk.TTabbedNotebook.DoubleBuffered 4456
TabNotBk.TTabbedNotebook.Enabled 4458
TabNotBk.TTabbedNotebook.Font 4458
TabNotBk.TTabbedNotebook.GetChildOwner 4447

TabNotBk.TTabbedNotebook.GetChildren 4447
TabNotBk.TTabbedNotebook.GetIndexForPage 4448
TabNotBk.TTabbedNotebook.KeyDown 4448
TabNotBk.TTabbedNotebook.Loaded 4449
TabNotBk.TTabbedNotebook.OnChange 4473
TabNotBk.TTabbedNotebook.OnClick 4473
TabNotBk.TTabbedNotebook.OnContextPopup 4474
TabNotBk.TTabbedNotebook.OnEnter 4474
TabNotBk.TTabbedNotebook.OnExit 4477
TabNotBk.TTabbedNotebook.PageIndex 4459
TabNotBk.TTabbedNotebook.Pages 4460
TabNotBk.TTabbedNotebook.ParentDoubleBuffered 4460
TabNotBk.TTabbedNotebook.ParentShowHint 4461
TabNotBk.TTabbedNotebook.PopupMenu 4461
TabNotBk.TTabbedNotebook.ReadState 4449
TabNotBk.TTabbedNotebook.SetPageIndex 4450
TabNotBk.TTabbedNotebook.ShowControl 4450
TabNotBk.TTabbedNotebook.ShowHint 4466
TabNotBk.TTabbedNotebook.TabFont 4468
TabNotBk.TTabbedNotebook.TabFontChanged 4450
TabNotBk.TTabbedNotebook.TabOrder 4468
TabNotBk.TTabbedNotebook.Tabs 4469
TabNotBk.TTabbedNotebook.TabsPerRow 4470
TabNotBk.TTabbedNotebook.TabStop 4471
TabNotBk.TTabbedNotebook.TopFont 4471
TabNotBk.TTabbedNotebook.Visible 4471
TabNotBk.TTabPage 4480
TabNotBk.TTabPage.Caption 4482
TabNotBk.TTabPage.Create 4481
TabNotBk.TTabPage.Enabled 4485
TabNotBk.TTabPage.Height 4485
TabNotBk.TTabPage.ReadState 4481
TabNotBk.TTabPage.TabOrder 4486
TabNotBk.TTabPage.Visible 4487
TabNotBk.TTabPage.Width 4488
Tabs 4493
Tabs namespace 4493
Classes 4494
Structs, Records, Enums 4573
Types 4576
Tabs.TDrawTabEvent 4576
Tabs.TDrawTabEvent type 4576
Tabs.TEdgePart 4573
Tabs.TEdgePart enumeration 4573
Tabs.TEdgeType 4573
Tabs.TEdgeType enumeration 4573
Tabs.TMeasureTabEvent 4576
Tabs.TMeasureTabEvent type 4576
Tabs.TScrollBtn 4574
Tabs.TScrollBtn enumeration 4574
Tabs.TScroller 4494
Tabs.TScroller.Change 4497
Tabs.TScroller.Create 4495
Tabs.TScroller.Destroy 4496
Tabs.TScroller.Max 4497
Tabs.TScroller.Min 4498
Tabs.TScroller.OnClick 4499
Tabs.TScroller.Paint 4496
Tabs.TScroller.Position 4498
Tabs.TScroller.ScrollOrientation 4498
Tabs.TScrollOrientation 4574
Tabs.TScrollOrientation enumeration 4574
Tabs.TTabChangeEvent 4577
Tabs.TTabChangeEvent type 4577
Tabs.TTabList 4499
Tabs.TTabList.Add 4500
Tabs.TTabList.AddStrings 4503
Tabs.TTabList.Clear 4504
Tabs.TTabList.Create 4500
Tabs.TTabList.Delete 4505
Tabs.TTabList.Insert 4505
Tabs.TTabList.Put 4506
Tabs.TTabSet 4506
Tabs.TTabSet.Align 4517
Tabs.TTabSet.Anchors 4521
Tabs.TTabSet.AutoScroll 4521
Tabs.TTabSet.BackgroundColor 4522
Tabs.TTabSet.CanChange 4510

Tabs.TTabSet.Canvas 4522
Tabs.TTabSet.CMHintShow 4510
Tabs.TTabSet.Constraints 4524
Tabs.TTabSet.Create 4509
Tabs.TTabSet.CreateParams 4511
Tabs.TTabSet.Destroy 4509
Tabs.TTabSet.DitherBackground 4525
Tabs.TTabSet.DoubleBuffered 4525
Tabs.TTabSet.DragCursor 4527
Tabs.TTabSet.DragKind 4527
Tabs.TTabSet.DragMode 4528
Tabs.TTabSet.DrawTab 4511
Tabs.TTabSet.Enabled 4529
Tabs.TTabSet.EndMargin 4530
Tabs.TTabSet.FirstIndex 4530
Tabs.TTabSet.Font 4530
Tabs.TTabSet.GetChildren 4511
Tabs.TTabSet.GetImageIndex 4512
Tabs.TTabSet.Images 4532
Tabs.TTabSet.ItemAtPos 4512
Tabs.TTabSet.ItemRect 4512
Tabs.TTabSet.ItemWidth 4513
Tabs.TTabSet.MeasureTab 4513
Tabs.TTabSet.MinClientRect 4513
Tabs.TTabSet.MouseDown 4514
Tabs.TTabSet.Notification 4515
Tabs.TTabSet.OnAlignInsertBefore 4545
Tabs.TTabSet.OnAlignPosition 4546
Tabs.TTabSet.OnChange 4547
Tabs.TTabSet.OnClick 4547
Tabs.TTabSet.OnDragDrop 4549
Tabs.TTabSet.OnDragOver 4552
Tabs.TTabSet.OnDrawTab 4555
Tabs.TTabSet.OnEndDock 4555
Tabs.TTabSet.OnEndDrag 4556
Tabs.TTabSet.OnEnter 4557
Tabs.TTabSet.OnExit 4559
Tabs.TTabSet.OnGetImageIndex 4562
Tabs.TTabSet.OnMeasureTab 4562
Tabs.TTabSet.OnMouseActivate 4563
Tabs.TTabSet.OnMouseDown 4564
Tabs.TTabSet.OnMouseEnter 4566
Tabs.TTabSet.OnMouseLeave 4566
Tabs.TTabSet.OnMouseMove 4566
Tabs.TTabSet.OnMouseUp 4569
Tabs.TTabSet.OnStartDock 4571
Tabs.TTabSet.OnStartDrag 4572
Tabs.TTabSet.Paint 4515
Tabs.TTabSet.ParentBackground 4532
Tabs.TTabSet.ParentDoubleBuffered 4532
Tabs.TTabSet.ParentShowHint 4532
Tabs.TTabSet.PopupMenu 4533
Tabs.TTabSet.Scroller 4538
Tabs.TTabSet.SelectedColor 4538
Tabs.TTabSet.SelectNext 4516
Tabs.TTabSet.SetTabIndex 4516
Tabs.TTabSet.ShowHint 4538
Tabs.TTabSet.ShrinkToFit 4540
Tabs.TTabSet.SoftTop 4540
Tabs.TTabSet.StartMargin 4540
Tabs.TTabSet.Style 4541
Tabs.TTabSet.TabHeight 4541
Tabs.TTabSet.TabIndex 4542
Tabs.TTabSet.TabPosition 4542
Tabs.TTabSet.Tabs 4542
Tabs.TTabSet.UnselectedColor 4543
Tabs.TTabSet.Visible 4543
Tabs.TTabSet.VisibleTabs 4544
Tabs.TTabSetDrawTabEvent 4577
Tabs.TTabSetDrawTabEvent type 4577
Tabs.TTabSetTabStyle 4575
Tabs.TTabSetTabStyle enumeration 4575
Tabs.TTabStyle 4578
Tabs.TTabStyle type 4578
TAbstractAdapterRequest class 5929
- about TAbstractAdapterRequest class 5929
- Create 5931
- Request 5932

TAbstractAdapterRequest Members 5930
TAbstractAdapterRequest properties 5931
TAbstractAdapterResponse class 5932
about TAbstractAdapterResponse class 5932
Create 5934
Response 5934
TAbstractAdapterResponse Members 5932
TAbstractAdapterResponse properties 5934
TAbstractControlRequirements class 6281
about TAbstractControlRequirements class 6281
DefineFunction 6282
DefineVariable 6282
IsFunctionDefined 6283
IsVariableDefined 6283
RequiresMultipartForm 6284
TAbstractControlRequirements Members 6281
TAbstractControlRequirements methods 6282
TAbstractDesigntimeWarnings class 905
about TAbstractDesigntimeWarnings class 905
AddObject 906
AddString 906
TAbstractDesigntimeWarnings Members 905
TAbstractDesigntimeWarnings methods 906
TAbstractFieldValuesAdapter class 5169
about TAbstractFieldValuesAdapter class 5169
GetField 5175
GetValueFieldClass 5175
GetValueIndex 5176
ImplCanAddFieldClass 5176
ImplEndIterator 5177
ImplGetField 5177
ImplGetFieldsList 5177
ImplGetValueIndex 5178
ImplNextIteration 5178
ImplStartIterator 5179
TAbstractFieldValuesAdapter Members 5171
TAbstractFieldValuesAdapter methods 5174
TAbstractFormRequirements class 6284
about TAbstractFormRequirements class 6284
DefineFunction 6285
DefineVariable 6285
IsFunctionDefined 6285
IsVariableDefined 6286
TAbstractFormRequirements Members 6284
TAbstractFormRequirements methods 6285
TAbstractInetFileType class 907
about TAbstractInetFileType class 907
FileExt 909
FileExtCount 909
GetExtCount 908
GetFileExt 908
GetTypeID 909
TAbstractInetFileType Members 907
TAbstractInetFileType methods 908
TAbstractInetFileType properties 909
TypeID 910
TAbstractNamedVariants class 6090
about TAbstractNamedVariants class 6090
Add 6093
Append 6094
Assign 6094
BeginUpdate 6094
Capacity 6103
Clear 6095
CompareStrings 6095
Count 6103
DefineProperties 6096
Delete 6096
Destroy 6092
EndUpdate 6097
Equals 6097
Error 6098
Exchange 6098
Get 6098
GetCapacity 6099
GetCount 6099
IndexOfName 6100
Insert 6100

Move 6101
Names 6104
Put 6101
SetCapacity 6102
SetUpdateState 6102
TAbstractNamedVariants Members 6091
TAbstractNamedVariants methods 6092
TAbstractNamedVariants properties 6103
UpdateCount 6104
Values 6105
Variants 6105
TAbstractScriptEngine class 6614
about TAbstractScriptEngine class 6614
AddObjectFactory 6616
Create 6615
CreateComponentStub 6617
Destroy 6615
ListLanguageNames 6617
Name 6617
ScriptObjectFactories 6618
ScriptProducerClass 6618
SupportsLanguage 6617
TAbstractScriptEngine Members 6614
TAbstractScriptEngine methods 6616
TAbstractScriptEngine properties 6617
TAbstractScriptObjectFactories class 6618
about TAbstractScriptObjectFactories class 6618
Count 6621
Factories 6621
GetCount 6620
GetFactory 6620
GetObjectList 6620
ObjectList 6621
TAbstractScriptObjectFactories Members 6618
TAbstractScriptObjectFactories methods 6619
TAbstractScriptObjectFactories properties 6620
TAbstractScriptObjectFactory class 6621
about TAbstractScriptObjectFactory class 6621
AddGlobalObjects 6622
CreateComponentObject 6622
CreateGlobalObject 6623
CreateProducerObject 6623
GetScriptEngine 6623
TAbstractScriptObjectFactory Members 6622
TAbstractScriptObjectFactory methods 6622
TAbstractSessionIDGen class 328
about TAbstractSessionIDGen class 328
NextID 329
TAbstractSessionIDGen Members 329
TAbstractSessionIDGen methods 329
TAbstractSessionStatusChange class 329
about TAbstractSessionStatusChange class 329
SessionStarted 330
SessionTerminated 330
SessionTimeout 331
TAbstractSessionStatusChange Members 330
TAbstractSessionStatusChange methods 330
TAbstractSocket class 4
about TAbstractSocket class 4
Active 11
Address 12
Close 6
DoActivate 6
Error 6
Event 7
Host 12
InitSocket 8
Loaded 8
Open 9
Port 13
Service 13
SetActive 9
SetAddress 10
SetHost 10
SetPort 10
SetService 11
TAbstractSocket Members 5
TAbstractSocket methods 5

TAbstractSocket properties 11
TAbstractWebContext class 5935
about TAbstractWebContext class 5935
AdapterRequest 5946
AdapterResponse 5946
Create 5937
Destroy 5937
DispatchedPageName 5947
EndUser 5947
FindApplicationModule 5938
FindDefaultPageInfo 5939
FindModuleClass 5939
FindModuleName 5940
FindPage 5940
FindPageInfo 5941
FindVariableContainer 5941
GetAdapterRequest 5942
GetAdapterResponse 5942
GetDispatchedPageName 5942
GetEndUser 5942
GetPageCount 5943
GetPageInfo 5943
GetRequestedPageName 5943
GetSession 5943
GetWebModules 5944
GetWebRequest 5944
GetWebResponse 5944
PageCount 5947
Pages 5948
Request 5948
RequestedPageName 5949
Response 5949
Session 5949
SetAdapterRequest 5944
SetAdapterResponse 5945
SetDispatchedPageName 5945
SetRequestedPageName 5945
TAbstractWebContext Members 5935
TAbstractWebContext methods 5938
TAbstractWebContext properties 5945
WebModules 5950
TAdapter class 5179
about TAdapter class 5179
Actions 5184
Data 5185
ModifyAccess 5185
OnAfterExecuteAction 5186
OnAfterGetActionResponse 5186
OnAfterUpdateFields 5187
OnAfterValidateFields 5187
OnBeforeExecuteAction 5187
OnBeforeGetActionResponse 5188
OnBeforeUpdateFields 5188
OnBeforeValidateFields 5189
OnGetActionParams 5189
OnIterateRecords 5189
TAdapter events 5186
TAdapter Members 5181
TAdapter properties 5184
ViewAccess 5185
TAdapterAction class 5190
about TAdapterAction class 5190
ActionName 5194
DisplayLabel 5194
ExecuteAccess 5195
OnAfterExecute 5196
OnAfterGetResponse 5196
OnBeforeExecute 5197
OnBeforeGetResponse 5197
OnExecute 5197
OnGetEnabled 5198
OnGetParams 5198
RedirectOptions 5195
TAdapterAction events 5195
TAdapterAction Members 5191
TAdapterAction properties 5194
TAdapterActionButton class 6286
about TAdapterActionButton class 6286

ActionName 6288
Align 6288
Caption 6289
Custom 6289
DisplayColumns 6290
DisplayType 6290
ErrorPageName 6291
HideOptions 6291
ImageHeight 6291
ImageProducer 6292
ImageWidth 6292
PageName 6293
Style 6293
StyleRule 6293
TAdapterActionButton Members 6287
TAdapterActionButton properties 6288
VAlign 6294
TAdapterActionName class 6502
about TAdapterActionName class 6502
Action 6504
ActionName 6505
Adapter 6505
AssignTo 6503
Create 6503
TAdapterActionName Members 6502
TAdapterActionName methods 6503
TAdapterActionName properties 6504
TAdapterActions class 5198
about TAdapterActions class 5198
AdapterCanAddClass 5201
ImplCanAddClass 5201
ImplGetWebComponentNodeName 5202
SetWebParent 5202
TAdapterActions Members 5199
TAdapterActions methods 5201
TAdapterActions properties 5203
WebActions 5203
TAdapterActionsListHelper class 6294
about TAdapterActionsListHelper class 6294
AddDefaultActions 6301
Create 6296
FindAction 6297
GetActionsList 6297
GetAddAllFieldsItem 6298
GetAddFieldsDlgCaption 6298
GetAddFieldsItem 6299
GetNewFieldPrefix 6299
GetVisibleActions 6300
IsActionInUse 6300
TAdapterActionsListHelper Members 6295
TAdapterActionsListHelper methods 6297
TAdapterActionsListHelper properties 6301
TAdapterApplicationField class 5203
about TAdapterApplicationField class 5203
Adapter 5207
TAdapterApplicationField Members 5204
TAdapterApplicationField properties 5207
TAdapterApplicationTitleField class 5207
about TAdapterApplicationTitleField class 5207
GetDefaultFieldName 5211
ImplGetValue 5211
TAdapterApplicationTitleField Members 5208
TAdapterApplicationTitleField methods 5211
TAdapterBooleanField class 5212
about TAdapterBooleanField class 5212
DisplayLabel 5217
DisplayWidth 5217
FieldName 5217
ModifyAccess 5218
OnGetValue 5219
OnUpdateValue 5219
OnValidateValue 5220
ReadOnly 5218
TAdapterBooleanField events 5219
TAdapterBooleanField Members 5213
TAdapterBooleanField properties 5216
ValuesList 5218
ViewAccess 5219

TAdapterCommandColumn class 6301
about TAdapterCommandColumn class 6301
AddDefaultCommands 6304
Align 6304
Caption 6304
CaptionAttributes 6305
Custom 6305
DisplayColumns 6306
Style 6306
StyleRule 6306
TAdapterCommandColumn Members 6302
TAdapterCommandColumn properties 6303
VAlign 6307
TAdapterCommandGroup class 6307
about TAdapterCommandGroup class 6307
AddDefaultCommands 6310
Custom 6310
DisplayComponent 6311
Style 6311
StyleRule 6311
TAdapterCommandGroup Members 6308
TAdapterCommandGroup properties 6310
TAdapterControlGroup class 6312
about TAdapterControlGroup class 6312
Adapter 6322
AdapterChange 6315
AdapterMode 6323
AddDefaultFields 6323
AddNotify 6315
Create 6313
CreateDefaultFields 6316
Destroy 6314
FindField 6316
GetAdapter 6317
GetAdapterModeOfAdapter 6317
GetAdapterModeOfDisplay 6317
GetAdapterVariableName 6318
GetControlRequirements 6318
GetDesigntimeWarnings 6318
GetFieldsList 6319
GetVisibleFields 6319
ImplGetControlRequirements 6320
IsFieldInUse 6320
Notification 6320
NotifyAdapterChange 6321
RemoveNotify 6321
SetAdapter 6322
TAdapterControlGroup Members 6313
TAdapterControlGroup methods 6314
TAdapterControlGroup properties 6322
VisibleFields 6324
TAdapterDataDisplay class 6324
about TAdapterDataDisplay class 6324
Align 6332
Create 6326
FieldConditions 6327
FieldName 6332
FindAssociatedField 6327
FormatCaption 6327
FormatColumnData 6327
FormatColumnHeading 6327
GetAdapterModeOfAdapter 6328
GetAdapterVariableName 6328
GetCaption 6328
GetControlRequirements 6328
GetDesigntimeWarnings 6329
GetFieldName 6329
GetVariablesContainer 6329
HideOptions 6332
ImplContent 6329
ImplFormatColumnData 6330
ImplFormatColumnHeading 6330
ImplGetControlRequirements 6330
ImplRestoreDefaults 6330
RestoreDefaults 6331
SetCaption 6331
SetFieldName 6331
TAdapterDataDisplay Members 6325

TAdapterDataDisplay methods 6326
TAdapterDataDisplay properties 6331
VAlign 6332
TAdapterDefaultActions class 5220
about TAdapterDefaultActions class 5220
ClearDefaultComponents 5223
DefaultWebComponents 5226
Destroy 5222
GetAddAllFieldsItem 5224
GetAddFieldsDlgCaption 5224
GetAddFieldsItem 5224
GetDefaultWebComponents 5224
GetNewFieldPrefix 5225
GetVariablesInUse 5225
SetDefaultWebComponents 5225
TAdapterDefaultActions Members 5221
TAdapterDefaultActions methods 5223
TAdapterDefaultActions properties 5226
TAdapterDefaultFields class 5226
about TAdapterDefaultFields class 5226
ClearDefaultComponents 5229
DefaultWebComponents 5232
Destroy 5229
GetAddAllFieldsItem 5230
GetAddFieldsDlgCaption 5230
GetAddFieldsItem 5230
GetDefaultWebComponents 5230
GetNewFieldPrefix 5231
GetVariablesInUse 5231
SetDefaultWebComponents 5231
TAdapterDefaultFields Members 5227
TAdapterDefaultFields methods 5229
TAdapterDefaultFields properties 5232
TAdapterDispatcher class 6122
about TAdapterDispatcher class 6122
OnActionRequestNotHandled 6124
OnAfterDispatchAction 6125
OnAfterDispatchImage 6125
OnBeforeDispatchAction 6125
OnBeforeDispatchImage 6126
OnGetAdapterRequestParams 6126
OnImageRequestNotHandled 6127
TAdapterDispatcher events 6124
TAdapterDispatcher Members 6123
TAdapterDispatchParams class 6128
about TAdapterDispatchParams class 6128
Create 6130
Destroy 6130
GetFailurePage 6131
GetHandler 6131
GetObjectIdentifier 6132
GetParams 6132
GetRequestIdentifier 6132
GetSuccessPage 6132
Handler 6133
ObjectIdentifier 6133
RequestIdentifier 6134
TAdapterDispatchParams Members 6128
TAdapterDispatchParams methods 6131
TAdapterDispatchParams properties 6133
TAdapterDisplayColumn class 6333
about TAdapterDisplayColumn class 6333
Align 6335
Caption 6336
CaptionAttributes 6336
Custom 6336
DisplayType 6337
FieldName 6337
HideOptions 6338
ImageHeight 6338
ImageWidth 6338
Style 6338
StyleRule 6339
TAdapterDisplayColumn Members 6334
TAdapterDisplayColumn properties 6335
VAlign 6339
TAdapterDisplayField class 6339
about TAdapterDisplayField class 6339

Align 6343
Caption 6343
CaptionAttributes 6343
CaptionPosition 6344
Custom 6344
DisplayColumns 6344
DisplayRows 6345
DisplayType 6345
DisplayWidth 6345
FieldName 6346
HideOptions 6346
ImageHeight 6346
ImageWidth 6347
InputType 6347
MaxLength 6347
SelectRows 6348
Style 6348
StyleRule 6348
TAdapterDisplayField Members 6341
TAdapterDisplayField properties 6342
TextAreaWrap 6349
VAlign 6349
ViewMode 6349
TAdapterDisplayReferenceGroup class 6349
about TAdapterDisplayReferenceGroup class 6349
AdapterChange 6352
Create 6351
Destroy 6351
DisplayComponent 6355
DisplayComponentChange 6352
GetAdapterDisplayCharacteristics 6353
GetDesigntimeWarnings 6353
NotifyAdapterChange 6353
OnDisplayComponentChange 6354
SetWebDisplayComponent 6354
TAdapterDisplayReferenceGroup Members 6350
TAdapterDisplayReferenceGroup methods 6351
TAdapterDisplayReferenceGroup properties 6355
TAdapterEditColumn class 6355
about TAdapterEditColumn class 6355
Align 6358
Caption 6358
CaptionAttributes 6359
CaptionPosition 6359
Custom 6360
DisplayColumns 6360
DisplayRows 6360
DisplayType 6361
DisplayWidth 6361
FieldName 6361
HideOptions 6362
ImageHeight 6362
ImageWidth 6362
InputType 6362
MaxLength 6363
SelectRows 6363
Style 6363
StyleRule 6364
TAdapterEditColumn Members 6356
TAdapterEditColumn properties 6358
TextAreaWrap 6364
VAlign 6364
ViewMode 6365
TAdapterEndUserDisplayNameField class 5233
about TAdapterEndUserDisplayNameField class 5233
GetDefaultFieldName 5236
ImplGetValue 5236
TAdapterEndUserDisplayNameField Members 5233
TAdapterEndUserDisplayNameField methods 5236
TAdapterEndUserField class 5237
about TAdapterEndUserField class 5237
Adapter 5240
TAdapterEndUserField Members 5238
TAdapterEndUserField properties 5240
TAdapterEndUserLoggedInField class 5241
about TAdapterEndUserLoggedInField class 5241
GetDefaultFieldName 5244
ImplGetValue 5244

TAdapterEndUserLoggedInField Members 5241
TAdapterEndUserLoggedInField methods 5244
TAdapterError class 5245
about TAdapterError class 5245
Create 5247
GetErrorText 5247
GetField 5247
GetID 5248
TAdapterError Members 5245
TAdapterError methods 5247
TAdapterErrorList class 6365
about TAdapterErrorList class 6365
Adapter 6367
TAdapterErrorList Members 6366
TAdapterErrorList properties 6367
TAdapterErrorsList class 5248
about TAdapterErrorsList class 5248
AddError 5252
AddError method 5252
Create 5251
Destroy 5251
FieldNames 5255
IDs 5255
ImplClearErrors 5253
ImplDefineLabel 5253
ImplGetError 5253
ImplGetErrorCount 5254
ImplGetObjectErrors 5254
ImplHasObjectErrors 5254
Messages 5255
TAdapterErrorsList Members 5249
TAdapterErrorsList methods 5252
TAdapterErrorsList properties 5254
TAdapterField class 5256
about TAdapterField class 5256
DisplayLabel 5261
DisplayWidth 5261
FieldName 5261
ModifyAccess 5262
OnGetDisplayText 5264
OnGetValue 5264
OnUpdateValue 5264
OnValidateValue 5265
ReadOnly 5262
Required 5262
TAdapterField events 5264
TAdapterField Members 5257
TAdapterField properties 5260
ValuesList 5263
ViewAccess 5263
TAdapterFieldGroup class 6367
about TAdapterFieldGroup class 6367
Adapter 6370
AdapterMode 6371
AddDefaultFields 6371
Custom 6372
Style 6372
StyleRule 6373
TAdapterFieldGroup Members 6369
TAdapterFieldGroup properties 6370
TAdapterFields class 5265
about TAdapterFields class 5265
AdapterCanAddClass 5267
ImplCanAddClass 5268
ImplGetWebComponentNodeName 5269
SetWebParent 5269
TAdapterFields Members 5266
TAdapterFields methods 5267
TAdapterFields properties 5269
WebFields 5270
TAdapterFileField class 5270
about TAdapterFileField class 5270
DisplayLabel 5276
DisplayWidth 5276
FieldName 5277
ModifyAccess 5277
OnGetFileCount 5278
OnGetFileNames 5279

OnUploadFiles 5279
OnValidateFiles 5279
ReadOnly 5277
Required 5278
TAdapterFileField events 5278
TAdapterFileField Members 5272
TAdapterFileField properties 5276
ViewAccess 5278
TAdapterForm class 6373
about TAdapterForm class 6373
Custom 6375
FormMethod 6375
Style 6376
StyleRule 6376
TAdapterForm Members 6374
TAdapterForm properties 6375
TAdapterFormLayout class 6376
about TAdapterFormLayout class 6376
BreakButtons 6385
ChildTableIndent 6385
ColumnCount 6385
Create 6378
EndButtons 6379
EndField 6380
EndFields 6380
EndLayout 6380
EndRow 6380
EndTable 6380
FieldIndent 6386
GetAdapterFormLayout 6381
GetChildTableIndent 6381
GetFieldIndent 6381
GetTableIndent 6381
ImplLayoutButton 6381
ImplLayoutField 6382
ImplLayoutLabelAndField 6382
ImplLayoutTable 6383
NextColumn 6384
StartButtons 6384
StartFields 6384
StartRow 6384
StartTable 6384
TableHeader 6386
TableIndent 6386
TAdapterFormLayout Members 6377
TAdapterFormLayout methods 6379
TAdapterFormLayout properties 6385
TAdapterGotoPageAction class 5280
about TAdapterGotoPageAction class 5280
ExecuteAccess 5285
OnAfterExecute 5286
OnAfterGetResponse 5286
OnBeforeExecute 5286
OnBeforeGetResponse 5287
OnGetPageName 5287
OnGetParams 5288
RedirectOptions 5285
TAdapterGotoPageAction events 5285
TAdapterGotoPageAction Members 5281
TAdapterGotoPageAction properties 5284
TAdapterGrid class 6386
about TAdapterGrid class 6386
Adapter 6389
AdapterMode 6390
AddDefaultFields 6390
HeadingAttributes 6391
RowAttributes 6391
TableAttributes 6392
TAdapterGrid Members 6388
TAdapterGrid properties 6389
TAdapterHiddenField class 5288
about TAdapterHiddenField class 5288
Create 5290
GetName 5290
GetValue 5290
TAdapterHiddenField Members 5288
TAdapterHiddenField methods 5290
TAdapterHiddenFieldsList class 5290

about TAdapterHiddenFieldsList class 5290
Add 5294
Clear 5294
Create 5293
Destroy 5293
EndIterator 5295
FieldCount 5296
Fields 5296
GetCurrentIntf 5295
GetField 5295
GetFieldCount 5295
NextIteration 5295
StartIterator 5296
TAdapterHiddenFieldsList Members 5291
TAdapterHiddenFieldsList methods 5293
TAdapterHiddenFieldsList properties 5296
TAdapterImageField class 5297
about TAdapterImageField class 5297
DisplayLabel 5302
DisplayWidth 5303
FieldName 5303
HREF 5303
ModifyAccess 5304
OnGetHREF 5306
OnGetImage 5306
OnGetImageName 5306
OnGetParams 5307
OnUploadFile 5307
OnValidateFile 5307
ReadOnly 5304
Required 5304
TAdapterImageField events 5305
TAdapterImageField Members 5298
TAdapterImageField properties 5302
ValuesList 5305
ViewAccess 5305
TAdapterItemRequestParams class 6134
about TAdapterItemRequestParams class 6134
Create 6136
Destroy 6137
FailurePage 6139
GetFailurePage 6137
GetParamValues 6137
GetSuccessPage 6138
ParamValues 6139
SetFailurePage 6138
SetSuccessPage 6138
SuccessPage 6140
TAdapterItemRequestParams Members 6135
TAdapterItemRequestParams methods 6137
TAdapterItemRequestParams properties 6139
TAdapterLoginField class 5308
about TAdapterLoginField class 5308
Adapter 5311
TAdapterLoginField Members 5309
TAdapterLoginField properties 5311
TAdapterMemoField class 5312
about TAdapterMemoField class 5312
DisplayLabel 5317
DisplayWidth 5317
FieldName 5317
ModifyAccess 5318
OnGetValue 5319
OnUpdateValue 5319
OnUploadFile 5320
ReadOnly 5318
Required 5318
TAdapterMemoField events 5319
TAdapterMemoField Members 5313
TAdapterMemoField properties 5316
ViewAccess 5319
TAdapterMultiValueField class 5320
about TAdapterMultiValueField class 5320
DisplayLabel 5326
DisplayWidth 5326
FieldName 5327
ModifyAccess 5327
OnGetValueCount 5329

OnGetValues 5329
OnHasValue 5330
OnUpdateValues 5330
OnValidateValues 5330
ReadOnly 5327
Required 5328
TAdapterMultiValueField events 5329
TAdapterMultiValueField Members 5322
TAdapterMultiValueField properties 5326
ValuesList 5328
ViewAccess 5328
TAdapterNamedDisplayField class 5331
about TAdapterNamedDisplayField class 5331
Create 5334
DisplayLabel 5338
DisplayWidth 5338
GetDisplayStyle 5334
GetDisplayStyleType 5335
GetInputStyle 5335
GetInputStyleType 5335
GetViewMode 5336
GetViewModeType 5336
ImplGetDisplayLabel 5337
ImplGetDisplayWidth 5337
ImplGetRequired 5337
ReadOnly 5339
Required 5339
SetDisplayLabel 5337
TAdapterNamedDisplayField Members 5332
TAdapterNamedDisplayField methods 5334
TAdapterNamedDisplayField properties 5338
TAdapterNamedField class 5339
about TAdapterNamedField class 5339
ActionOrigValue 5355
ActionValue 5356
Adapter 5356
AsFormatted 5343
CheckModifyAccess 5343
CheckViewAccess 5344
EchoActionFieldValue 5356
FieldName 5357
FullyQualifyInputName 5344
GetAdapter 5344
GetDefaultDisplayLabel 5345
GetDefaultFieldName 5345
GetDisplayLabel 5345
GetDisplayWidth 5346
GetEchoActionFieldValue 5346
GetInputName 5346
GetMaxLength 5347
GetRequired 5347
GetValue 5347
GetVariableName 5348
GetVisible 5348
GetWebSnapObjClass 5348
HasModifyAccess 5349
HasViewAccess 5349
ImplAsFormatted 5349
ImplGetDisplayLabel 5349
ImplGetDisplayWidth 5350
ImplGetEchoActionFieldValue 5350
ImplGetInputName 5350
ImplGetMaxLength 5351
ImplGetRequired 5351
ImplGetValue 5351
ImplGetVariableName 5352
ImplGetVisible 5352
ImplHasModifyAccess 5353
ImplHasViewAccess 5353
ImplSetEchoActionFieldValue 5353
ImplSetFieldName 5354
ModifyAccess 5357
ParentChanged 5354
SetEchoActionFieldValue 5354
SetFieldName 5355
TAdapterNamedField Members 5340
TAdapterNamedField methods 5342
TAdapterNamedField properties 5355

Value 5357
ViewAccess 5358
TAdapterNextPageAction class 5358
about TAdapterNextPageAction class 5358
DisplayLabel 5364
ExecuteAccess 5364
GetDefaultActionName 5363
GetDefaultDisplayLabel 5363
GetNewPage 5363
OnAfterExecute 5365
OnAfterGetResponse 5366
OnBeforeExecute 5366
OnBeforeGetResponse 5366
OnGetParams 5367
RedirectOptions 5365
TAdapterNextPageAction events 5365
TAdapterNextPageAction Members 5359
TAdapterNextPageAction methods 5362
TAdapterNextPageAction properties 5364
TAdapterNextPageField class 5367
about TAdapterNextPageField class 5367
Destroy 5371
DisplayLabel 5374
DisplayWidth 5374
FieldName 5375
GetAdapterValuesList 5372
GetInputStyleType 5372
ImplGetEchoActionFieldValue 5372
ImplGetValue 5373
ImplSetEchoActionFieldValue 5373
TAdapterNextPageField Members 5369
TAdapterNextPageField methods 5371
TAdapterNextPageField properties 5374
UpdateValue 5374
TAdapterObjectErrorsList class 5375
about TAdapterObjectErrorsList class 5375
Create 5377
ImplDefineLabel 5378
ImplGetError 5378
ImplGetErrorCount 5378
TAdapterObjectErrorsList Members 5376
TAdapterObjectErrorsList methods 5378
TAdapterPageProducer class 1076
about TAdapterPageProducer class 1076
HTMLDoc 1079
HTMLFile 1080
OnAfterGetContent 1082
OnBeforeGetContent 1082
Styles 1080
StylesFile 1081
TAdapterPageProducer events 1082
TAdapterPageProducer Members 1077
TAdapterPageProducer properties 1078
WebPageItems 1081
TAdapterPasswordField class 5379
about TAdapterPasswordField class 5379
DisplayLabel 5383
DisplayWidth 5383
FieldName 5384
GetInputStyleType 5382
ImplGetValue 5383
TAdapterPasswordField Members 5380
TAdapterPasswordField methods 5382
TAdapterPasswordField properties 5383
TAdapterPrevPageAction class 5384
about TAdapterPrevPageAction class 5384
DisplayLabel 5390
ExecuteAccess 5391
GetDefaultActionName 5389
GetDefaultDisplayLabel 5389
GetNewPage 5390
OnAfterExecute 5392
OnAfterGetResponse 5392
OnBeforeExecute 5392
OnBeforeGetResponse 5393
OnGetParams 5393
RedirectOptions 5391
TAdapterPrevPageAction events 5391

TAdapterPrevPageAction Members 5385
TAdapterPrevPageAction methods 5389
TAdapterPrevPageAction properties 5390
TAdapterSubComponent class 5394
about TAdapterSubComponent class 5394
Adapter 5406
CanAddClass 5396
EndIterator 5397
FindNamedVariable 5397
FindVariable 5398
FindVariableContainer 5398
GetCurrentObject 5399
GetDefaultWebComponents 5399
GetParentComponent 5400
GetVariable 5400
GetVariableCount 5401
GetVariablesInUse 5401
GetWebComponentNodeName 5401
HasParent 5402
ImplCanAddClass 5402
ImplGetWebComponentNodeName 5403
NextIteration 5403
NotifyAdapterChange 5404
ReadState 5404
SetDefaultWebComponents 5404
SetParentComponent 5405
SetWebParent 5405
StartIterator 5406
TAdapterSubComponent Members 5395
TAdapterSubComponent methods 5396
TAdapterSubComponent properties 5406
VariablesInUse 5407
TAdapterUpdateField class 5407
about TAdapterUpdateField class 5407
CheckValueChange 5412
ConvertValue 5413
ConvertValueCount 5413
ConvertValues 5413
Create 5411
DoOnValidateValue 5414
GetAdapterValuesList 5414
GetInputStyleType 5414
GetValueCount 5415
GetValues 5415
ImplAsFormatted 5415
ImplCheckValueChange 5416
ImplGetDisplayLabel 5416
ImplGetDisplayWidth 5416
ImplGetEchoActionFieldValue 5417
ImplGetVariableName 5417
ImplSetEchoActionFieldValue 5417
ImplSetFieldName 5417
ImplUpdateValue 5418
ImplValidateValue 5418
IsMultiValueField 5418
Notification 5419
OnGetDisplayText 5421
OnValidateValue 5421
TAdapterUpdateField events 5421
TAdapterUpdateField Members 5409
TAdapterUpdateField methods 5412
TAdapterUpdateField properties 5420
UpdateValue 5419
ValidateValue 5419
Value 5420
ValuesList 5420
TAdapterUserNameField class 5422
about TAdapterUserNameField class 5422
DisplayLabel 5426
DisplayWidth 5426
FieldName 5426
ImplGetValue 5425
TAdapterUserNameField Members 5423
TAdapterUserNameField methods 5425
TAdapterUserNameField properties 5426
TAggregatedObject class 3328
_AddRef 3330
_Release 3330

about TAggregatedObject class 3328
Controller 3331
Create 3329
QueryInterface 3330
TAggregatedObject Members 3328
TAggregatedObject methods 3330
TAggregatedObject properties 3330
TAllocatedActions class 251
about TAllocatedActions class 251
AddAction 253
Clear 253
Count 254
Create 252
Delete 253
Destroy 252
Exists 253
Find 254
GetItem 254
TAllocatedActions Members 252
TAllocatedActions methods 253
TAllocatedActions properties 254
TApplicationAdapter class 5427
about TApplicationAdapter class 5427
Actions 5432
ApplicationTitle 5432
Data 5433
OnAfterExecuteAction 5433
OnAfterGetActionResponse 5433
OnBeforeExecuteAction 5434
OnBeforeGetActionResponse 5434
OnGetActionParams 5435
TApplicationAdapter events 5433
TApplicationAdapter Members 5428
TApplicationAdapter properties 5432
TAssociateItem class 255
about TAssociateItem class 255
Associate 256
Create 255
Destroy 255
PopupTip 256
TAssociateItem Members 255
TAssociateItem properties 256
TAssociateItemList class 256
about TAssociateItemList class 256
Add 258
Create 257
Destroy 257
Find 258
Get 258
Remove 258
TAssociateItemList Members 257
TAssociateItemList methods 257
TBaseAdapterAction class 5435
about TBaseAdapterAction class 5435
ActionName 5462
Adapter 5462
CheckExecuteAccess 5440
CreateRequestContext 5440
DisplayLabel 5462
DoAfterExecuteActionRequest 5441
DoAfterGetExecuteActionResponse 5441
DoBeforeExecuteActionRequest 5442
DoBeforeGetExecuteActionResponse 5442
EndIterator 5443
Errors 5463
ExecuteAccess 5463
ExecuteActionRequest 5443
GetActionName 5444
GetAdapter 5444
GetAdapterItemRequestParams 5444
GetDefaultActionName 5445
GetDefaultDisplayLabel 5445
GetDisplayLabel 5445
GetDisplayStyle 5445
GetDisplayStyleType 5446
GetErrors 5446
GetExecuteActionResponse 5446
GetFailureOptions 5447

GetInputStyle 5447
GetIsAdapterActionList 5448
GetItemCount 5448
GetIteratorIndex 5448
GetSuccessOptions 5449
GetVariableName 5449
GetViewMode 5450
GetViewModeType 5450
GetVisible 5450
GetWebSnapObjClass 5451
HandleRequest 5451
HasExecuteAccess 5451
ImplCreateRequestContext 5452
ImplEndIterator 5452
ImplExecuteActionRequest 5452
ImplGetActionName 5453
ImplGetAdapterItemRequestParams 5453
ImplGetDisplayLabel 5453
ImplGetExecuteActionResponse 5454
ImplGetFailureOptions 5454
ImplGetIsAdapterActionList 5454
ImplGetIteratorIndex 5455
ImplGetSuccessOptions 5455
ImplGetVariableName 5455
ImplGetVisible 5456
ImplHandleRequest 5456
ImplHasExecuteAccess 5456
ImplInIterator 5457
ImplIsDefaultAction 5457
ImplNextIteration 5458
ImplStartIterator 5458
ImplWebEnabled 5458
InIterator 5459
IsDefaultAction 5459
ItemCount 5463
ItemIndex 5464
NextIteration 5460
OnAfterExecute 5465
OnAfterGetResponse 5465
OnBeforeExecute 5465
OnBeforeGetResponse 5466
OnGetParams 5466
ParentChanged 5460
RedirectOptions 5464
SetDisplayLabel 5460
StartIterator 5461
TBaseAdapterAction events 5464
TBaseAdapterAction Members 5436
TBaseAdapterAction methods 5438
TBaseAdapterAction properties 5461
WebEnabled 5461
TBaseAdapterErrorsList class 5467
about TBaseAdapterErrorsList class 5467
AddError 5470
AddError method 5469
ClearErrors 5470
DefineLabel 5470
EndIterator 5471
ErrorCount 5474
Errors 5475
GetCurrentIntf 5471
GetError 5471
GetErrorCount 5471
GetObjectErrors 5472
HasObjectErrors 5472
ImplClearErrors 5472
ImplDefineLabel 5472
ImplGetError 5473
ImplGetErrorCount 5473
ImplGetObjectErrors 5473
ImplHasObjectErrors 5473
NextIteration 5474
StartIterator 5474
TBaseAdapterErrorsList Members 5467
TBaseAdapterErrorsList methods 5469
TBaseAdapterErrorsList properties 5474
TBaseAdapterMultiValueField class 5475
about TBaseAdapterMultiValueField class 5475

Create 5480
Destroy 5480
EndValueIterator 5481
GetCurrentValue 5481
GetFieldValuesAdapter 5482
GetValueCount 5482
GetValues 5482
HasValue 5483
ImplGetCurrentValue 5483
ImplGetFieldValuesAdapter 5483
ImplHasValue 5484
NextValueIteration 5484
StartValueIterator 5484
TBaseAdapterMultiValueField Members 5477
TBaseAdapterMultiValueField methods 5481
TBaseAdapterPageProducer class 1083
about TBaseAdapterPageProducer class 1083
CanAddClass 1086
ContentFromStream 1087
Create 1085
CreatePageElements 1087
DeclareStyles 1088
DeclareWarnings 1088
Destroy 1085
DoAfterGetContent 1088
DoBeforeGetContent 1089
DoTagEvent 1089
GeneratePageElements 1090
GetContentOptions 1090
GetDefaultTemplate 1090
GetRequiredTags 1091
GetScriptEngines 1091
GetTemplateFileType 1092
GetXMLData 1092
GetXSLData 1092
HandleScriptTag 1093
HasHTMLBrowserView 1093
HasScriptView 1094
HasXMLBrowserView 1094
HasXSLBrowserView 1094
ImplCanAddClass 1095
OnAfterGetContent 1098
OnBeforeGetContent 1099
PageElements 1097
SetStyles 1095
SetStylesFile 1096
SetWebContentOptions 1096
Styles 1097
StylesFile 1098
TBaseAdapterPageProducer events 1098
TBaseAdapterPageProducer Members 1084
TBaseAdapterPageProducer methods 1085
TBaseAdapterPageProducer properties 1096
TBaseEndUserAdapter class 5485
about TBaseEndUserAdapter class 5485
CanExecuteLoginForm 5491
CanExecuteLogout 5491
CanProduceLoginPage 5492
DoOnHasRights 5492
ExecuteLogin 5492
ExecuteLoginForm 5493
ExecuteLogout 5493
GetDisplayName 5493
GetDisplayNameValue 5494
GetEndUserAdapter 5494
GetLoggedIn 5495
GetLoggedInValue 5495
GetLoginFormAction 5495
GetLogoutAction 5496
GetUserID 5496
HasExecuteRights 5496
HasModifyRights 5497
HasViewRights 5497
ImplCanAddActionClass 5497
ImplCanAddFieldClass 5498
ImplGetActionsList 5498
ImplGetDisplayName 5498
ImplGetEndUserAdapter 5499

ImplGetFieldsList 5499
ImplGetLoggedIn 5499
ImplGetLoginFormAction 5499
ImplGetLogoutAction 5500
ImplHasExecuteRights 5500
ImplHasModifyRights 5500
ImplHasViewRights 5501
ImplLogIn 5501
ImplProduceLoginPage 5501
LogIn 5502
LoginPage 5503
OnHasRights 5503
OnProduceLoginPage 5504
ProduceLoginPage 5502
TBaseEndUserAdapter events 5503
TBaseEndUserAdapter Members 5486
TBaseEndUserAdapter methods 5490
TBaseEndUserAdapter properties 5503
UserList 5503
TBaseRequestHandler class 6140
about TBaseRequestHandler class 6140
AppDispatcher 6142
HandleRequest 6141
ImplHandleRequest 6141
SetAppDispatcher 6142
TBaseRequestHandler Members 6140
TBaseRequestHandler methods 6141
TBaseRequestHandler properties 6142
TBaseSocket class 1302
about TBaseSocket class 1302
Active 1314
BlockMode 1315
BytesReceived 1315
BytesSent 1315
Close 1305
Create 1303
Destroy 1304
DoCreateHandle 1305
DoDestroyHandle 1305
DoHandleError 1305
Domain 1316
DoReceive 1306
DoSend 1306
ErrorCheck 1306
Handle 1316
Loaded 1306
MapDomain 1307
MapSockType 1307
OnCreateHandle 1317
OnDestroyHandle 1319
OnError 1320
OnReceive 1320
OnSend 1321
Open 1307
PeekBuf 1308
Protocol 1316
ReceiveBuf 1308
ReceiveIn 1310
Select 1310
SendBuf 1310
SendIn 1311
SendStream 1311
SetBytesReceived 1312
SetBytesSent 1313
SockType 1317
TBaseSocket events 1317
TBaseSocket Members 1302
TBaseSocket methods 1304
TBaseSocket properties 1313
WaitForData 1313
TBaseValuesListAdapter class 5504
about TBaseValuesListAdapter class 5504
GetListImage 5510
GetListImageOfValue 5510
GetListName 5511
GetListNameOfValue 5511
GetListValue 5511
GetNameFieldClass 5512

GetValueFieldClass 5512
GetWebSnapObjClass 5512
ImplCanAddFieldClass 5512
ImplGetFieldsList 5513
ImplGetListImage 5513
ImplGetListImageOfValue 5513
ImplGetListName 5514
ImplGetListNameOfValue 5514
ImplGetListValue 5514
TBaseValuesListAdapter Members 5506
TBaseValuesListAdapter methods 5509
TBaseValuesListField class 5515
about TBaseValuesListField class 5515
Adapter 5518
GetAdapter 5518
TBaseValuesListField Members 5516
TBaseValuesListField methods 5518
TBaseValuesListField properties 5518
TBaseWebModuleFactory class 6250
about TBaseWebModuleFactory class 6250
Create 6251
GetCacheMode 6252
GetComponentClass 6252
GetCreateMode 6253
GetIsAppModule 6253
GetModule 6253
GetModuleName 6254
PreventDestruction 6254
TBaseWebModuleFactory Members 6251
TBaseWebModuleFactory methods 6252
TBaseWebPageInfo class 6255
about TBaseWebPageInfo class 6255
Create 6256
GetIsLoginRequired 6257
GetIsPublished 6257
GetPageDescription 6257
GetPageFile 6258
GetPageHREF 6258
GetPageName 6258
GetPageTitle 6259
GetViewAccess 6259
SetFactory 6260
TBaseWebPageInfo Members 6255
TBaseWebPageInfo methods 6256
TBaseWebPageModuleFactory class 6260
about TBaseWebPageModuleFactory class 6260
Create 6261
GetCacheMode 6262
GetComponentClass 6262
GetCreateMode 6263
GetIsAppModule 6263
GetModule 6263
GetModuleName 6264
PreventDestruction 6264
TBaseWebPageModuleFactory Members 6261
TBaseWebPageModuleFactory methods 6262
tbButtonDontCare enumeration member 4636
tbButtonRoot enumeration member 4636
tbCheckBoxCheckedDisabled enumeration member 4636
tbCheckBoxCheckedHot enumeration member 4636
tbCheckBoxCheckedNormal enumeration member 4636
tbCheckBoxCheckedPressed enumeration member 4636
tbCheckBoxMixedDisabled enumeration member 4636
tbCheckBoxMixedHot enumeration member 4636
tbCheckBoxMixedNormal enumeration member 4636
tbCheckBoxMixedPressed enumeration member 4636
tbCheckBoxUncheckedDisabled enumeration member 4636
tbCheckBoxUncheckedHot enumeration member 4636
tbCheckBoxUncheckedNormal enumeration member 4636
tbCheckBoxUncheckedPressed enumeration member 4636
tbGroupBoxDisabled enumeration member 4636
tbGroupBoxNormal enumeration member 4636
TBigEndianUnicodeEncoding class 4033
about TBigEndianUnicodeEncoding class 4033
GetBytes 4034
GetChars 4035
GetPreamble 4035
TBigEndianUnicodeEncoding Members 4033

TBigEndianUnicodeEncoding methods 4034
TBinding class 6854
about TBinding class 6854
AfterConstruction 6860
GetBindingOperations 6860
GetName 6860
GetType_ 6861
SetName 6861
SetType_ 6861
TBinding Members 6855
TBinding methods 6859
TBindingOperation class 6861
about TBindingOperation class 6861
AfterConstruction 6867
GetFaults 6867
GetInput 6868
GetName 6868
GetOutput 6868
SetName 6868
TBindingOperation Members 6862
TBindingOperation methods 6867
TBindingOperations class 6869
about TBindingOperations class 6869
Add 6874
GetBindingOperation 6875
Insert 6875
TBindingOperations Members 6869
TBindingOperations methods 6874
TBindings class 6875
about TBindings class 6875
Add 6881
GetBinding 6881
Insert 6881
TBindings Members 6875
TBindings methods 6881
tbPushButtonDefaulted enumeration member 4636
tbPushButtonDisabled enumeration member 4636
tbPushButtonHot enumeration member 4636
tbPushButtonNormal enumeration member 4636
tbPushButtonPressed enumeration member 4636
tbRadioButtonCheckedDisabled enumeration member 4636
tbRadioButtonCheckedHot enumeration member 4636
tbRadioButtonCheckedNormal enumeration member 4636
tbRadioButtonCheckedPressed enumeration member 4636
tbRadioButtonUncheckedDisabled enumeration member 4636
tbRadioButtonUncheckedHot enumeration member 4636
tbRadioButtonUncheckedNormal enumeration member 4636
tbRadioButtonUncheckedPressed enumeration member 4636
TBrowseForFolder class 1696
about TBrowseForFolder class 1696
BeforeExecute 1704
BrowseOptions 1699
Caption 1699
DialogCaption 1700
DoCallback 1697
DoValidate 1697
Enabled 1700
ExecuteTarget 1698
Folder 1700
GetDialogClass 1698
HelpContext 1700
HelpKeyword 1701
HelpType 1701
Hint 1702
ImageIndex 1702
OnAccept 1704
OnCallback 1705
OnCancel 1705
OnHint 1705
OnValidate 1705
RootDir 1702
SecondaryShortCuts 1703
ShortCut 1703
TBrowseForFolder events 1704
TBrowseForFolder Members 1696
TBrowseForFolder methods 1697
TBrowseForFolder properties 1698
Visible 1703

tbUserButton enumeration member 4636
TButton class 1961
about TButton class 1961
Action 1965
Align 1966
Anchors 1969
BiDiMode 1970
Cancel 1970
Caption 1971
CommandLinkHint 1973
Constraints 1973
Default 1974
DisabledImageIndex 1975
DoubleBuffered 1976
DragCursor 1977
DragKind 1978
DragMode 1978
DropDownMenu 1980
ElevationRequired 1980
Enabled 1981
Font 1981
HotImageIndex 1982
ImageAlignment 1983
ImageIndex 1983
ImageMargins 1984
Images 1984
ModalResult 1985
OnClick 1998
OnContextPopup 2000
OnDragDrop 2001
OnDragOver 2004
OnDropDownClick 2007
OnEndDock 2008
OnEndDrag 2008
OnEnter 2009
OnExit 2011
OnKeyDown 2014
OnKeyPress 2018
OnKeyUp 2019
OnMouseActivate 2021
OnMouseDown 2022
OnMouseEnter 2024
OnMouseLeave 2024
OnMouseMove 2024
OnMouseUp 2027
OnStartDock 2029
OnStartDrag 2030
ParentBiDiMode 1985
ParentDoubleBuffered 1986
ParentFont 1986
ParentShowHint 1987
PopupMenu 1987
PressedImageIndex 1992
SelectedImageIndex 1993
ShowHint 1993
Style 1995
TabOrder 1995
TabStop 1996
TButton events 1998
TButton Members 1962
TButton properties 1964
Visible 1996
WordWrap 1997
TButtonActionLink class 2030
about TButtonActionLink class 2030
AssignClient 2032
FClient 2031
IsCheckedLinked 2032
SetChecked 2032
TButtonActionLink fields 2031
TButtonActionLink Members 2031
TButtonActionLink methods 2031
TButtonControl class 2033
about TButtonControl class 2033
ActionChange 2034
Checked 2037
ClicksDisabled 2039
Create 2034

CreateParams 2035
GetActionLinkClass 2035
GetChecked 2036
SetChecked 2036
TButtonControl Members 2033
TButtonControl methods 2034
TButtonControl properties 2036
WndProc 2036
WordWrap 2039
tcClockDontCare enumeration member 4638
tcClockRoot enumeration member 4638
tcComboBoxDontCare enumeration member 4638
tcComboBoxRoot enumeration member 4638
tcDropDownButtonDisabled enumeration member 4638
tcDropDownButtonHot enumeration member 4638
tcDropDownButtonNormal enumeration member 4638
tcDropDownButtonPressed enumeration member 4638
TCheckBox class 2040
about TCheckBox class 2040
Action 2043
Align 2043
Alignment 2047
AllowGrayed 2047
Anchors 2048
BiDiMode 2049
Caption 2049
Checked 2052
Color 2054
Constraints 2055
Ctl3D 2055
DoubleBuffered 2056
DragCursor 2057
DragKind 2058
DragMode 2058
Enabled 2060
Font 2060
OnClick 2074
OnContextPopup 2076
OnDragDrop 2077
OnDragOver 2080
OnEndDock 2083
OnEndDrag 2084
OnEnter 2085
OnExit 2087
OnKeyDown 2090
OnKeyPress 2093
OnKeyUp 2095
OnMouseActivate 2096
OnMouseDown 2097
OnMouseEnter 2100
OnMouseLeave 2100
OnMouseMove 2100
OnMouseUp 2102
OnStartDock 2105
OnStartDrag 2105
ParentBiDiMode 2061
ParentColor 2062
ParentCtl3D 2062
ParentDoubleBuffered 2062
ParentFont 2063
ParentShowHint 2063
PopupMenu 2064
ShowHint 2069
State 2070
TabOrder 2071
TabStop 2072
TCheckBox events 2074
TCheckBox Members 2040
TCheckBox properties 2042
Visible 2072
WordWrap 2073
TClassHelperBase class 3331
_Create 3333
about TClassHelperBase class 3331
FInstance 3333
TClassHelperBase fields 3333
TClassHelperBase Members 3331
TClientSocket class 14

about TClientSocket class 14
Active 18
Address 18
ClientType 19
Create 16
Destroy 16
DoActivate 17
GetClientType 17
Host 20
OnConnect 22
OnConnecting 22
OnDisconnect 23
OnError 23
OnLookup 24
OnRead 24
OnWrite 25
Port 20
Service 21
SetClientType 17
Socket 21
TClientSocket events 22
TClientSocket Members 15
TClientSocket methods 17
TClientSocket properties 18
TClientSocketThread class 1322
about TClientSocketThread class 1322
ClientSocket 1325
Create 1323
Destroy 1323
Execute 1324
ExecuteSyncProc 1324
ServerSocketThread 1326
SyncProc 1325
TClientSocketThread Members 1322
TClientSocketThread methods 1324
TClientSocketThread properties 1325
TClientWinSocket class 25
about TClientWinSocket class 25
ClientType 28
Connect 28
SetClientType 28
TClientWinSocket Members 26
TClientWinSocket methods 27
TClientWinSocket properties 28
TColorSelect class 1706
about TColorSelect class 1706
BeforeExecute 1712
Caption 1708
Dialog 1708
Enabled 1709
GetDialogClass 1707
HelpContext 1709
HelpKeyword 1709
HelpType 1710
Hint 1710
ImageIndex 1711
OnAccept 1713
OnCancel 1713
OnHint 1713
OnUpdate 1714
SecondaryShortCuts 1711
ShortCut 1712
TColorSelect events 1712
TColorSelect Members 1706
TColorSelect methods 1707
TColorSelect properties 1707
Visible 1712
TComboBox class 2106
about TComboBox class 2106
Align 2112
Anchors 2115
AutoCloseUp 2116
AutoComplete 2116
AutoCompleteDelay 2117
AutoDropDown 2117
BevelEdges 2117
BevelInner 2118
BevelKind 2118

BevelOuter 2119
BiDiMode 2119
CharCase 2120
Color 2120
Constraints 2121
Ctl3D 2121
DoubleBuffered 2122
DragCursor 2124
DragKind 2124
DragMode 2124
DropDownCount 2126
Enabled 2126
Font 2127
ImeMode 2128
ImeName 2128
ItemHeight 2129
ItemIndex 2130
Items 2131
MaxLength 2133
OnChange 2149
OnClick 2152
OnCloseUp 2153
OnContextPopup 2154
OnDblClick 2154
OnDragDrop 2155
OnDragOver 2158
OnDrawItem 2161
OnDropDown 2161
OnEndDock 2163
OnEndDrag 2163
OnEnter 2164
OnExit 2167
OnKeyDown 2170
OnKeyPress 2173
OnKeyUp 2174
OnMeasureItem 2176
OnMouseEnter 2177
OnMouseLeave 2177
OnSelect 2177
OnStartDock 2177
OnStartDrag 2178
ParentBiDiMode 2134
ParentColor 2134
ParentCtl3D 2135
ParentDoubleBuffered 2135
ParentFont 2135
ParentShowHint 2136
PopupMenu 2136
ShowHint 2141
Sorted 2143
Style 2144
TabOrder 2145
TabStop 2145
TComboBox events 2149
TComboBox Members 2108
TComboBox properties 2111
Text 2146
TextHint 2147
Visible 2147
TCommonDialogAction class 1715
about TCommonDialogAction class 1715
BeforeExecute 1721
Create 1716
Destroy 1717
DoAccept 1718
DoCancel 1718
ExecuteResult 1720
ExecuteTarget 1718
FDialog 1717
GetDialogClass 1719
HandlesTarget 1719
Notification 1720
OnAccept 1721
OnCancel 1722
SetupDialog 1720
TCommonDialogAction events 1721
TCommonDialogAction fields 1717
TCommonDialogAction Members 1716

TCommonDialogAction methods 1717
TCommonDialogAction properties 1720
TComponentFactory class 5156
about TComponentFactory class 5156
Create 5157
CreateComObject 5157
CreateInstance 5158
TComponentFactory Members 5156
TComponentFactory methods 5157
UpdateRegistry 5158
TConditionVariableCS class 3058
about TConditionVariableCS class 3058
Acquire 3060
Create 3059
FConditionVariable 3060
Release 3061
ReleaseAll 3061
TConditionVariableCS fields 3060
TConditionVariableCS Members 3058
TConditionVariableCS methods 3060
WaitFor 3062
WaitFor method 3061
TConditionVariableHelper class helper 3062
about TConditionVariableHelper class helper 3062
Create 3063
Free 3063
SleepCS 3063
TConditionVariableHelper Members 3062
TConditionVariableHelper methods 3063
Wake 3064
WakeAll 3064
TConditionVariableMutex class 3064
about TConditionVariableMutex class 3064
Acquire 3066
Create 3065
Destroy 3066
Release 3067
ReleaseAll 3067
TConditionVariableMutex Members 3064
TConditionVariableMutex methods 3066
WaitFor 3067
TConnect 4579
TConnect namespace 4579
Classes 4580
TConnect.TLocalConnection 4580
TConnect.TLocalConnection.AppServer 4591
TConnect.TLocalConnection.AS_ApplyUpdates 4583
TConnect.TLocalConnection.AS_DataRequest 4583
TConnect.TLocalConnection.AS_Execute 4584
TConnect.TLocalConnection.AS_GetParams 4585
TConnect.TLocalConnection.AS_GetProviderNames 4586
TConnect.TLocalConnection.AS_GetRecords 4586
TConnect.TLocalConnection.AS_RowRequest 4587
TConnect.TLocalConnection.Create 4581
TConnect.TLocalConnection.Destroy 4582
TConnect.TLocalConnection.GetConnected 4588
TConnect.TLocalConnection.GetProvider 4588
TConnect.TLocalConnection.GetProviderNames 4589
TConnect.TLocalConnection.GetServer 4589
TConnect.TLocalConnection.InterfaceSupportsErrorInfo 4589
TConnect.TLocalConnection.Notification 4589
TConnect.TLocalConnection.ProviderCount 4591
TConnect.TLocalConnection.Providers 4592
TConnect.TLocalConnection.RegisterProvider 4590
TConnect.TLocalConnection.SafeCallException 4590
TConnect.TLocalConnection.UnRegisterProvider 4591
TConnectionData class 1416
about TConnectionData class 1416
AddProperties 1418
BeginUpdate 1419
ConnectionName 1420
Create 1417, 1418
Create constructor 1417
DelegateConnection 1420
Destroy 1418
DriverName 1421
EndUpdate 1419
GetFullParams 1419

IsModified 1421
Properties 1421
RefreshProperties 1419
ReloadProperties 1420
TConnectionData Members 1417
TConnectionData methods 1418
TConnectionData properties 1420
UpdateProperties 1420
TContainedObject class 3333
about TContainedObject class 3333
QueryInterface 3335
TContainedObject Members 3334
TContainedObject methods 3335
TControlRequirements class 6392
about TControlRequirements class 6392
Create 6393
DefineFunction 6394
DefineVariable 6395
Destroy 6394
IsFunctionDefined 6395
IsVariableDefined 6396
NeedMultipartForm 6396
RequiresMultipartForm 6396
TControlRequirements Members 6393
TControlRequirements methods 6394
TControlRequirements properties 6396
TCriticalSection class 3068
about TCriticalSection class 3068
Acquire 3070
Create 3069
Destroy 3069
Enter 3071
FSection 3070
Leave 3071
Release 3072
TCriticalSection fields 3070
TCriticalSection Members 3068
TCriticalSection methods 3070
TryEnter 3072
TCriticalSectionHelper class helper 3072
about TCriticalSectionHelper class helper 3072
Destroy 3073
Enter 3073
Free 3074
Initialize 3074
Leave 3074
TCriticalSectionHelper Members 3073
TCriticalSectionHelper methods 3073
TryEnter 3074
tcTimeNormal enumeration member 4638
TCurrentWebPageInfo class 6623
about TCurrentWebPageInfo class 6623
Create 6625
GetHasViewAccess 6626
GetLoginRequired 6626
GetPageDescription 6626
GetPageHREF 6627
GetPageIsPublished 6627
GetPageName 6627
GetPageTitle 6628
TCurrentWebPageInfo Members 6624
TCurrentWebPageInfo methods 6626
WebPageInfo 6628
TCustomAdapter class 5519
about TCustomAdapter class 5519
Actions 5558
AddAdapterModeHiddenField 5526
AddOrigValueHiddenFields 5526
AddRecordKeysHiddenFields 5526
CanAddActionClass 5527
CanAddFieldClass 5527
CheckModifyAccess 5527
ClearAdapterValues 5528
ClearErrors 5528
Create 5523
CreateActionRequestContext 5528
CreateErrorsList 5529
Data 5558

Destroy 5523
EchoActionFieldValues 5559
EncodeActionParams 5529
EndIterator 5529
Errors 5559
ExecuteAccess 5560
ExtractRequestParams 5530
FindAction 5530
FindField 5530
FindNamedAction 5531
FindNamedField 5531
FindNamedVariable 5531
FindVariable 5532
FindVariableContainer 5532
GetActionsClass 5533
GetActionsList 5533
GetAdapterActions 5533
GetAdapterErrors 5534
GetAdapterErrorsList 5534
GetAdapterFields 5534
GetAdapterHiddenFields 5534
GetAdapterHiddenRecordFields 5535
GetChildren 5535
GetDataClass 5535
GetEchoActionFieldValues 5536
GetFieldsList 5536
GetHiddenFieldOptions 5536
GetIteratorIndex 5537
GetVariable 5537
GetVariableCount 5538
GetVariableName 5538
GetVisibleActions 5538
GetVisibleFields 5538
GetWebSnapObjClass 5539
HasExecuteAccess 5539
HasModifyAccess 5539
HasViewAccess 5539
ImplCanAddActionClass 5540
ImplCanAddFieldClass 5540
ImplClearAdapterValues 5540
ImplEndIterator 5541
ImplFindVariable 5541
ImplFindVariableContainer 5541
ImplGetActionsList 5542
ImplGetEchoActionFieldValues 5542
ImplGetFieldsList 5542
ImplGetIteratorIndex 5543
ImplGetVariable 5543
ImplGetVariableCount 5544
ImplGetVariableName 5544
ImplGetVisibleActions 5544
ImplGetVisibleFields 5544
ImplHasExecuteAccess 5545
ImplHasModifyAccess 5545
ImplHasViewAccess 5546
ImplInIterator 5546
ImplNextIteration 5546
ImplNotifyActivate 5547
ImplNotifyDeactivate 5547
ImplSetActions 5547
ImplSetData 5548
ImplSetEchoActionFieldValues 5548
ImplStartIterator 5548
InIterator 5549
IsActionInUse 5549
IsFieldInUse 5549
ListHiddenFields 5550
ListHiddenRecordFields 5550
ModifyAccess 5560
NextIteration 5551
Notification 5551
NotifyActivate 5551
NotifyAdapterChange 5551
NotifyDeactivate 5552
NotifyList 5560
OnAfterExecuteAction 5561
OnAfterGetActionResponse 5562
OnAfterUpdateFields 5562

OnAfterValidateFields 5563
OnBeforeExecuteAction 5563
OnBeforeGetActionResponse 5563
OnBeforeUpdateFields 5564
OnBeforeValidateFields 5564
OnGetActionParams 5564
OnIterateRecords 5565
SetActions 5552
SetData 5553
SetEchoActionFieldValues 5553
StartIterator 5553
TCustomAdapter events 5561
TCustomAdapter Members 5520
TCustomAdapter methods 5523
TCustomAdapter properties 5558
UpdateFieldsExecuteValues 5554
UpdateFieldsGetAnyChanges 5554
UpdateFieldsUpdateValues 5555
UpdateFieldsValidateValues 5555
UpdateRecords 5556
UpdateRecordsGetAnyChanges 5556
UpdateRecordsUpdateValues 5557
UpdateRecordsValidateValues 5557
ViewAccess 5560
VisibleActions 5561
VisibleFields 5561
WebComponentsChange 5557
TCustomAdapterAction class 5565
about TCustomAdapterAction class 5565
ActionName 5572
ImplExecuteActionRequest 5570
ImplGetActionName 5570
ImplGetAdapterItemRequestParams 5570
ImplGetSuccessOptions 5571
ImplSetActionName 5571
ImplWebEnabled 5571
OnAfterExecute 5573
OnAfterGetResponse 5573
OnBeforeExecute 5574
OnBeforeGetResponse 5574
OnExecute 5574
OnGetEnabled 5575
SetActionName 5572
TCustomAdapterAction events 5573
TCustomAdapterAction Members 5566
TCustomAdapterAction methods 5569
TCustomAdapterAction properties 5572
TCustomAdapterActionButton class 6397
about TCustomAdapterActionButton class 6397
ActionName 6405
Align 6405
Create 6398
Destroy 6399
DisplayColumns 6405
DisplayType 6406
ErrorPageName 6406
FindAssociatedAction 6400
GetActionName 6400
GetAdapterModeOfAdapter 6400
GetAdapterVariableName 6400
GetCaption 6401
GetControlRequirements 6401
GetDesigntimeWarnings 6401
GetHTMLFormVarName 6401
GetLayoutAttributes 6402
HideOptions 6407
ImageHeight 6407
ImageProducer 6408
ImageWidth 6408
ImplContent 6402
ImplGetControlRequirements 6402
ImplRestoreDefaults 6403
Notification 6403
PageName 6408
RestoreDefaults 6403
SetActionName 6404
SetCaption 6404
TCustomAdapterActionButton Members 6398

TCustomAdapterActionButton methods 6399
TCustomAdapterActionButton properties 6404
VAlign 6409
TCustomAdapterBooleanField class 5575
about TCustomAdapterBooleanField class 5575
ConvertValue 5580
GetDisplayStyleType 5580
GetInputStyleType 5581
ImplGetValue 5581
ImplIsEqual 5582
ImplUpdateValue 5582
IsEqual 5582
OnGetValue 5583
OnUpdateValue 5583
TCustomAdapterBooleanField events 5583
TCustomAdapterBooleanField Members 5577
TCustomAdapterBooleanField methods 5580
TCustomAdapterCommandColumn class 6409
about TCustomAdapterCommandColumn class 6409
AdapterActionsListHelper 6416
AddDefaultCommands 6416
Align 6417
Caption 6417
CaptionAttributes 6418
Create 6411
Custom 6418
Destroy 6412
DisplayColumns 6418
FormatColumnData 6412
FormatColumnHeading 6413
GetAdapterDisplayCharacteristics 6413
GetLayoutAttributes 6413
ImplCanAddClass 6414
ImplContent 6414
NotifyAdapterChange 6415
Style 6419
StyleRule 6419
TCustomAdapterCommandColumn Members 6410
TCustomAdapterCommandColumn methods 6412
TCustomAdapterCommandColumn properties 6416
VAlign 6420
VisibleCommands 6420
TCustomAdapterCommandGroup class 6421
about TCustomAdapterCommandGroup class 6421
Adapter 6426
AdapterActionsListHelper 6426
AddDefaultCommands 6426
Create 6423
Custom 6426
Destroy 6423
GetAdapter 6424
GetAdapterFormLayout 6424
GetAdapterModeOfAdapter 6424
GetAdapterModeOfDisplay 6424
ImplCanAddClass 6425
ImplContent 6425
Style 6427
StyleRule 6427
TCustomAdapterCommandGroup Members 6421
TCustomAdapterCommandGroup methods 6423
TCustomAdapterCommandGroup properties 6425
VisibleCommands 6427
TCustomAdapterDispatcher class 6142
about TCustomAdapterDispatcher class 6142
AddSpecialParams 6144
DoAdapterRequestNotHandled 6145
DoAfterDispatchAdapterRequest 6146
DoBeforeDispatchAdapterRequest 6146
EncodeAdapterItemRequestAsFieldValue 6147
EncodeAdapterItemRequestAsHREF 6147
GetAdapterItemRequestParams 6148
GetDispatchParams 6148
ImplEncodeAdapterItemRequestAsFieldValue 6149
ImplEncodeAdapterItemRequestAsHREF 6149
ImplGetAdapterItemRequestParams 6150
ImplHandleRequest 6150
OnActionRequestNotHandled 6151
OnAfterDispatchAction 6152

OnAfterDispatchImage 6152
OnBeforeDispatchAction 6153
OnBeforeDispatchImage 6153
OnGetAdapterRequestParams 6154
OnImageRequestNotHandled 6154
TCustomAdapterDispatcher events 6151
TCustomAdapterDispatcher Members 6143
TCustomAdapterDispatcher methods 6144
TCustomAdapterDisplayColumn class 6428
about TCustomAdapterDisplayColumn class 6428
CanAddClass 6430
ControlContent 6431
Create 6430
DisplayType 6433
FormatImage 6431
FormatListDisplay 6431
FormatTextDisplay 6431
ImageHeight 6433
ImageWidth 6434
ImplCanAddClass 6432
ImplFormatColumnHeading 6432
ImplGetControlRequirements 6432
ImplRestoreDefaults 6432
IsColumn 6432
TCustomAdapterDisplayColumn Members 6428
TCustomAdapterDisplayColumn methods 6430
TCustomAdapterDisplayColumn properties 6433
TCustomAdapterDisplayField class 6434
about TCustomAdapterDisplayField class 6434
AddAttributes 6437
ControlContent 6437
Create 6436
DisplayColumns 6439
DisplayRows 6439
DisplayType 6439
DisplayWidth 6440
GetLayoutAttributes 6437
ImageHeight 6440
ImageWidth 6440
ImplContent 6437
ImplGetControlRequirements 6438
ImplRestoreDefaults 6438
InputType 6441
MaxLength 6441
SelectRows 6441
TCustomAdapterDisplayField Members 6435
TCustomAdapterDisplayField methods 6436
TCustomAdapterDisplayField properties 6438
TextAreaWrap 6441
ViewMode 6442
TCustomAdapterEditColumn class 6442
about TCustomAdapterEditColumn class 6442
ImplContent 6444
TCustomAdapterEditColumn Members 6443
TCustomAdapterEditColumn methods 6444
TCustomAdapterErrorList class 6445
about TCustomAdapterErrorList class 6445
Adapter 6448
Content 6447
Create 6446
Destroy 6446
GetDesigntimeWarnings 6447
ImplContent 6448
Notification 6448
TCustomAdapterErrorList Members 6445
TCustomAdapterErrorList methods 6447
TCustomAdapterErrorList properties 6448
TCustomAdapterField class 5583
about TCustomAdapterField class 5583
DoOnValidateValue 5588
ImplGetValue 5588
ImplUpdateValue 5589
OnGetValue 5589
OnUpdateValue 5590
TCustomAdapterField events 5589
TCustomAdapterField Members 5584
TCustomAdapterField methods 5588
TCustomAdapterFieldGroup class 6449

about TCustomAdapterFieldGroup class 6449
Create 6451
Custom 6455
Destroy 6451
GetAdapterDisplayCharacteristics 6452
GetAdapterFormLayout 6452
GetAddAllFieldsItem 6453
GetAddFieldsDlgCaption 6453
GetAddFieldsItem 6453
GetLayoutAttributes 6453
GetNewFieldPrefix 6454
ImplCanAddClass 6454
ImplContent 6454
Style 6455
StyleRule 6456
TCustomAdapterFieldGroup Members 6450
TCustomAdapterFieldGroup methods 6452
TCustomAdapterFieldGroup properties 6455
TCustomAdapterFileField class 5590
about TCustomAdapterFileField class 5590
DoOnValidateValue 5595
FileCount 5598
FileNames 5598
GetDisplayStyleType 5595
GetFileNames 5596
GetInputStyleType 5596
GetValueCount 5596
GetValues 5597
ImplCheckValueChange 5597
ImplUpdateValue 5597
OnGetFileCount 5599
OnGetFileNames 5599
OnUploadFiles 5599
OnValidateFiles 5600
TCustomAdapterFileField events 5599
TCustomAdapterFileField Members 5591
TCustomAdapterFileField methods 5594
TCustomAdapterFileField properties 5598
TCustomAdapterForm class 6456
about TCustomAdapterForm class 6456
Create 6457
Custom 6461
Destroy 6458
FormMethod 6461
GetAdapterFormLayout 6458
GetFormRequirements 6458
GetHTMLFormName 6459
GetHTMLFormVarName 6459
GetLayoutAttributes 6459
ImplCanAddClass 6460
ImplContent 6460
Style 6462
StyleRule 6462
TCustomAdapterForm Members 6457
TCustomAdapterForm methods 6458
TCustomAdapterForm properties 6461
TCustomAdapterGotoPageAction class 5600
about TCustomAdapterGotoPageAction class 5600
GetDefaultActionName 5605
GetDisplayStyleType 5605
GetItemCount 5606
ImplExecuteActionRequest 5606
ImplGetAdapterItemRequestParams 5606
ImplGetDisplayLabel 5607
ImplGetIsAdapterActionList 5607
ImplIsDefaultAction 5608
ImplWebEnabled 5608
OnGetPageName 5608
TCustomAdapterGotoPageAction events 5608
TCustomAdapterGotoPageAction Members 5601
TCustomAdapterGotoPageAction methods 5605
TCustomAdapterGrid class 6462
about TCustomAdapterGrid class 6462
Create 6465
Destroy 6465
GetAdapterDisplayCharacteristics 6466
GetAdapterFormLayout 6466
GetAddAllFieldsItem 6466

GetAddFieldsDlgCaption 6467
GetAddFieldsItem 6467
GetNewFieldPrefix 6467
HeadingAttributes 6468
ImplCanAddClass 6467
ImplContent 6468
RowAttributes 6469
TableAttributes 6469
TCustomAdapterGrid Members 6464
TCustomAdapterGrid methods 6466
TCustomAdapterGrid properties 6468
TCustomAdapterImageField class 5609
about TCustomAdapterImageField class 5609
CreateRequestContext 5614
DoOnValidateValue 5614
GetAdapterImage 5615
GetAdapterItemRequestParams 5615
GetDisplayStyleType 5616
GetInputStyleType 5616
HandleRequest 5616
HREF 5620
ImplCheckValueChange 5617
ImplCreateRequestContext 5617
ImplGetValue 5618
ImplHandleRequest 5618
ImplUpdateValue 5619
ImplWebImageHREF 5619
OnGetHREF 5621
OnGetImage 5622
OnGetImageName 5622
OnGetParams 5622
OnUploadFile 5623
OnValidateFile 5623
ReadOnly 5621
RenderAdapterImage 5619
TCustomAdapterImageField events 5621
TCustomAdapterImageField Members 5610
TCustomAdapterImageField methods 5613
TCustomAdapterImageField properties 5620
WebImageHREF 5620
TCustomAdapterMemoField class 5624
about TCustomAdapterMemoField class 5624
DoOnValidateValue 5628
GetDisplayStyleType 5628
GetInputStyleType 5629
ImplCheckValueChange 5629
ImplGetValue 5630
ImplUpdateValue 5630
OnGetValue 5631
OnUpdateValue 5631
OnUploadFile 5631
TCustomAdapterMemoField events 5630
TCustomAdapterMemoField Members 5625
TCustomAdapterMemoField methods 5628
TCustomAdapterMultiValueField class 5632
about TCustomAdapterMultiValueField class 5632
DoOnValidateValue 5637
GetDisplayStyleType 5637
GetInputStyleType 5637
GetValueCount 5638
GetValues 5638
ImplHasValue 5639
ImplUpdateValue 5639
OnGetValueCount 5640
OnGetValues 5641
OnHasValue 5641
OnUpdateValues 5642
OnValidateValues 5642
TCustomAdapterMultiValueField events 5640
TCustomAdapterMultiValueField Members 5633
TCustomAdapterMultiValueField methods 5636
TCustomAdapterMultiValueField properties 5639
ValueCount 5639
Values 5640
TCustomAdapterPageAction class 5642
about TCustomAdapterPageAction class 5642
GetDisplayStyleType 5647
GetNewPage 5647

ImplExecuteActionRequest 5648
ImplGetAdapterItemRequestParams 5648
ImplIsDefaultAction 5649
ImplWebEnabled 5649
TCustomAdapterPageAction Members 5643
TCustomAdapterPageAction methods 5647
TCustomAdapterPageProducer class 1099
about TCustomAdapterPageProducer class 1099
Create 1101
Destroy 1102
GetTemplateFileName 1102
GetTemplateStream 1103
HTMLDoc 1105
HTMLFile 1106
SetHTMLDoc 1103
SetHTMLFile 1104
TCustomAdapterPageProducer Members 1100
TCustomAdapterPageProducer methods 1102
TCustomAdapterPageProducer properties 1105
WebContent 1104
TCustomApplicationAdapter class 5649
about TCustomApplicationAdapter class 5649
ApplicationTitle 5657
GetApplicationAdapter 5655
GetApplicationTitle 5655
GetWebSnapObjClass 5655
ImplCanAddActionClass 5655
ImplCanAddFieldClass 5656
ImplGetActionsList 5656
ImplGetFieldsList 5656
TCustomApplicationAdapter Members 5650
TCustomApplicationAdapter methods 5654
TCustomApplicationAdapter properties 5657
TCustomApplicationAdapterAction class 5657
about TCustomApplicationAdapterAction class 5657
Adapter 5662
TCustomApplicationAdapterAction Members 5658
TCustomApplicationAdapterAction properties 5662
TCustomButton class 2179
about TCustomButton class 2179
ActionChange 2182
Cancel 2187
Click 2183
CommandLinkHint 2187
Create 2180
CreateParams 2183
CreateWnd 2184
Default 2188
Destroy 2181
DisabledImageIndex 2189
DropDownMenu 2189
ElevationRequired 2190
GetActionLinkClass 2184
HotImageIndex 2190
ImageAlignment 2191
ImageIndex 2191
ImageMargins 2192
Images 2192
Loaded 2184
ModalResult 2193
Notification 2185
OnDropDownClick 2195
PressedImageIndex 2193
SelectedImageIndex 2194
SetButtonStyle 2185
Style 2194
TabStop 2195
TCustomButton enumerations 2181
TCustomButton events 2195
TCustomButton Members 2179
TCustomButton methods 2182
TCustomButton properties 2186
UpdateImageList 2186
UpdateImages 2186
UseRightToLeftAlignment 2186
TCustomCheckBox class 2196
about TCustomCheckBox class 2196
Alignment 2200

AllowGrayed 2201
Click 2197
Create 2197
CreateParams 2198
CreateWnd 2198
GetChecked 2199
GetControlsAlignment 2199
SetChecked 2199
State 2201
TabStop 2202
TCustomCheckBox Members 2196
TCustomCheckBox methods 2197
TCustomCheckBox properties 2200
Toggle 2200
TCustomCombo class 2203
about TCustomCombo class 2203
AddItem 2209
AdjustDropDown 2211
Canvas 2224
Change 2211
Clear 2212
ClearSelection 2212
CloseUp 2212
ComboWndProc 2213
CopySelection 2213
Create 2205
CreateWnd 2214
DeleteSelected 2214
Destroy 2205
DestroyWindowHandle 2214
DropDown 2215
DropDownCount 2225
DroppedDown 2225
EditHandle 2225
EditWndProc 2215
FDefEditProc 2206
FDefListProc 2206
FDropHandle 2207
FDroppingDown 2207
FEditHandle 2207
FEditInstance 2207
FFocusChanged 2207
FIsFocused 2208
FListHandle 2208
FListInstance 2208
Focused 2215
FSaveIndex 2208
GetCount 2216
GetDroppedDown 2216
GetItemCount 2216
GetItemHt 2217
GetItemIndex 2217
GetItemsClass 2217
GetSelLength 2218
GetSelStart 2218
ItemCount 2226
ItemHeight 2226
Items 2227
ListHandle 2229
ListWndProc 2218
Loaded 2219
MaxLength 2229
OnChange 2237
OnCloseUp 2239
OnDropDown 2239
OnSelect 2241
ParentColor 2230
Select 2219
SelectAll 2220
SelLength 2230
SelStart 2233
SetDropDownCount 2220
SetDroppedDown 2220
SetItemHeight 2221
SetItemIndex 2221
SetItems 2221
SetMaxLength 2222
SetSelLength 2222

SetSelStart 2222
TabStop 2236
TCustomCombo events 2236
TCustomCombo fields 2206
TCustomCombo Members 2203
TCustomCombo methods 2208
TCustomCombo properties 2223
WndProc 2223
TCustomComboBox class 2241
about TCustomComboBox class 2241
AutoCloseUp 2250
AutoComplete 2251
AutoCompleteDelay 2252
AutoDropDown 2253
CharCase 2253
Create 2244
CreateParams 2245
CreateWnd 2245
Destroy 2244
DestroyWnd 2246
DoSetTextHint 2246
DrawItem 2246
GetItemCount 2247
GetItemHt 2247
GetItemsClass 2247
GetSelText 2248
KeyPress 2248
MeasureItem 2249
OnDrawItem 2256
OnMeasureItem 2257
SelectItem 2249
SelText 2253
SetStyle 2249
Sorted 2254
Style 2255
TCustomComboBox events 2256
TCustomComboBox Members 2242
TCustomComboBox methods 2245
TCustomComboBox properties 2250
TextHint 2256
WndProc 2250
TCustomComboBoxStrings class 2257
about TCustomComboBoxStrings class 2257
Clear 2259
ComboBox 2263
Create 2258
Delete 2260
Destroy 2259
Get 2260
GetCount 2261
GetObject 2261
IndexOf 2261
PutObject 2262
SetUpdateState 2262
TCustomComboBoxStrings Members 2258
TCustomComboBoxStrings methods 2259
TCustomComboBoxStrings properties 2263
TCustomDataBlockInterpreter class 148
about TCustomDataBlockInterpreter class 148
AddDispatch 150
CallCreateObject 150
CallFreeObject 150
CallGetIDsOfNames 150
CallGetServerList 151
CallInvoke 151
CanCreateObject 151
CreateObject 151
InternalCreateObject 151
InterpretData 152
LockObject 152
ReleaseObject 152
RemoveDispatch 152
StoreObject 153
TCustomDataBlockInterpreter Members 149
TCustomDataBlockInterpreter methods 149
UnlockObject 153
TCustomEdit class 2263
about TCustomEdit class 2263

Alignment 2283
AutoSelect 2284
AutoSize 2284
BorderStyle 2285
CanUndo 2285
Change 2266
CharCase 2286
Clear 2267
ClearSelection 2267
ClearUndo 2268
CopyToClipboard 2268
Create 2265
CreateParams 2269
CreateWindowHandle 2270
CreateWnd 2270
CutToClipboard 2271
DefaultHandler 2272
DestroyWnd 2273
DoSetMaxLength 2273
DoSetTextHint 2273
GetControlsAlignment 2274
GetSelLength 2274
GetSelStart 2274
GetSelText 2275
GetSelTextBuf 2275
HideSelection 2286
MaxLength 2286
Modified 2287
NumbersOnly 2287
OEMConvert 2288
OnChange 2299
ParentColor 2288
PasswordChar 2289
PasteFromClipboard 2276
ReadOnly 2289
SelectAll 2278
SelLength 2289
SelStart 2293
SelText 2296
SetAlignment 2279
SetAutoSize 2279
SetSelLength 2280
SetSelStart 2280
SetSelText 2281
SetSelTextBuf 2281
TabStop 2297
TCustomEdit events 2299
TCustomEdit Members 2264
TCustomEdit methods 2266
TCustomEdit properties 2282
Text 2297
TextHint 2299
Undo 2282
TCustomEndUserAdapter class 5662
about TCustomEndUserAdapter class 5662
ExecuteLogin 5669
ExecuteLogout 5669
GetDisplayNameValue 5669
GetLoggedInValue 5670
GetUserID 5670
OnGetDisplayName 5671
OnGetUserID 5671
OnIsLoggedIn 5672
OnLogin 5672
OnLogout 5673
TCustomEndUserAdapter events 5671
TCustomEndUserAdapter Members 5664
TCustomEndUserAdapter methods 5668
TCustomEndUserAdapterAction class 5673
about TCustomEndUserAdapterAction class 5673
Adapter 5678
TCustomEndUserAdapterAction Members 5674
TCustomEndUserAdapterAction properties 5677
TCustomEndUserSessionAdapter class 5678
about TCustomEndUserSessionAdapter class 5678
ExecuteLogin 5685
ExecuteLogout 5685
GetDisplayNameValue 5685

GetLoggedInValue 5686
GetUserID 5686
TCustomEndUserSessionAdapter Members 5680
TCustomEndUserSessionAdapter methods 5684
TCustomEndUserSessionAdapter properties 5686
UserID 5687
TCustomGroupBox class 2299
about TCustomGroupBox class 2299
AdjustClientRect 2301
Create 2300
CreateParams 2301
Paint 2302
TCustomGroupBox Members 2300
TCustomGroupBox methods 2301
TCustomIpClient class 1326
about TCustomIpClient class 1326
Close 1329
Connect 1329
Connected 1331
Create 1328
Disconnect 1329
DoConnect 1330
DoDisconnect 1330
GetThreadObject 1330
OnConnect 1332
OnDisconnect 1332
Open 1330
TCustomIpClient events 1331
TCustomIpClient Members 1326
TCustomIpClient methods 1328
TCustomIpClient properties 1331
TCustomLabel class 2302
about TCustomLabel class 2302
AdjustBounds 2304
Alignment 2307
AutoSize 2307
Canvas 2308
Caption 2310
Create 2303
DoDrawText 2304
EllipsisPosition 2313
FocusControl 2313
GetLabelText 2305
GlowSize 2314
Layout 2314
Loaded 2305
Notification 2305
Paint 2306
SetAutoSize 2306
ShowAccelChar 2314
TCustomLabel Members 2303
TCustomLabel methods 2304
TCustomLabel properties 2306
Transparent 2315
WordWrap 2316
TCustomListBox class 2316
about TCustomListBox class 2316
AddItem 2320
AutoComplete 2340
AutoCompleteDelay 2341
BorderStyle 2341
Canvas 2341
Clear 2321
ClearSelection 2321
Columns 2344
CopySelection 2322
Count 2344
Create 2318
CreateParams 2322
CreateWnd 2323
DeleteSelected 2324
DeleteString 2324
Destroy 2319
DestroyWnd 2324
DoFindData 2325
DoGetData 2325
DoGetDataObject 2326
DragCanceled 2326

DrawItem 2326
ExtendedSelect 2345
FMoving 2319
GetCount 2327
GetItemData 2327
GetItemIndex 2328
GetSelCount 2328
GetSelected 2328
IntegralHeight 2345
InternalGetItemData 2328
InternalSetItemData 2329
ItemAtPos 2329
ItemHeight 2346
ItemRect 2331
Items 2346
KeyPress 2332
LoadRecreateItems 2333
MeasureItem 2333
OnData 2356
OnDataFind 2356
OnDataObject 2357
OnDrawItem 2357
OnMeasureItem 2359
ParentColor 2349
ResetContent 2334
SaveRecreateItems 2334
ScrollWidth 2349
SelectAll 2334
Selected 2350
SetItemData 2334
SetItemIndex 2335
SetMultiSelect 2335
Sorted 2352
Style 2353
TabStop 2354
TabWidth 2354
TCustomListBox events 2355
TCustomListBox fields 2319
TCustomListBox Members 2317
TCustomListBox methods 2320
TCustomListBox properties 2340
TopIndex 2354
WndProc 2335
TCustomLocateFileService class 6155
about TCustomLocateFileService class 6155
FindIncludeFile 6156
FindStream 6157
FindTemplateFile 6157
GetFileStream 6158
GetTemplateStream 6158
ImplFindIncludeFile 6159
ImplGetTemplateStream 6159
OnFindIncludeFile 6160
OnFindStream 6161
OnFindTemplateFile 6161
TCustomLocateFileService events 6160
TCustomLocateFileService Members 6155
TCustomLocateFileService methods 6156
TCustomLoginFormAdapter class 5687
about TCustomLoginFormAdapter class 5687
EndUser 5693
ExecuteLogin 5693
GetNextPage 5694
ImplCanAddActionClass 5694
ImplCanAddFieldClass 5695
ImplGetActionsList 5695
ImplGetFieldsList 5696
ImplNotifyDeactivate 5696
NextPage 5699
OnLogin 5701
OnValidatePassword 5702
OnValidateUserName 5702
Password 5700
PasswordRequired 5700
RaiseBlankPassword 5697
RaiseBlankUserName 5697
TCustomLoginFormAdapter events 5701
TCustomLoginFormAdapter Members 5688

TCustomLoginFormAdapter methods 5692
TCustomLoginFormAdapter properties 5699
UserList 5697
UserName 5700
ValidatePassword 5698
ValidateUserName 5698
TCustomLoginFormAdapterAction class 5703
about TCustomLoginFormAdapterAction class 5703
Adapter 5707
TCustomLoginFormAdapterAction Members 5704
TCustomLoginFormAdapterAction properties 5707
TCustomMemo class 2361
about TCustomMemo class 2361
CaretPos 2368
Create 2363
CreateParams 2364
CreateWindowHandle 2365
Destroy 2364
GetCaretPos 2365
KeyPress 2366
Lines 2368
Loaded 2366
ScrollBars 2369
SetCaretPos 2366
SetLines 2367
SetScrollBars 2367
SetWordWrap 2367
TCustomMemo Members 2361
TCustomMemo methods 2364
TCustomMemo properties 2368
WantReturns 2372
WantTabs 2372
WordWrap 2372
TCustomPagedAdapter class 5707
about TCustomPagedAdapter class 5707
TCustomPagedAdapterAction class 5708
about TCustomPagedAdapterAction class 5708
Adapter 5713
GetAdapter 5713
TCustomPagedAdapterAction Members 5709
TCustomPagedAdapterAction methods 5713
TCustomPagedAdapterAction properties 5713
TCustomPageDispatcher class 6162
about TCustomPageDispatcher class 6162
DefaultPage 6169
DispatchPageName 6164
EndUser 6164
GetCanView 6165
GetDefaultPageName 6165
GetLoginRequired 6165
GetRequestedPage 6166
ImplDispatchPageName 6166
ImplGetCanView 6166
ImplGetDefaultPageName 6166
ImplGetLoginRequired 6167
ImplHandleRequest 6167
ImplPageNameOfRequest 6167
ImplRedirectToPageName 6167
IsLoggedIn 6168
OnAfterDispatchPage 6170
OnAfterRedirectToPage 6171
OnBeforeDispatchPage 6171
OnBeforeRedirectToPage 6172
OnCanViewPage 6172
OnIsLoginRequired 6173
OnPageAccessDenied 6174
OnPageNotFound 6174
PageNameOfRequest 6168
ProduceLoginPage 6168
RedirectToPageName 6168
TCustomPageDispatcher events 6170
TCustomPageDispatcher Members 6162
TCustomPageDispatcher methods 6163
TCustomPageDispatcher properties 6169
ValidatePage 6169
TCustomScreenTipsManager class 259
about TCustomScreenTipsManager class 259
AddPopup 262

BorderStyle 267
ClearScreenTips 262
Create 260
CreateScreenTip 262
Destroy 261
DisplayOffset 267
DoCreateScreenTip 263
DoDuplicateCreateError 263
DoGetScreenTipClass 263
DoScreenTipsLoaded 263
DoShow 263
DoShowScreenTip 264
Font 267
Footer 267
FooterImage 268
GenerateScreenTips 264
GetScreenTip 264
GradientEndColor 268
GradientStartColor 268
LinkedActionLists 268
Loaded 264
Notification 265
OnCreateScreenTip 269
OnDuplicateCreateError 269
OnGetScreenTipWinClass 270
OnScreenTipsLoaded 270
OnShowScreenTip 270
Options 268
RemovePopup 265
RemoveScreenTip 266
ScreenTips 269
TCustomScreenTipsManager events 269
TCustomScreenTipsManager Members 259
TCustomScreenTipsManager methods 261
TCustomScreenTipsManager properties 266
UpdateActionInfo 266
UpdateOptions 266
TCustomScreenTipsPopup class 270
about TCustomScreenTipsPopup class 270
Associate 275
Create 271
Destroy 272
GetScreenTipItem 273
Glyph 275
LinkPopupToManager 273
Notification 273
Paint 274
PopupType 275
RemoveLinkToManager 274
ScreenTip 276
ScreenTipManager 276
ShowHint 276
TCustomScreenTipsPopup Members 271
TCustomScreenTipsPopup methods 272
TCustomScreenTipsPopup properties 275
Transparent 278
UpdateAssociateToManager 274
TCustomServerSocket class 29
about TCustomServerSocket class 29
Destroy 31
DoActivate 32
FServerSocket 32
GetGetSocketEvent 33
GetGetThreadEvent 33
GetOnClientError 34
GetOnClientEvent 34
GetOnThreadEnd 34
GetOnThreadStart 35
GetServerType 35
GetThreadCacheSize 35
OnClientConnect 41
OnClientDisconnect 42
OnClientError 42
OnClientRead 43
OnClientWrite 43
OnGetSocket 44
OnGetThread 44
OnThreadEnd 45

OnThreadStart 45
ServerType 39
SetGetSocketEvent 36
SetGetThreadEvent 36
SetOnClientError 37
SetOnClientEvent 37
SetOnThreadEnd 37
SetOnThreadStart 38
SetServerType 38
SetThreadCacheSize 39
TCustomServerSocket events 40
TCustomServerSocket fields 32
TCustomServerSocket Members 30
TCustomServerSocket methods 32
TCustomServerSocket properties 39
ThreadCacheSize 40
TCustomSessionsService class 6712
about TCustomSessionsService class 6712
CheckStatusChange 6714
Clear 6715
Connect 6715
ConnectActive 6716
Create 6713
DefaultTimeout 6720
DoEndSession 6716
DoStartSession 6716
GetSessionIDs 6717
LoadFromFile 6717
LoadFromStream 6718
MaxSessions 6720
NewSession 6718
NotifyStatusChange 6719
OnEndSession 6721
OnStartSession 6721
SaveToFile 6719
SaveToStream 6719
TCustomSessionsService events 6721
TCustomSessionsService Members 6713
TCustomSessionsService methods 6714
TCustomSessionsService properties 6720
TCustomSocket class 46
about TCustomSocket class 46
Error 48
Event 48
OnAccept 49
OnConnect 50
OnConnecting 50
OnDisconnect 51
OnError 51
OnListen 51
OnLookup 52
OnRead 53
OnWrite 53
TCustomSocket events 49
TCustomSocket Members 46
TCustomSocket methods 47
TCustomSQLDataSet class 1421
about TCustomSQLDataSet class 1421
Active 1453
AddFieldDesc 1428
AddIndexDefs 1428
AfterClose 1468
AfterOpen 1469
AfterRefresh 1469
AfterScroll 1469
BeforeClose 1469
BeforeOpen 1469
BeforeRefresh 1470
BeforeScroll 1470
BlobBuffer 1454
Check 1428
CheckPrepareError 1428
ClearIndexDefs 1429
CloseCursor 1429
CloseStatement 1429
CommandText 1455
CommandType 1456
Create 1425

CreateBlobStream 1429
CurrentBlobSize 1456
DataLink 1457
DataSource 1457
DefineProperties 1430
DesignerData 1458
Destroy 1426
ExecSQL 1430
ExecuteStatement 1430
FreeBuffers 1431
FreeCursor 1431
FreeStatement 1431
GetBlobFieldData 1431
GetCanModify 1432
GetDataSource 1432
GetDetailLinkFields 1432
GetFieldData 1433
GetFieldData method 1433
GetKeyFieldNames 1434
GetKeyFieldNames method 1433
GetMetadata 1458
GetObjectTypeNames 1434
GetOutputParams 1434
GetQuoteChar 1434
GetRecord 1435
GetRecordCount 1435
GetSortFieldNames 1435
IndexDefs 1459
InitRecord 1435
InternalClose 1436
InternalConnection 1459
InternalHandleException 1436
InternalInitFieldDefs 1436
InternalOpen 1436
InternalRefresh 1436
IsCursorOpen 1437
IsSequenced 1437
LastError 1460
Loaded 1437
Locate 1438
LocateRecord 1438
Lookup 1439
MaxBlobSize 1460
NativeCommand 1460
NumericMapping 1461
ObjectView 1461
OnCalcFields 1470
OpenCursor 1439
OpenSchema 1439
ParamByName 1440
ParamCheck 1462
ParamCount 1462
Params 1463
ParseDeleteSql 1470
ParseInsertSql 1470
ParseSelectSql 1470
ParseUpdateSql 1471
Prepared 1464
PrepareStatement 1440
ProcParams 1464
PropertyChanged 1440
PSEndTransaction 1440
PSExecute 1441
PSExecuteStatement 1441
PSGetAttributes 1442
PSGetCommandText 1442
PSGetCommandType 1443
PSGetDefaultOrder 1443
PSGetIndexDefs 1443
PSGetKeyFieldsW 1444
PSGetParams 1444
PSGetQuoteCharW 1445
PSGetTableNameW 1445
PSGetUpdateException 1445
PSInTransaction 1445
PSIsSQLBased 1446
PSIsSQLSupported 1446
PSReset 1447

PSSetCommandText 1447
PSSetParams 1448
PSStartTransaction 1448
PSUpdateRecord 1448
RecordCount 1465
RowsAffected 1465
SchemaInfo 1466
SchemaName 1466
SetBufListSize 1449
SetCommandText 1449
SetFCommandText 1449
SetFieldData 1450
SetMaxBlobSize 1450
SetParamsFromCursor 1450
SetSchemaInfo 1450
SetSortFieldNames 1451
SortFieldNames 1466
SQLConnection 1467
SQLError 1452
TCustomSQLDataSet events 1468
TCustomSQLDataSet Members 1422
TCustomSQLDataSet methods 1426
TCustomSQLDataSet properties 1452
TransactionLevel 1467
UpdateIndexDefs 1452
TCustomStaticText class 2375
about TCustomStaticText class 2375
Alignment 2378
AutoSize 2379
BorderStyle 2379
Create 2376
CreateParams 2376
FocusControl 2379
Loaded 2377
Notification 2377
SetAutoSize 2378
ShowAccelChar 2380
TCustomStaticText Members 2375
TCustomStaticText methods 2376
TCustomStaticText properties 2378
Transparent 2380
TCustomStringsValuesList class 5714
about TCustomStringsValuesList class 5714
Create 5719
Destroy 5719
GetNameFieldClass 5720
GetValueFieldClass 5720
ImplEndIterator 5720
ImplGetListName 5721
ImplGetListNameOfValue 5721
ImplGetListValue 5721
ImplNextIteration 5722
ImplNotifyDeactivate 5722
ImplStartIterator 5722
OnPrepareStrings 5724
PrepareStrings 5723
Strings 5723
StringsPrepared 5723
TCustomStringsValuesList events 5724
TCustomStringsValuesList Members 5715
TCustomStringsValuesList methods 5719
TCustomStringsValuesList properties 5723
TCustomTcpServer class 1332
about TCustomTcpServer class 1332
Accept 1336
Accept method 1335
BlockMode 1338
Close 1336
Create 1334
Destroy 1335
DoAccept 1336
Listen 1337
Listening 1338
OnAccept 1339
OnGetThread 1340
OnListening 1341
Open 1337
ServerSocketThread 1338

TCustomTcpServer events 1339
TCustomTcpServer Members 1333
TCustomTcpServer methods 1335
TCustomTcpServer properties 1338
WaitForConnection 1337
TCustomVariantType class 5011
_AddRef 5017
_Release 5017
about TCustomVariantType class 5011
BinaryOp 5018
Cast 5018
CastTo 5019
CastToOle 5019
Clear 5019
Compare 5020
CompareOp 5021
Copy 5022
Create 5015
Create constructor 5014
Destroy 5015
DispInvoke 5022
IsClear 5023
LeftPromotion 5023
OlePromotion 5024
QueryInterface 5025
RaiseCastError 5025
RaiseDispError 5026
RaiseInvalidOp 5026
RightPromotion 5027
SimplisticClear 5028
SimplisticCopy 5028
TCustomVariantType Members 5012
TCustomVariantType methods 5016
TCustomVariantType properties 5036
UnaryOp 5029
VarDataCast 5029
VarDataCastTo 5030
VarDataCastTo method 5030
VarDataCastToOleStr 5030
VarDataClear 5031
VarDataCopy 5031
VarDataCopyNoInd 5032
VarDataFromOleStr 5032
VarDataFromStr 5033
VarDataInit 5033
VarDataIsArray 5033
VarDataIsByRef 5034
VarDataIsEmptyParam 5034
VarDataIsFloat 5034
VarDataIsNumeric 5035
VarDataIsOrdinal 5035
VarDataIsStr 5036
VarDataToStr 5036
VarType 5037
TCustomWebAppComponents class 6175
about TCustomWebAppComponents class 6175
AdapterDispatcher 6185
ApplicationAdapter 6186
CreateContext 6178
DispatchActions 6186
EndUserAdapter 6187
GetAdapterDispatcher 6179
GetLocateFileService 6179
GetPageDispatcher 6179
GetWebApplicationInfo 6180
GetWebEndUser 6180
GetWebUserList 6180
ImplFinishContext 6181
ImplHandleException 6181
ImplHandleRequest 6182
ImplInitContext 6182
InitModule 6183
InitModules 6184
ListRequestHandlers 6184
LocateFileService 6187
Notification 6185
OnException 6189
PageDispatcher 6188

Sessions 6188
TCustomWebAppComponents events 6189
TCustomWebAppComponents Members 6176
TCustomWebAppComponents methods 6178
TCustomWebAppComponents properties 6185
UserListService 6189
TCustomWebAppDataModule class 6505
about TCustomWebAppDataModule class 6505
AppServices 6510
Create 6507
CreateNew 6507
Destroy 6508
GetWebAppServices 6508
Notification 6509
QueryInterface 6509
TCustomWebAppDataModule Members 6506
TCustomWebAppDataModule methods 6508
TCustomWebAppDataModule properties 6509
TCustomWebAppPageModule class 6510
about TCustomWebAppPageModule class 6510
AppServices 6515
Create 6512
CreateNew 6513
Destroy 6513
GetWebAppServices 6514
Notification 6514
QueryInterface 6514
SetAppServices 6515
TCustomWebAppPageModule Members 6511
TCustomWebAppPageModule methods 6513
TCustomWebAppPageModule properties 6515
TCustomWebDataModule class 6516
about TCustomWebDataModule class 6516
Create 6517
CreateNew 6518
Destroy 6518
GetSiteModuleHelper 6519
GetWebSnapObjClass 6519
Notification 6519
OnActivate 6522
OnDeactivate 6522
Request 6520
Response 6520
Session 6521
SiteModuleHelper 6521
TCustomWebDataModule events 6521
TCustomWebDataModule Members 6517
TCustomWebDataModule methods 6519
TCustomWebDataModule properties 6520
TCustomWebPageModule class 6522
about TCustomWebPageModule class 6522
Create 6525
CreateNew 6525
DefaultAction 6527
Destroy 6526
GetSiteModuleHelper 6526
GetSitePageModuleHelper 6527
Notification 6527
OnAfterDispatchPage 6529
OnAfterRedirectToPage 6529
OnBeforeDispatchPage 6530
OnBeforeRedirectToPage 6530
PageProducer 6528
SitePageModuleHelper 6528
TCustomWebPageModule events 6529
TCustomWebPageModule Members 6523
TCustomWebPageModule methods 6526
TCustomWebPageModule properties 6527
TCustomWebUserList class 6744
about TCustomWebUserList class 6744
CheckAccessRights 6746
Create 6745
Destroy 6745
LoadFromFile 6746
OnAfterCheckAccessRights 6748
OnAfterValidateUser 6749
OnBeforeCheckAccessRights 6749
OnBeforeValidateUser 6749

OnUserIDNotFound 6750
OnValidateUserError 6750
SaveToFile 6747
TCustomWebUserList events 6748
TCustomWebUserList Members 6744
TCustomWebUserList methods 6746
TCustomWebUserList properties 6747
UserItems 6747
Users 6748
ValidateUser 6747
TCustomWinSocket class 54
about TCustomWinSocket class 54
Accept 58
Addr 72
AsyncInitSocket 58
ASyncStyles 72
Close 59
Connect 59
Connected 73
Create 56
Data 73
DefaultHandler 60
Destroy 56
Disconnect 60
DoListen 60
DoOpen 61
Error 61
Event 62
Handle 73
InitSocket 62
Listen 63
LocalAddress 74
LocalHost 74
LocalPort 75
Lock 64
LookupName 64
LookupService 65
LookupState 75
OnErrorEvent 79
OnSocketEvent 79
Open 65
Read 66
ReceiveBuf 66
ReceiveLength 67
ReceiveText 67
RemoteAddr 76
RemoteAddress 76
RemoteHost 77
RemotePort 77
SendBuf 68
SendStream 68
SendStreamThenDrop 69
SendText 69
SetAsyncStyles 70
SocketHandle 78
TCustomWinSocket events 78
TCustomWinSocket Members 54
TCustomWinSocket methods 57
TCustomWinSocket properties 71
Unlock 70
Write 70
TCustomXSLPageProducer class 7822
about TCustomXSLPageProducer class 7822
Content 7829
ContentFromStream 7829
ContentFromString 7830
ContentFromWideString 7830
Dispatcher 7835
DispatcherComponent 7836
GetAppDispatcher 7831
GetProducerTemplateFileName 7831
GetProducerTemplateStream 7831
GetTemplateFileType 7831
GetTemplateStream 7832
GetXMLData 7832
GetXMLDocument 7832
GetXSLData 7832
HasHTMLBrowserView 7832

HasScriptView 7833
HasXMLBrowserView 7833
HasXSLBrowserView 7833
Notification 7833
ProduceContent 7834
ProduceContentFromStream 7834
ProduceContentFromString 7834
SetAppDispatcher 7835
TCustomXSLPageProducer Members 7823
TCustomXSLPageProducer methods 7828
TCustomXSLPageProducer properties 7835
XMLData 7836
TCustomZStream class 7849
about TCustomZStream class 7849
Create 7850
DoProgress 7850
OnProgress 7850
TCustomZStream events 7850
TCustomZStream Members 7849
TCustomZStream methods 7850
TDataBlock class 153
about TDataBlock class 153
BytesReserved 161
Clear 156
Create 155
Destroy 155
GetBytesReserved 157
GetMemory 157
GetSignature 157
GetSize 158
GetStream 158
IgnoreStream 158
InitData 158
Memory 161
Read 159
SetSignature 159
SetSize 160
Signature 162
Size 162
Stream 162
TDataBlock Members 153
TDataBlock methods 156
TDataBlock properties 160
Write 160
TDataBlockInterpreter class 163
about TDataBlockInterpreter class 163
AddDispatch 165
CallCreateObject 166
CallFreeObject 166
CallGetIDsOfNames 166
CallGetServerList 166
CallInvoke 167
CanCreateObject 167
Create 164
CreateObject 167
Destroy 164
DoCreateObject 167
DoCustomAction 167
DoFreeObject 168
DoGetAppServerList 168
DoGetIDsOfNames 168
DoGetServerList 168
DoInvoke 169
InternalCreateObject 169
InterpretData 169
LockObject 169
ReleaseObject 170
RemoveDispatch 170
StoreObject 170
TDataBlockInterpreter Members 163
TDataBlockInterpreter methods 165
UnlockObject 170
TDataDispatch class 171
about TDataDispatch class 171
Create 172
Destroy 173
DispatchIndex 174
GetIDsOfNames 173

GetTypeInfo 174
GetTypeInfoCount 174
Invoke 174
TDataDispatch Members 171
TDataDispatch methods 173
TDataDispatch properties 174
TDateTime class 3335
- 3347
-- 3347
- operator 3346
-- operator 3347
!= 3348
+ 3348
+ operator 3348
++ 3349
++ operator 3349
+= 3349
+= operator 3349
< 3350
<= 3350
= 3351
-= 3352
= operator 3350
-= operator 3351
== 3352
> 3352
>= 3353
about TDateTime class 3335
CurrentDate 3340
CurrentDateTime 3341
CurrentTime 3341
DateString 3342
DateTimeString 3342
DayOfWeek 3342
DecodeDate 3343
DecodeTime 3343
double 3353
FileDate 3343
FileDateToDateTime 3344
FormatString 3344
int 3353
System::String 3353
TDateTime 3338
TDateTime constructor 3338
TDateTime enumerations 3339
TDateTime Members 3336
TDateTime methods 3340
TDateTime operators 3346
TimeString 3345
TDateTimeBase class 3354
about TDateTimeBase class 3354
TDateTimeBase data members 3354
TDateTimeBase Members 3354
Val 3354
TDefaultFieldsAdapter class 5724
about TDefaultFieldsAdapter class 5724
Actions 5732
AddDefaultActions 5729
AddDefaultFields 5730
Data 5733
DefaultWebComponentsChange 5730
GetActionsClass 5730
GetDataClass 5731
ImplGetVisibleActions 5731
ImplGetVisibleFields 5731
ImplSetActions 5732
ImplSetData 5732
TDefaultFieldsAdapter Members 5725
TDefaultFieldsAdapter methods 5729
TDefaultFieldsAdapter properties 5732
TDefaultFieldsPagedAdapter class 5733
about TDefaultFieldsPagedAdapter class 5733
AddDefaultActions 5739
CurrentPage 5746
EncodeActionParams 5740
EndOfPage 5740
ExtractRequestParams 5740
GetEOF 5741

GetFirstRecord 5741
GetNextRecord 5741
GetPageCount 5742
GetRecordCount 5742
GetRecordIndex 5742
ImplCanAddActionClass 5743
ImplEndIterator 5743
ImplGetActionsList 5743
ImplNextIteration 5744
ImplNotifyDeactivate 5744
ImplStartIterator 5745
OnGetEOF 5748
OnGetFirstRecord 5749
OnGetNextRecord 5749
OnGetRecordCount 5749
OnGetRecordIndex 5749
PageCount 5747
PageSize 5747
RecordCount 5747
RecordIndex 5748
SetPageSize 5745
SkipToPage 5746
TDefaultFieldsPagedAdapter events 5748
TDefaultFieldsPagedAdapter Members 5735
TDefaultFieldsPagedAdapter methods 5739
TDefaultFieldsPagedAdapter properties 5746
TDefinition class 6882
about TDefinition class 6882
AfterConstruction 6887
GetBindings 6888
GetImports 6888
GetMessages 6888
GetName 6889
GetPortTypes 6889
GetServices 6889
GetTargetNamespace 6889
GetTypes 6889
SetName 6890
SetTargetNamespace 6890
TDefinition Members 6882
TDefinition methods 6887
TDependencies class 3009
about TDependencies class 3009
Create 3010
GetOwner 3010
Items 3011
TDependencies Members 3009
TDependencies methods 3010
TDependencies properties 3011
TDependency class 3012
about TDependency class 3012
GetDisplayName 3013
IsGroup 3013
Name 3014
TDependency Members 3012
TDependency methods 3013
TDependency properties 3013
TDesigntimeWarnings class 6470
about TDesigntimeWarnings class 6470
AddObject 6471
AddString 6471
Create 6470
Destroy 6470
Strings 6472
TDesigntimeWarnings Members 6470
TDesigntimeWarnings methods 6471
TDesigntimeWarnings properties 6472
TDisplayOffset class 278
about TDisplayOffset class 278
Control 279
Create 278
TDisplayOffset Members 278
TDisplayOffset properties 279
X 279
Y 279
TDocumentation class 6890
about TDocumentation class 6890
TDocumentationCollection class 6891

about TDocumentationCollection class 6891
Add 6896
GetDocumentation 6897
Insert 6897
TDocumentationCollection Members 6891
TDocumentationCollection methods 6896
TDocumented class 6897
about TDocumented class 6897
AfterConstruction 6902
GetDocumentationCollection 6903
TDocumented Members 6897
TDocumented methods 6902
TDOMStrings class 7133
about TDOMStrings class 7133
Add 7134
Clear 7135
Count 7135
Create 7134
Destroy 7134
IndexOf 7135
IndexOfIgnoreCase 7135
Insert 7135
Strings 7136
TDOMStrings Members 7133
TDOMStrings methods 7134
TDOMStrings properties 7136
TDOMVendor class 7466
about TDOMVendor class 7466
Description 7467
DOMImplementation 7467
TDOMVendor Members 7466
TDOMVendor methods 7466
TDOMVendorList class 7468
about TDOMVendorList class 7468
Add 7469
Count 7469
Find 7470
GetVendors 7470
Remove 7471
TDOMVendorList Members 7468
TDOMVendorList methods 7469
TDOMVendorList properties 7471
Vendors 7471
tebExplorerBarDontCare enumeration member 4641
tebExplorerBarRoot enumeration member 4641
tebHeaderBackgroundHot enumeration member 4641
tebHeaderBackgroundNormal enumeration member 4641
tebHeaderBackgroundPressed enumeration member 4641
tebHeaderCloseHot enumeration member 4641
tebHeaderCloseNormal enumeration member 4641
tebHeaderClosePressed enumeration member 4641
tebHeaderPinHot enumeration member 4641
tebHeaderPinNormal enumeration member 4641
tebHeaderPinPressed enumeration member 4641
tebHeaderPinSelectedHot enumeration member 4641
tebHeaderPinSelectedNormal enumeration member 4641
tebHeaderPinSelectedPressed enumeration member 4641
tebIEBarMenuHot enumeration member 4641
tebIEBarMenuNormal enumeration member 4641
tebIEBarMenuPressed enumeration member 4641
tebNormalGroupBackground enumeration member 4641
tebNormalGroupCollapseHot enumeration member 4641
tebNormalGroupCollapseNormal enumeration member 4641
tebNormalGroupCollapsePressed enumeration member 4641
tebNormalGroupExpandHot enumeration member 4641
tebNormalGroupExpandNormal enumeration member 4641
tebNormalGroupExpandPressed enumeration member 4641
tebNormalGroupHead enumeration member 4641
tebSpecialGroupBackground enumeration member 4641
tebSpecialGroupCollapseHot enumeration member 4641
tebSpecialGroupCollapsePressed enumeration member 4641
tebSpecialGroupCollapseSpecial enumeration member 4641
tebSpecialGroupExpandHot enumeration member 4641
tebSpecialGroupExpandPressed enumeration member 4641
tebSpecialGroupExpandSpecial enumeration member 4641
tebSpecialGroupHead enumeration member 4641
teButton enumeration member 4639
teClock enumeration member 4639

teComboBox enumeration member 4639
TEdit class 2380
about TEdit class 2380
Align 2385
Alignment 2388
Anchors 2389
AutoSelect 2389
AutoSize 2390
BevelEdges 2391
BevelInner 2391
BevelKind 2392
BevelOuter 2392
BevelWidth 2392
BiDiMode 2393
BorderStyle 2393
CharCase 2394
Color 2394
Constraints 2395
Ctl3D 2395
DoubleBuffered 2396
DragCursor 2398
DragKind 2398
DragMode 2398
Enabled 2400
Font 2400
HideSelection 2402
ImeMode 2402
ImeName 2403
MaxLength 2403
NumbersOnly 2404
OEMConvert 2404
OnChange 2419
OnClick 2419
OnContextPopup 2421
OnDblClick 2422
OnDragDrop 2422
OnDragOver 2425
OnEndDock 2428
OnEndDrag 2429
OnEnter 2430
OnExit 2432
OnKeyDown 2435
OnKeyPress 2438
OnKeyUp 2440
OnMouseActivate 2441
OnMouseDown 2442
OnMouseEnter 2445
OnMouseLeave 2445
OnMouseMove 2445
OnMouseUp 2447
OnStartDock 2450
OnStartDrag 2450
ParentBiDiMode 2404
ParentColor 2405
ParentCtl3D 2405
ParentDoubleBuffered 2406
ParentFont 2406
ParentShowHint 2406
PasswordChar 2407
PopupMenu 2407
ReadOnly 2412
ShowHint 2413
TabOrder 2414
TabStop 2415
TEdit events 2418
TEdit Members 2381
TEdit properties 2384
Text 2415
TextHint 2416
Visible 2417
TEditAction class 1722
about TEditAction class 1722
Control 1726
Destroy 1723
GetControl 1724
HandlesTarget 1724
Notification 1725
TEditAction Members 1723

TEditAction methods 1723
TEditAction properties 1725
UpdateTarget 1725
TEditCopy class 1726
about TEditCopy class 1726
ExecuteTarget 1727
TEditCopy Members 1727
TEditCopy methods 1727
TEditCut class 1728
about TEditCut class 1728
ExecuteTarget 1729
TEditCut Members 1729
TEditCut methods 1729
TEditDelete class 1730
about TEditDelete class 1730
ExecuteTarget 1731
TEditDelete Members 1730
TEditDelete methods 1731
UpdateTarget 1731
TEditPaste class 1732
about TEditPaste class 1732
ExecuteTarget 1733
TEditPaste Members 1733
TEditPaste methods 1733
UpdateTarget 1734
TEditSelectAll class 1734
about TEditSelectAll class 1734
ExecuteTarget 1736
TEditSelectAll Members 1735
TEditSelectAll methods 1735
UpdateTarget 1736
TEditUndo class 1736
about TEditUndo class 1736
ExecuteTarget 1738
TEditUndo Members 1737
TEditUndo methods 1738
UpdateTarget 1738
teEdit enumeration member 4639
teEditCaret enumeration member 4639
teEditDontCare enumeration member 4639
teEditRoot enumeration member 4639
teEditTextAssist enumeration member 4639
teEditTextDisabled enumeration member 4639
teEditTextFocused enumeration member 4639
teEditTextHot enumeration member 4639
teEditTextNormal enumeration member 4639
teEditTextReadOnly enumeration member 4639
teEditTextSelected enumeration member 4639
teExplorerBar enumeration member 4639
teHeader enumeration member 4639
teListView enumeration member 4639
teMenu enumeration member 4639
TEncoding class 4035
about TEncoding class 4035
ASCII 4048
BigEndianUnicode 4049
Convert 4041
Convert method 4040
Default 4049
FIsSingleByte 4039
FMaxCharSize 4040
FreeEncodings 4041
GetBufferEncoding 4042
GetByteCount 4042
GetByteCount method 4042
GetBytes 4043
GetBytes method 4043
GetCharCount 4044
GetCharCount method 4044
GetChars 4045
GetChars method 4045
GetEncoding 4046
GetMaxByteCount 4046
GetMaxCharCount 4046
GetPreamble 4047
GetString 4047
GetString method 4047
IsSingleByte 4049

IsStandardEncoding 4048
TEncoding fields 4039
TEncoding Members 4038
TEncoding methods 4040
TEncoding properties 4048
Unicode 4050
UTF7 4050
UTF8 4050
TEndUserAdapter class 5750
about TEndUserAdapter class 5750
Actions 5756
Data 5756
LoginPage 5757
OnAfterExecuteAction 5757
OnAfterGetActionResponse 5758
OnBeforeExecuteAction 5758
OnBeforeGetActionResponse 5758
OnGetActionParams 5759
OnGetDisplayName 5759
OnHasRights 5760
OnIsLoggedIn 5760
OnLogin 5761
OnLogout 5761
OnProduceLoginPage 5762
TEndUserAdapter events 5757
TEndUserAdapter Members 5751
TEndUserAdapter properties 5756
TEndUserAdapterLoginFormAction class 5762
about TEndUserAdapterLoginFormAction class 5762
DisplayLabel 5768
GetDefaultActionName 5766
ImplExecuteActionRequest 5767
ImplWebEnabled 5767
OnAfterExecute 5768
OnAfterGetResponse 5769
OnBeforeExecute 5769
OnBeforeGetResponse 5769
OnGetParams 5770
TEndUserAdapterLoginFormAction events 5768
TEndUserAdapterLoginFormAction Members 5763
TEndUserAdapterLoginFormAction methods 5766
TEndUserAdapterLoginFormAction properties 5767
TEndUserAdapterLogoutAction class 5770
about TEndUserAdapterLogoutAction class 5770
DisplayLabel 5776
GetDefaultActionName 5775
ImplExecuteActionRequest 5775
ImplWebEnabled 5775
OnAfterExecute 5777
OnAfterGetResponse 5777
OnBeforeExecute 5777
OnBeforeGetResponse 5778
OnGetParams 5778
TEndUserAdapterLogoutAction events 5776
TEndUserAdapterLogoutAction Members 5771
TEndUserAdapterLogoutAction methods 5774
TEndUserAdapterLogoutAction properties 5776
TEndUserSessionAdapter class 5779
about TEndUserSessionAdapter class 5779
Actions 5785
Data 5786
LoginPage 5786
OnAfterExecuteAction 5787
OnAfterGetActionResponse 5787
OnBeforeExecuteAction 5787
OnBeforeGetActionResponse 5788
OnGetActionParams 5788
OnHasRights 5788
OnProduceLoginPage 5789
TEndUserSessionAdapter events 5786
TEndUserSessionAdapter Members 5780
TEndUserSessionAdapter properties 5785
tePage enumeration member 4639
teProgress enumeration member 4639
teRebar enumeration member 4639
teScrollBar enumeration member 4639
teSpin enumeration member 4639
teStartPanel enumeration member 4639

teStatus enumeration member 4639
teTab enumeration member 4639
teTaskBand enumeration member 4639
teTaskBar enumeration member 4639
teToolBar enumeration member 4639
teToolTip enumeration member 4639
teTrackBar enumeration member 4639
teTrayNotify enumeration member 4639
teTreeview enumeration member 4639
TEvent class 3074
about TEvent class 3074
Create 3076
Create constructor 3076
ResetEvent 3077
SetEvent 3078
TEvent Members 3075
TEvent methods 3077
TEventLogger class 3014
about TEventLogger class 3014
Create 3015
Destroy 3016
LogMessage 3016
TEventLogger Members 3015
TEventLogger methods 3016
teWindow enumeration member 4639
TFault class 6903
about TFault class 6903
TFaults class 6904
about TFaults class 6904
Add 6909
GetFault 6910
Insert 6910
TFaults Members 6904
TFaults methods 6909
TFieldValuesAdapter class 5789
about TFieldValuesAdapter class 5789
Create 5795
Field 5798
GetValueFieldClass 5795
ImplEndIterator 5796
ImplGetField 5796
ImplGetValueIndex 5796
ImplNextIteration 5797
ImplStartIterator 5797
TFieldValuesAdapter Members 5791
TFieldValuesAdapter methods 5795
TFieldValuesAdapter properties 5798
TFieldValuesValueField class 5799
about TFieldValuesValueField class 5799
Field 5803
GetDefaultFieldName 5802
ImplAsFormatted 5802
ImplGetValue 5803
TFieldValuesValueField Members 5799
TFieldValuesValueField methods 5801
TFieldValuesValueField properties 5803
TFileAction class 1739
about TFileAction class 1739
FileName 1741
GetDialog 1740
TFileAction Members 1739
TFileAction methods 1740
TFileAction properties 1740
TFileExit class 1741
about TFileExit class 1741
Caption 1743
Enabled 1744
ExecuteTarget 1742
HandlesTarget 1743
HelpContext 1744
HelpKeyword 1745
HelpType 1745
Hint 1746
ImageIndex 1746
OnHint 1747
OnUpdate 1748
SecondaryShortCuts 1746
ShortCut 1747

TFileExit events 1747
TFileExit Members 1742
TFileExit methods 1742
TFileExit properties 1743
Visible 1747
TFileOpen class 1749
about TFileOpen class 1749
BeforeExecute 1758
Caption 1753
Create 1751
Dialog 1753
Enabled 1753
ExecuteTarget 1751
GetDialogClass 1752
HelpContext 1754
HelpKeyword 1754
HelpType 1755
Hint 1755
ImageIndex 1755
OnAccept 1758
OnCancel 1758
OnHint 1758
OnUpdate 1759
SecondaryShortCuts 1756
ShortCut 1756
TFileOpen events 1757
TFileOpen Members 1750
TFileOpen methods 1751
TFileOpen properties 1752
UseDefaultApp 1757
Visible 1757
TFileOpenWith class 1760
about TFileOpenWith class 1760
AfterOpen 1765
BeforeExecute 1765
ExecuteTarget 1762
FileName 1763
HelpContext 1763
HelpKeyword 1764
HelpType 1764
OnUpdate 1766
TFileOpenWith events 1765
TFileOpenWith Members 1761
TFileOpenWith methods 1762
TFileOpenWith properties 1763
TFilePageSetup class 1767
about TFilePageSetup class 1767
BeforeExecute 1774
Caption 1769
Dialog 1770
Enabled 1770
GetDialogClass 1769
HelpContext 1770
HelpKeyword 1771
HelpType 1771
Hint 1772
ImageIndex 1772
OnAccept 1774
OnCancel 1774
OnHint 1775
OnUpdate 1775
SecondaryShortCuts 1772
ShortCut 1773
TFilePageSetup events 1773
TFilePageSetup Members 1767
TFilePageSetup methods 1768
TFilePageSetup properties 1769
Visible 1773
TFilePrintSetup class 1776
about TFilePrintSetup class 1776
BeforeExecute 1783
Caption 1778
Dialog 1779
Enabled 1779
GetDialogClass 1778
HelpContext 1779
HelpKeyword 1780
HelpType 1780

Hint 1781
ImageIndex 1781
OnAccept 1783
OnCancel 1783
OnHint 1784
OnUpdate 1784
SecondaryShortCuts 1781
ShortCut 1782
TFilePrintSetup events 1783
TFilePrintSetup Members 1777
TFilePrintSetup methods 1778
TFilePrintSetup properties 1778
Visible 1782
TFileSaveAs class 1785
about TFileSaveAs class 1785
BeforeExecute 1791
Caption 1788
Create 1787
Dialog 1788
Enabled 1789
GetDialogClass 1787
HelpContext 1789
Hint 1789
ImageIndex 1790
OnAccept 1792
OnCancel 1792
OnHint 1792
OnUpdate 1793
SecondaryShortCuts 1790
ShortCut 1791
TFileSaveAs events 1791
TFileSaveAs Members 1786
TFileSaveAs methods 1787
TFileSaveAs properties 1788
Visible 1791
TFLDDesc class 1471
about TFLDDesc class 1471
bCalcField 1478
efldrRights 1479
efldvVchk 1479
GetbCalcField 1473
GetefldrRights 1473
GetefldvVchk 1473
GetiFldNum 1473
GetiFldType 1474
GetiLen 1474
GetiNullOffset 1474
GetiOffset 1474
GetiSubType 1474
GetiUnits1 1475
GetiUnits2 1475
GetszName 1475
iFldNum 1479
iFldType 1479
iLen 1479
iNullOffset 1480
iOffset 1480
iSubType 1480
iUnits1 1480
iUnits2 1481
SetbCalcField 1475
SetefldrRights 1476
SetefldvVchk 1476
SetiFldNum 1476
SetiFldType 1476
SetiLen 1476
SetiNullOffset 1477
SetiOffset 1477
SetiSubType 1477
SetiUnits1 1477
SetiUnits2 1478
SetszName 1478
szName 1481
TFLDDesc Members 1471
TFLDDesc methods 1472
TFLDDesc properties 1478
TFontEdit class 1794
about TFontEdit class 1794

BeforeExecute 1801
Caption 1796
Dialog 1796
Enabled 1797
GetDialogClass 1795
HelpContext 1797
HelpKeyword 1798
HelpType 1798
Hint 1799
ImageIndex 1799
OnAccept 1801
OnCancel 1801
OnHint 1801
OnUpdate 1802
SecondaryShortCuts 1799
ShortCut 1800
TFontEdit events 1800
TFontEdit Members 1794
TFontEdit methods 1795
TFontEdit properties 1796
Visible 1800
TFTCompT class 1247
about TFTCompT class 1247
CStringEvent 1248
fireEvent 1248
fireEventEx 1248
StringEvent 1248
TFTCompT events 1248
TFTCompT Members 1247
TFTCompT methods 1247
VStringEvent 1249
TFTCompTI class 1249
about TFTCompTI class 1249
Add 1251
Append 1251
Doubler 1251
GetAsObject 1252
TFTCompTI Members 1249
TFTCompTI methods 1251
UpCase 1252
TGeneratedFunctions class 6472
about TGeneratedFunctions class 6472
Add 6473
Create 6473
Destroy 6473
Exists 6474
ItemCount 6474
Names 6474
Script 6474
TGeneratedFunctions Members 6472
TGeneratedFunctions methods 6473
TGeneratedFunctions properties 6474
TGroupBox class 2451
about TGroupBox class 2451
Align 2454
Anchors 2457
BiDiMode 2458
Caption 2458
Color 2461
Constraints 2462
Ctl3D 2462
DockSite 2463
DoubleBuffered 2464
DragCursor 2465
DragKind 2466
DragMode 2466
Enabled 2468
Font 2468
OnAlignInsertBefore 2482
OnAlignPosition 2482
OnClick 2483
OnContextPopup 2485
OnDblClick 2486
OnDockDrop 2486
OnDockOver 2488
OnDragDrop 2490
OnDragOver 2493
OnEndDock 2496

OnEndDrag 2497
OnEnter 2498
OnExit 2500
OnGetSiteInfo 2503
OnMouseActivate 2505
OnMouseDown 2506
OnMouseEnter 2508
OnMouseLeave 2508
OnMouseMove 2509
OnMouseUp 2511
OnStartDock 2513
OnStartDrag 2514
OnUnDock 2514
Padding 2469
ParentBackground 2469
ParentBiDiMode 2470
ParentColor 2470
ParentCtl3D 2471
ParentDoubleBuffered 2471
ParentFont 2471
ParentShowHint 2472
PopupMenu 2472
ShowHint 2477
TabOrder 2479
TabStop 2479
TGroupBox events 2481
TGroupBox Members 2451
TGroupBox properties 2453
Visible 2480
THandleObject class 3078
about THandleObject class 3078
Create 3080
Destroy 3080
FHandle 3081
FLastError 3081
FUseCOMWait 3081
Handle 3082
LastError 3083
THandleObject fields 3081
THandleObject Members 3079
THandleObject methods 3081
THandleObject properties 3082
WaitFor 3082
WaitForMultiple 3082
THelpAction class 1803
about THelpAction class 1803
Create 1804
HandlesTarget 1804
THelpAction Members 1804
THelpAction methods 1804
UpdateTarget 1805
THelpContents class 1805
about THelpContents class 1805
ExecuteTarget 1806
THelpContents Members 1806
THelpContents methods 1806
THelpContextAction class 1806
about THelpContextAction class 1806
ExecuteTarget 1807
THelpContextAction Members 1807
THelpContextAction methods 1807
UpdateTarget 1808
THelpOnHelp class 1808
about THelpOnHelp class 1808
ExecuteTarget 1809
THelpOnHelp Members 1809
THelpOnHelp methods 1809
THelpTopicSearch class 1810
about THelpTopicSearch class 1810
ExecuteTarget 1811
THelpTopicSearch Members 1810
THelpTopicSearch methods 1811
ThemedActnCtrls 4593
ThemedActnCtrls namespace 4593
Classes 4594
ThemedActnCtrls.TThemedAddRemoveItem 4594
ThemedActnCtrls.TThemedAddRemoveItem.CalcBounds 4595
ThemedActnCtrls.TThemedAddRemoveItem.DrawBackground 4595

ThemedActnCtrls.TThemedAddRemoveItem.DrawGlyph 4596
ThemedActnCtrls.TThemedAddRemoveItem.DrawSeparator 4596
ThemedActnCtrls.TThemedAddRemoveItem.DrawText 4596
ThemedActnCtrls.TThemedAddRemoveItem.DrawUnusedEdges 4597
ThemedActnCtrls.TThemedAddRemoveItem.GetImageSize 4597
ThemedActnCtrls.TThemedButtonControl 4597
ThemedActnCtrls.TThemedButtonControl.DrawBackground 4598
ThemedActnCtrls.TThemedButtonControl.DrawFrame 4598
ThemedActnCtrls.TThemedButtonControl.DrawGlyph 4598
ThemedActnCtrls.TThemedButtonControl.DrawLargeGlyph 4599
ThemedActnCtrls.TThemedButtonControl.DrawText 4599
ThemedActnCtrls.TThemedCustomizePopup 4599
ThemedActnCtrls.TThemedCustomizePopup.Create 4600
ThemedActnCtrls.TThemedCustomizePopup.GetAddRemoveItemClass 4600
ThemedActnCtrls.TThemedCustomizePopup.GetDefaultColorMapClass 4601
ThemedActnCtrls.TThemedCustomizePopup.NCPaint 4601
ThemedActnCtrls.TThemedDropDownButton 4601
ThemedActnCtrls.TThemedDropDownButton.DrawBackground 4602
ThemedActnCtrls.TThemedDropDownButton.DrawFrame 4602
ThemedActnCtrls.TThemedDropDownButton.DrawGlyph 4603
ThemedActnCtrls.TThemedDropDownButton.DrawLargeGlyph 4603
ThemedActnCtrls.TThemedDropDownButton.GetDropDownButtonWidth 4603
ThemedActnCtrls.TThemedDropDownButton.GetPopupClass 4603
ThemedActnCtrls.TThemedExpandBtn 4604
ThemedActnCtrls.TThemedExpandBtn.DrawBackground 4604
ThemedActnCtrls.TThemedExpandBtn.DrawFrame 4605
ThemedActnCtrls.TThemedExpandBtn.GetImageSize 4605
ThemedActnCtrls.TThemedMenuButton 4605
ThemedActnCtrls.TThemedMenuButton.CalcBounds 4606
ThemedActnCtrls.TThemedMenuButton.DrawBackground 4606
ThemedActnCtrls.TThemedMenuButton.DrawText 4607
ThemedActnCtrls.TThemedMenuItem 4607
ThemedActnCtrls.TThemedMenuItem.CalcBounds 4608
ThemedActnCtrls.TThemedMenuItem.DrawBackground 4608
ThemedActnCtrls.TThemedMenuItem.DrawGlyph 4609
ThemedActnCtrls.TThemedMenuItem.DrawLargeGlyph 4609
ThemedActnCtrls.TThemedMenuItem.DrawSeparator 4609
ThemedActnCtrls.TThemedMenuItem.DrawShadowedText 4610
ThemedActnCtrls.TThemedMenuItem.DrawSubMenuGlyph 4610
ThemedActnCtrls.TThemedMenuItem.DrawText 4610
ThemedActnCtrls.TThemedMenuItem.DrawUnusedEdges 4610
ThemedActnCtrls.TThemedMenuItem.GetImageSize 4611
ThemedActnCtrls.TThemedPopupMenu 4611
ThemedActnCtrls.TThemedPopupMenu.Create 4611
ThemedActnCtrls.TThemedPopupMenu.GetExpandBtnClass 4612
ThemedActnCtrls.TThemedPopupMenu.NCPaint 4612
ThemedActnCtrls.TThemedPopupMenu.PositionPopup 4612
ThemedActnCtrls.TThemedPopupMenu.VisibleChanging 4613
ThemedActnCtrls.TThemedToolScrollBtn 4613
ThemedActnCtrls.TThemedToolScrollBtn.DrawBackground 4614
Themes 4615
Themes namespace 4615
- Classes 4617
- Functions 4634
- Structs, Records, Enums 4636
- Types 4663
- Variables 4665

Themes.PThemedElementDetails 4663
Themes.PThemedElementDetails type 4663
Themes.ThemeControl 4634
Themes.ThemeControl function 4634
Themes.ThemeServices 4634
Themes.ThemeServices function 4634
Themes.ThemeServicesClass 4665
Themes.ThemeServicesClass variable 4665
Themes.TThemeData 4663
Themes.TThemeData type 4663

Themes.TThemedButton 4636
Themes.TThemedButton enumeration 4636
Themes.TThemedClock 4638
Themes.TThemedClock enumeration 4638
Themes.TThemedComboBox 4638
Themes.TThemedComboBox enumeration 4638
Themes.TThemedEdit 4639
Themes.TThemedEdit enumeration 4639
Themes.TThemedElement 4639
Themes.TThemedElement enumeration 4639
Themes.TThemedElementDetails 4640
Themes.TThemedElementDetails record 4640
Themes.TThemedExplorerBar 4641
Themes.TThemedExplorerBar enumeration 4641
Themes.TThemedHeader 4642
Themes.TThemedHeader enumeration 4642
Themes.TThemedListview 4643
Themes.TThemedListview enumeration 4643
Themes.TThemedMenu 4643
Themes.TThemedMenu enumeration 4643
Themes.TThemedPage 4644
Themes.TThemedPage enumeration 4644
Themes.TThemedProgress 4645
Themes.TThemedProgress enumeration 4645
Themes.TThemedRebar 4646
Themes.TThemedRebar enumeration 4646
Themes.TThemedScrollBar 4647
Themes.TThemedScrollBar enumeration 4647
Themes.TThemedSpin 4649
Themes.TThemedSpin enumeration 4649
Themes.TThemedStartPanel 4650
Themes.TThemedStartPanel enumeration 4650
Themes.TThemedStatus 4650
Themes.TThemedStatus enumeration 4650
Themes.TThemedTab 4651
Themes.TThemedTab enumeration 4651
Themes.TThemedTaskBand 4653
Themes.TThemedTaskBand enumeration 4653
Themes.TThemedTaskBar 4653
Themes.TThemedTaskBar enumeration 4653
Themes.TThemedToolBar 4654
Themes.TThemedToolBar enumeration 4654
Themes.TThemedToolTip 4655
Themes.TThemedToolTip enumeration 4655
Themes.TThemedTrackBar 4656
Themes.TThemedTrackBar enumeration 4656
Themes.TThemedTrayNotify 4657
Themes.TThemedTrayNotify enumeration 4657
Themes.TThemedTreeview 4658
Themes.TThemedTreeview enumeration 4658
Themes.TThemedWindow 4658
Themes.TThemedWindow enumeration 4658
Themes.TThemeServices 4617
Themes.TThemeServices.ApplyThemeChange 4620
Themes.TThemeServices.ColorToRGB 4620, 4621
Themes.TThemeServices.ContentRect 4621
Themes.TThemeServices.Create 4618
Themes.TThemeServices.Destroy 4619
Themes.TThemeServices.DoOnThemeChange 4621
Themes.TThemeServices.DrawEdge 4622
Themes.TThemeServices.DrawElement 4622, 4623
Themes.TThemeServices.DrawIcon 4623
Themes.TThemeServices.DrawParentBackground 4624
Themes.TThemeServices.DrawText 4625
Themes.TThemeServices.GetElementDetails 4626, 4627, 4628, 4629, 4630, 4631
Themes.TThemeServices.HasTransparentParts 4631
Themes.TThemeServices.OnThemeChange 4633
Themes.TThemeServices.PaintBorder 4631
Themes.TThemeServices.Theme 4632
Themes.TThemeServices.ThemesAvailable 4632
Themes.TThemeServices.ThemesEnabled 4632
Themes.TThemeServices.UnloadThemeData 4631
Themes.TThemeServices.UpdateThemes 4632
Themes.TThemeServicesClass 4663
Themes.TThemeServicesClass type 4663
Themes.UnthemedDesigner 4634
Themes.UnthemedDesigner function 4634
thHeaderDontCare enumeration member 4642

thHeaderItemHot enumeration member 4642
thHeaderItemLeftHot enumeration member 4642
thHeaderItemLeftNormal enumeration member 4642
thHeaderItemLeftPressed enumeration member 4642
thHeaderItemNormal enumeration member 4642
thHeaderItemPressed enumeration member 4642
thHeaderItemRightHot enumeration member 4642
thHeaderItemRightNormal enumeration member 4642
thHeaderItemRightPressed enumeration member 4642
thHeaderRoot enumeration member 4642
thHeaderSortArrowSortedDown enumeration member 4642
thHeaderSortArrowSortedUp enumeration member 4642
THintAction class 1811
about THintAction class 1811
Create 1812
Hint 1813
THintAction Members 1812
THintAction properties 1813
THTMLItem class 6628
about THTMLItem class 6628
AssignString 6630
Create 6630
HTML 6630
Ident 6631
ItemLength 6631
TagToken 6631
THTMLItem Members 6629
THTMLItem methods 6630
THTMLItem properties 6630
THTMLItems class 6631
about THTMLItems class 6631
Create 6632
Destroy 6632
GetHTMLItem 6633
Items 6633
MakeIdentifier 6633
THTMLItems Members 6632
THTMLItems methods 6633
THTMLItems properties 6633
THTTPReqResp class 1214
_AddRef 1217
_Release 1218
about THTTPReqResp class 1214
AfterConstruction 1218
Agent 1226
BeforeExecute 1219
Check 1219
CheckContentType 1220
Connect 1220
ConnectTimeout 1226
Create 1216
Destroy 1216
Execute 1221
Execute method 1221
Get 1222
GetHTTPReqResp 1222
GetMimeBoundary 1222
HandleWinInetError 1223
InvokeOptions 1226
MaxSinglePostSize 1227
NewInstance 1223
OnBeforePost 1232
OnPostingData 1232
OnReceivingData 1233
OnWinInetError 1233
Password 1227
Proxy 1227
ProxyByPass 1228
Receive 1223
ReceiveTimeout 1228
Send 1224
SendGet 1224
SendTimeout 1229
SetMimeBoundary 1225
SoapAction 1229
THTTPReqResp events 1232
THTTPReqResp Members 1215
THTTPReqResp methods 1217

THTTPReqResp properties 1225
UDDIBindingKey 1229
UDDIOperator 1230
URL 1230
UserName 1231
UseUTF8InHeader 1231
WSDLView 1231
THTTPRIO class 1186
about THTTPRIO class 1186
Converter 1190
Create 1187
Destroy 1188
HTTPWebNode 1190
Notification 1188
Port 1191
QueryInterface 1189
Service 1191
THTTPRIO Members 1187
THTTPRIO methods 1188
THTTPRIO properties 1190
URL 1192
WSDLItems 1193
WSDLLocation 1193
THTTPSoapCppInvoker class 1204
about THTTPSoapCppInvoker class 1204
THTTPSoapDispatcher class 5920
about THTTPSoapDispatcher class 5920
Create 5921
Destroy 5922
DispatchEnabled 5922
DispatchMask 5923
DispatchMethodType 5923
DispatchRequest 5923
THTTPSoapDispatcher Members 5921
THTTPSoapDispatcher methods 5922
THTTPSoapDispatcher properties 5924
WebDispatch 5924
THTTPSoapDispatchNode class 1196
about THTTPSoapDispatchNode class 1196
Dispatcher 1199
DispatchSOAP 1197
FSoapDispatcher 1197
Notification 1198
THTTPSoapDispatchNode fields 1197
THTTPSoapDispatchNode Members 1196
THTTPSoapDispatchNode methods 1197
THTTPSoapDispatchNode properties 1198
THTTPSoapPascalInvoker class 1205
about THTTPSoapPascalInvoker class 1205
DispatchSOAP 1207
THTTPSoapPascalInvoker Members 1206
THTTPSoapPascalInvoker methods 1207
TImageMargins class 2517
about TImageMargins class 2517
Assign 2518
Bottom 2520
Change 2519
Left 2520
OnChange 2521
Right 2520
TImageMargins events 2521
TImageMargins Members 2518
TImageMargins methods 2518
TImageMargins properties 2519
Top 2521
Time enumeration member 3340
TImplementedAdapterAction class 5803
about TImplementedAdapterAction class 5803
TImport class 6910
about TImport class 6910
GetLocation 6915
GetNamespace 6916
SetLocation 6916
SetNamespace 6916
TImport Members 6910
TImport methods 6915
TImports class 6916
about TImports class 6916

Add 6922
GetImport 6922
Insert 6923
TImports Members 6917
TImports methods 6922
TInterfacedObject class 3355
_AddRef 3357
_Release 3357
about TInterfacedObject class 3355
AfterConstruction 3358
BeforeDestruction 3358
FRefCount 3356
NewInstance 3358
QueryInterface 3359
RefCount 3360
TInterfacedObject data members 3356
TInterfacedObject Members 3355
TInterfacedObject methods 3356
TInterfacedObject properties 3359
TInternalSQLDataSet class 830
about TInternalSQLDataSet class 830
CommandText 833
CommandType 834
DataSource 835
GetMetadata 836
MaxBlobSize 836
ParamCheck 837
Params 837
SortFieldNames 838
TInternalSQLDataSet Members 830
TInternalSQLDataSet properties 833
TInternetExplorer class 379
about TInternetExplorer class 379
AddressBar 407
Application 407
Busy 407
ClientToWindow 386
Connect 386
ConnectTo 386
Container 408
Create 383
DefaultInterface 408
Destroy 383
Disconnect 387
Document 408
ExecWB 387, 388
ExecWB method 387
FullName 408
FullScreen 408
Get_AddressBar 388
Get_Application 388
Get_Busy 388
Get_Container 389
Get_Document 389
Get_FullName 389
Get_FullScreen 389
Get_Height 389
Get_HWnd 390
Get_Left 390
Get_LocationName 390
Get_LocationURL 390
Get_MenuBar 391
Get_Name 391
Get_Offline 391
Get_Parent 391
Get_Path 391
Get_ReadyState 392
Get_RegisterAsBrowser 392
Get_RegisterAsDropTarget 392
Get_Resizable 392
Get_Silent 393
Get_StatusBar 393
Get_StatusText 393
Get_TheaterMode 393
Get_ToolBar 393
Get_Top 394
Get_TopLevelContainer 394
Get_type_ 394

Get_Visible 394
Get_Width 395
GetProperty 395
GoBack 395
GoForward 395
GoHome 395
GoSearch 396
Height 409
HWnd 409
InitServerData 396
InvokeEvent 396
Left 409
LocationName 409
LocationURL 410
MenuBar 410
Name 410
Navigate 397, 398
Navigate method 397
Navigate2 399, 400
Navigate2 method 398
Offline 412
OnBeforeNavigate2 417
OnClientToHostWindow 417
OnCommandStateChange 417
OnDocumentComplete 417
OnDownloadBegin 417
OnDownloadComplete 418
OnFileDownload 418
OnFullScreen 418
OnMenuBar 418
OnNavigateComplete2 419
OnNavigateError 419
OnNewWindow2 419
OnPrintTemplateInstantiation 419
OnPrintTemplateTeardown 419
OnPrivacyImpactedStateChange 420
OnProgressChange 420
OnPropertyChange 420
OnQuit 420
OnSetSecureLockIcon 421
OnStatusBar 421
OnStatusTextChange 421
OnTheaterMode 421
OnTitleChange 421
OnToolBar 422
OnUpdatePageStatus 422
OnVisible 422
OnWindowClosing 422
OnWindowSetHeight 423
OnWindowSetLeft 423
OnWindowSetResizable 423
OnWindowSetTop 423
OnWindowSetWidth 423
Parent 412
Path 413
PutProperty 400
QueryStatusWB 400
Quit 400
ReadyState 413
Refresh 400
Refresh2 401
Refresh2 method 401
RegisterAsBrowser 413
RegisterAsDropTarget 413
Resizable 413
Set_AddressBar 401
Set_FullScreen 402
Set_Height 402
Set_Left 402
Set_MenuBar 402
Set_Offline 402
Set_RegisterAsBrowser 403
Set_RegisterAsDropTarget 403
Set_Resizable 403
Set_Silent 403
Set_StatusBar 404
Set_StatusText 404
Set_TheaterMode 404

Set_ToolBar 404
Set_Top 404
Set_Visible 405
Set_Width 405
ShowBrowserBar 405, 406
ShowBrowserBar method 405
Silent 414
StatusBar 414
StatusText 414
Stop 406
TheaterMode 414
TInternetExplorer events 416
TInternetExplorer Members 380
TInternetExplorer methods 384
TInternetExplorer properties 406
ToolBar 415
Top 415
TopLevelContainer 415
type_ 415
Visible 415
Width 416
TInvalidWebSession class 6190
about TInvalidWebSession class 6190
Create 6191
GetSessionID 6191
GetTimeoutMinutes 6192
GetValue 6192
SetTimeoutMinutes 6193
SetValue 6193
Terminate 6193
TInvalidWebSession Members 6190
TInvalidWebSession methods 6191
UpdateResponse 6194
TInvokeableVariantType class 5037
about TInvokeableVariantType class 5037
DispInvoke 5041
DoFunction 5041
DoProcedure 5042
FixupIdent 5043
GetProperty 5043
SetProperty 5044
TInvokeableVariantType Members 5038
TInvokeableVariantType methods 5040
TIpSocket class 1341
about TIpSocket class 1341
Bind 1344
Create 1343
Domain 1348
GetSocketAddr 1344
LocalDomainName 1344
LocalHost 1348
LocalHostAddr 1345
LocalHostName 1345
LocalPort 1349
LookupHostAddr 1345
LookupHostName 1346
LookupPort 1346
LookupProtocol 1347
ReceiveFrom 1347
RemoteHost 1350
RemotePort 1352
SendTo 1347
TIpSocket Members 1342
TIpSocket methods 1343
TIpSocket properties 1348
TISQLCommand class 1481
about TISQLCommand class 1481
close 1482
Destroy 1482
execute 1483
executeImmediate 1483
getErrorMessage 1483, 1484
getErrorMessage method 1483
getErrorMessageLen 1484
getNextCursor 1484
GetOption 1484
getParameter 1484
getRowsAffected 1485

GetStringOption 1485
prepare 1485
SetOption 1485
setParameter 1486
SetStringOption 1486
TISQLCommand Members 1481
TISQLCommand methods 1482
TISQLConnection class 1486
about TISQLConnection class 1486
beginTransaction 1488
commit 1488
connect 1488
connect method 1488
Create 1487
disconnect 1489
getErrorMessage 1489
getErrorMessage method 1489
getErrorMessageLen 1490
getFldDescClass 1490
GetOption 1490
getSQLCommand 1490
getSQLMetaData 1490
GetStringOption 1491
rollback 1491
SetOption 1491
SetStringOption 1491
TISQLConnection Members 1487
TISQLConnection methods 1487
TISQLCursor class 1492
about TISQLCursor class 1492
getBcd 1494
getBlob 1494
getBlobSize 1494
getBytes 1494
getColumnCount 1495
getColumnLength 1495
getColumnName 1495
getColumnName method 1495
getColumnNameLength 1496
getColumnPrecision 1496
getColumnScale 1496
getColumnType 1496
getCurObjectTypeName 1497
getDate 1497
getDouble 1497
getErrorMessage 1497, 1498
getErrorMessage method 1497
getErrorMessageLen 1498
getInt64 1498
getLong 1498
GetOption 1499
getShort 1499
getString 1499
GetStringOption 1499
getTime 1500
getTimeStamp 1500
getWideString 1500
isAutoIncrement 1500
isBlobSizeExact 1501
isNullable 1501
isReadOnly 1501
isSearchable 1501
next 1501
SetOption 1502
SetStringOption 1502
TISQLCursor Members 1492
TISQLCursor methods 1493
TISQLMetaData class 1502
about TISQLMetaData class 1502
getColumns 1503
getErrorMessage 1504
getErrorMessage method 1504
getErrorMessageLen 1504
getIndices 1504
getObjectList 1505
GetOption 1505
getProcedureParams 1505
getProcedures 1505

GetStringOption 1506
getTables 1506
SetOption 1506
SetStringOption 1506
TISQLMetaData Members 1503
TISQLMetaData methods 1503
TItemProp class 4808
about TItemProp class 4808
AssignTo 4810
Create 4809
Destroy 4809
EditMask 4811
EditStyle 4812
HasPickList 4810
KeyDesc 4812
MaxLength 4813
PickList 4813
ReadOnly 4814
TItemProp Members 4808
TItemProp methods 4810
TItemProp properties 4811
UpdateEdit 4811
tkArray enumeration member 4787
tkChar enumeration member 4787
tkClass enumeration member 4787
tkDynArray enumeration member 4787
tkEnumeration enumeration member 4787
tkFloat enumeration member 4787
tkInt64 enumeration member 4787
tkInteger enumeration member 4787
tkInterface enumeration member 4787
tkLString enumeration member 4787
tkMethod enumeration member 4787
tkRecord enumeration member 4787
tkSet enumeration member 4787
tkString enumeration member 4787
tkUnknown enumeration member 4787
tkUString enumeration member 4787
tkVariant enumeration member 4787
tkWChar enumeration member 4787
tkWString enumeration member 4787
TLabel class 2522
about TLabel class 2522
Align 2525
Alignment 2528
Anchors 2528
AutoSize 2529
BiDiMode 2530
Caption 2530
Color 2533
Constraints 2534
DragCursor 2534
DragKind 2535
DragMode 2535
EllipsisPosition 2537
Enabled 2537
FocusControl 2538
Font 2538
GlowSize 2539
Layout 2539
OnClick 2552
OnContextPopup 2554
OnDblClick 2554
OnDragDrop 2555
OnDragOver 2558
OnEndDock 2561
OnEndDrag 2561
OnMouseActivate 2562
OnMouseDown 2563
OnMouseEnter 2566
OnMouseLeave 2566
OnMouseMove 2566
OnMouseUp 2568
OnStartDock 2571
OnStartDrag 2571
ParentBiDiMode 2540
ParentColor 2540
ParentFont 2540

ParentShowHint 2541
PopupMenu 2542
ShowAccelChar 2547
ShowHint 2547
TLabel events 2551
TLabel Members 2522
TLabel properties 2524
Transparent 2549
Visible 2549
WordWrap 2551
TLanguages class 4051
about TLanguages class 4051
Count 4053
Create 4052
Ext 4053
ID 4054
IndexOf 4052
LocaleID 4054
Name 4054
NameFromLCID 4055
NameFromLocaleID 4055
TLanguages Members 4051
TLanguages methods 4052
TLanguages properties 4053
TLayoutAttributes class 5985
about TLayoutAttributes class 5985
ControlAttributes 5987
Create 5987
LabelAttributes 5987
LabelPosition 5988
TLayoutAttributes fields 5987
TLayoutAttributes Members 5986
TLayoutWebContent class 5988
_AddRef 5991
_Release 5991
about TLayoutWebContent class 5988
Create 5990
ImplLayoutButton 5991
ImplLayoutField 5992
ImplLayoutLabelAndField 5992
ImplLayoutTable 5993
LayoutButton 5994
LayoutField 5994
LayoutLabelAndField 5994
LayoutTable 5995
ParentLayout 5996
QueryInterface 5995
TLayoutWebContent Members 5989
TLayoutWebContent methods 5990
TLayoutWebContent properties 5996
tlBottom enumeration member 2914
tlbsCRLF enumeration member 3811
tlbsLF enumeration member 3811
tlCenter enumeration member 2914
tlEmptyText enumeration member 4643
TLineNumberMap class 6634
about TLineNumberMap class 6634
Count 6636
DestLineToSourceLine 6636
Items 6637
New 6636
TLineNumberMap Members 6635
TLineNumberMap methods 6636
TLineNumberMap properties 6636
TLinkedRIO class 1252
about TLinkedRIO class 1252
Create 1253
CreateFile 1254
Destroy 1254
GetResponseStream 1255
QueryInterface 1255
TLinkedRIO Members 1253
TLinkedRIO methods 1255
TLinkedRIO properties 1256
WebNode 1256
TLinkedWebNode class 1257
about TLinkedWebNode class 1257
BeforeExecute 1259

Create 1258
Destroy 1258
Execute 1260
Execute method 1260
GetMimeBoundary 1260
GetResponseStream 1261
Invoker 1262
MimeBoundary 1262
SetMimeBoundary 1261
TLinkedWebNode Members 1257
TLinkedWebNode methods 1259
TLinkedWebNode properties 1261
TListBox class 2572
about TListBox class 2572
Align 2576
Anchors 2580
AutoComplete 2580
AutoCompleteDelay 2580
BevelEdges 2581
BevelInner 2581
BevelKind 2582
BevelOuter 2582
BevelWidth 2582
BiDiMode 2583
BorderStyle 2583
Color 2584
Columns 2585
Constraints 2585
Ctl3D 2586
DoubleBuffered 2586
DragCursor 2588
DragKind 2588
DragMode 2589
Enabled 2590
ExtendedSelect 2591
Font 2591
ImeMode 2592
ImeName 2593
IntegralHeight 2593
ItemHeight 2594
Items 2594
MultiSelect 2597
OnClick 2612
OnContextPopup 2614
OnData 2615
OnDataFind 2615
OnDataObject 2616
OnDblClick 2616
OnDragDrop 2616
OnDragOver 2619
OnDrawItem 2623
OnEndDock 2625
OnEndDrag 2625
OnEnter 2626
OnExit 2629
OnKeyDown 2631
OnKeyPress 2635
OnKeyUp 2636
OnMeasureItem 2638
OnMouseActivate 2639
OnMouseDown 2640
OnMouseEnter 2643
OnMouseLeave 2643
OnMouseMove 2643
OnMouseUp 2646
OnStartDock 2648
OnStartDrag 2648
ParentBiDiMode 2597
ParentColor 2598
ParentCtl3D 2598
ParentDoubleBuffered 2599
ParentFont 2599
ParentShowHint 2599
PopupMenu 2600
ScrollWidth 2605
ShowHint 2605
Sorted 2607
Style 2608

TabOrder 2609
TabStop 2609
TabWidth 2610
TListBox events 2612
TListBox Members 2572
TListBox properties 2575
Visible 2610
tlListDetail enumeration member 4643
tlListGroup enumeration member 4643
tlListItemDisabled enumeration member 4643
tlListItemHot enumeration member 4643
tlListItemNormal enumeration member 4643
tlListItemSelected enumeration member 4643
tlListItemSelectedNotFocus enumeration member 4643
tlListSortDetail enumeration member 4643
tlListviewDontCare enumeration member 4643
tlListviewRoot enumeration member 4643
TLocalConnection class 4580
about TLocalConnection class 4580
AppServer 4591
AS_ApplyUpdates 4583
AS_DataRequest 4583
AS_Execute 4584
AS_GetParams 4585
AS_GetProviderNames 4586
AS_GetRecords 4586
AS_RowRequest 4587
Create 4581
Destroy 4582
GetConnected 4588
GetProvider 4588
GetProviderNames 4589
GetServer 4589
InterfaceSupportsErrorInfo 4589
Notification 4589
ProviderCount 4591
Providers 4592
RegisterProvider 4590
SafeCallException 4590
TLocalConnection Members 4581
TLocalConnection methods 4582
TLocalConnection properties 4591
UnRegisterProvider 4591
TLocateFileService class 6194
about TLocateFileService class 6194
OnFindIncludeFile 6195
OnFindStream 6196
OnFindTemplateFile 6196
TLocateFileService events 6195
TLocateFileService Members 6195
TLogicalPageList class 5805
about TLogicalPageList class 5805
Create 5810
TLogicalPageList Members 5805
TLoginFormAdapter class 5810
about TLoginFormAdapter class 5810
Actions 5816
Data 5817
OnAfterExecuteAction 5818
OnAfterGetActionResponse 5818
OnBeforeExecuteAction 5819
OnBeforeGetActionResponse 5819
OnGetActionParams 5819
OnLogin 5820
OnValidatePassword 5820
OnValidateUserName 5821
PasswordRequired 5817
TLoginFormAdapter events 5818
TLoginFormAdapter Members 5812
TLoginFormAdapter properties 5816
TLoginFormAdapterLoginAction class 5821
about TLoginFormAdapterLoginAction class 5821
DisplayLabel 5828
GetDefaultActionName 5826
ImplExecuteActionRequest 5826
ImplGetAdapterItemRequestParams 5827
NextPage 5827
OnAfterExecute 5828

OnAfterGetResponse 5829
OnBeforeExecute 5829
OnBeforeGetResponse 5830
OnGetParams 5830
TLoginFormAdapterLoginAction events 5828
TLoginFormAdapterLoginAction Members 5822
TLoginFormAdapterLoginAction methods 5826
TLoginFormAdapterLoginAction properties 5828
TLogLinkedWebNode class 1262
about TLogLinkedWebNode class 1262
Create 1263
Destroy 1264
Execute 1264
GetResponseStream 1265
ReqFile 1265
RespFile 1265
TLogLinkedWebNode Members 1263
TLogLinkedWebNode methods 1264
TLogLinkedWebNode properties 1265
tlTop enumeration member 2914
TMBCSEncoding class 4056
about TMBCSEncoding class 4056
Create 4057, 4058
Create constructor 4057
GetByteCount 4058
GetBytes 4059
GetCharCount 4059
GetChars 4060
GetMaxByteCount 4060
GetMaxCharCount 4060
GetPreamble 4061
TMBCSEncoding Members 4056
TMBCSEncoding methods 4058
tmChevron enumeration member 4643
TMemo class 2649
about TMemo class 2649
Align 2654
Alignment 2657
Anchors 2658
BevelEdges 2658
BevelInner 2659
BevelKind 2659
BevelOuter 2660
BiDiMode 2660
BorderStyle 2660
CharCase 2661
Color 2661
Constraints 2662
Ctl3D 2663
DoubleBuffered 2663
DragCursor 2665
DragKind 2665
DragMode 2666
Enabled 2667
Font 2668
HideSelection 2669
ImeMode 2669
ImeName 2670
Lines 2670
MaxLength 2671
OEMConvert 2672
OnChange 2690
OnClick 2690
OnContextPopup 2692
OnDblClick 2693
OnDragDrop 2693
OnDragOver 2696
OnEndDock 2699
OnEndDrag 2700
OnEnter 2701
OnExit 2703
OnKeyDown 2706
OnKeyPress 2710
OnKeyUp 2711
OnMouseActivate 2713
OnMouseDown 2714
OnMouseEnter 2716
OnMouseLeave 2716

OnMouseMove 2716
OnMouseUp 2719
OnStartDock 2721
OnStartDrag 2721
ParentBiDiMode 2672
ParentColor 2673
ParentCtl3D 2673
ParentDoubleBuffered 2673
ParentFont 2674
ParentShowHint 2674
PopupMenu 2675
ReadOnly 2680
ScrollBars 2680
ShowHint 2682
TabOrder 2684
TabStop 2684
TMemo events 2689
TMemo Members 2650
TMemo properties 2653
Visible 2685
WantReturns 2686
WantTabs 2687
WordWrap 2687
TMessage class 6923
about TMessage class 6923
AfterConstruction 6928
GetName 6929
GetParts 6929
SetName 6929
TMessage Members 6923
TMessage methods 6928
TMessages class 6930
about TMessages class 6930
Add 6935
GetMessage 6936
Insert 6936
TMessages Members 6930
TMessages methods 6935
TMetaClass class 3360
about TMetaClass class 3360
tmMenuBarDropDown enumeration member 4643
tmMenuBarItem enumeration member 4643
tmMenuDontCare enumeration member 4643
tmMenuDropDown enumeration member 4643
tmMenuItemDemoted enumeration member 4643
tmMenuItemNormal enumeration member 4643
tmMenuItemSelected enumeration member 4643
tmMenuRoot enumeration member 4643
TMonitor record 3361
about TMonitor record 3361
Enter 3362, 3363
Enter method 3362
Exit 3363
Pulse 3363
PulseAll 3363
SetSpinCount 3363
TMonitor Members 3362
TMonitor methods 3362
TryEnter 3364
Wait 3364
tmSeparator enumeration member 4643
TMultiModuleWebAppServices class 6197
about TMultiModuleWebAppServices class 6197
Access_Request 6199
Access_Response 6199
CreateContext 6200
DoAfterDispatch 6200
DoBeforeDispatch 6200
DoneModule 6201
FinishContext 6201
GetExceptionHandler 6202
HandleException 6202
HandleRequest 6203
ImplFinishContext 6203
ImplHandleException 6204
ImplHandleRequest 6204
ImplInitContext 6205
InitContext 6205

InitModule 6206
InitModules 6207
ModuleAdded 6207
OnAfterDispatch 6209
OnBeforeDispatch 6209
Request 6208
Response 6208
TMultiModuleWebAppServices events 6209
TMultiModuleWebAppServices Members 6197
TMultiModuleWebAppServices methods 6198
TMultiModuleWebAppServices properties 6207
WebModules 6208
TMultiReadExclusiveWriteSynchronizer class 4061
about TMultiReadExclusiveWriteSynchronizer class 4061
BeginRead 4064
BeginWrite 4064
Create 4063
Destroy 4063
EndRead 4065
EndWrite 4065
RevisionLevel 4066
TMultiReadExclusiveWriteSynchronizer Members 4062
TMultiReadExclusiveWriteSynchronizer methods 4064
TMultiReadExclusiveWriteSynchronizer properties 4066
TMutex class 3083
about TMutex class 3083
Acquire 3086
Create 3085
Create constructor 3084
Release 3086
TMutex Members 3083
TMutex methods 3085
TNamedVariantsList class 6106
about TNamedVariantsList class 6106
Clear 6108
Delete 6108
Destroy 6107
Get 6109
GetCapacity 6109
GetCount 6110
Insert 6110
Put 6111
SetCapacity 6111
TNamedVariantsList Members 6106
TNamedVariantsList methods 6108
TNotifyList class 910
about TNotifyList class 910
AddNotify 912
Count 914
Create 911
Destroy 912
GetCount 913
GetItem 913
Items 914
RemoveNotify 913
TNotifyList Members 910
TNotifyList methods 912
TNotifyList properties 914
TObject class 3364
about TObject class 3364
AfterConstruction 3367
BeforeDestruction 3368
ClassInfo 3368
ClassName 3369
ClassNameIs 3371
ClassParent 3372
ClassType 3374
CleanupInstance 3377
Create 3366
DefaultHandler 3377
Destroy 3366
Dispatch 3379
Equals 3381
FieldAddress 3382
FieldAddress method 3382
Free 3382
FreeInstance 3386
GetHashCode 3386

GetInterface 3387
GetInterfaceEntry 3387
GetInterfaceTable 3388
InheritsFrom 3388
InitInstance 3389
InstanceSize 3389
MethodAddress 3390
MethodAddress method 3389
MethodName 3390
NewInstance 3391
SafeCallException 3391
TObject Members 3365
TObject methods 3367
ToString 3391
UnitName 3392
ToolWin 4667
ToolWin namespace 4667
Classes 4668
Structs, Records, Enums 4678
Types 4681
ToolWin.TEdgeBorder 4678
ToolWin.TEdgeBorder enumeration 4678
ToolWin.TEdgeBorders 4681
ToolWin.TEdgeBorders type 4681
ToolWin.TEdgeStyle 4679
ToolWin.TEdgeStyle enumeration 4679
ToolWin.TSizingOrientation 4679
ToolWin.TSizingOrientation enumeration 4679
ToolWin.TToolDockForm 4668
ToolWin.TToolDockForm.CanResize 4669
ToolWin.TToolDockForm.Create 4669
ToolWin.TToolDockForm.CreateParams 4670
ToolWin.TToolDockForm.DoAddDockClient 4670
ToolWin.TToolDockForm.WMNCCreate 4670
ToolWin.TToolDockForm.WMNCHitTest 4670
ToolWin.TToolDockForm.WMNCLButtonDown 4671
ToolWin.TToolDockForm.WMSize 4671
ToolWin.TToolDockForm.WMSysCommand 4671
ToolWin.TToolDockObject 4671
ToolWin.TToolDockObject.AdjustDockRect 4673
ToolWin.TToolDockObject.Create 4672
ToolWin.TToolDockObject.DrawDragDockImage 4673
ToolWin.TToolDockObject.EraseDragDockImage 4673
ToolWin.TToolWindow 4673
ToolWin.TToolWindow.CMBorderChanged 4675
ToolWin.TToolWindow.CMCtl3DChanged 4675
ToolWin.TToolWindow.Create 4674
ToolWin.TToolWindow.EdgeBorders 4676
ToolWin.TToolWindow.EdgeInner 4676
ToolWin.TToolWindow.EdgeOuter 4677
ToolWin.TToolWindow.NCPaint 4675
ToolWin.TToolWindow.WMNCCalcSize 4675
ToolWin.TToolWindow.WMNCPaint 4676
TOperation class 6936
about TOperation class 6936
AfterConstruction 6942
GetFaults 6942
GetInput 6942
GetName 6942
GetOutput 6943
SetName 6943
TOperation Members 6936
TOperation methods 6941
TOperations class 6943
about TOperations class 6943
Add 6949
Add method 6949
GetOperation 6950
Insert 6950
TOperations Members 6944
TOperations methods 6949
TPacketInterceptFactory class 175
about TPacketInterceptFactory class 175
TPacketInterceptFactory Members 175
TPacketInterceptFactory methods 176
UpdateRegistry 176
TPagedAdapter class 5830
about TPagedAdapter class 5830

Actions 5836
Data 5836
ModifyAccess 5837
OnAfterExecuteAction 5838
OnAfterGetActionResponse 5838
OnAfterUpdateFields 5839
OnAfterValidateFields 5839
OnBeforeExecuteAction 5839
OnBeforeGetActionResponse 5840
OnBeforeUpdateFields 5840
OnBeforeValidateFields 5841
OnGetActionParams 5841
OnGetEOF 5841
OnGetFirstRecord 5842
OnGetNextRecord 5842
OnGetRecordCount 5842
OnGetRecordIndex 5842
PageSize 5837
TPagedAdapter events 5838
TPagedAdapter Members 5831
TPagedAdapter properties 5836
ViewAccess 5837
TPageDispatcher class 6210
about TPageDispatcher class 6210
DefaultPage 6212
OnAfterDispatchPage 6213
OnBeforeDispatchPage 6213
OnCanViewPage 6214
OnIsLoginRequired 6215
OnPageAccessDenied 6215
OnPageNotFound 6216
TPageDispatcher events 6212
TPageDispatcher Members 6210
TPageDispatcher properties 6212
TParam class 6950
about TParam class 6950
GetMessage 6956
GetName 6956
SetMessage 6956
SetName 6956
TParam Members 6951
TParam methods 6955
TPart class 6957
about TPart class 6957
GetElement 6962
GetName 6962
GetType_ 6963
SetElement 6963
SetName 6963
SetType_ 6963
TPart Members 6957
TPart methods 6962
TParts class 6964
about TParts class 6964
Add 6969
GetPart 6970
Insert 6970
TParts Members 6964
TParts methods 6969
tpBar enumeration member 4645
tpBarVert enumeration member 4645
tpChunk enumeration member 4645
tpChunkVert enumeration member 4645
tpDownDisabled enumeration member 4644
tpDownHorzDisabled enumeration member 4644
tpDownHorzHot enumeration member 4644
tpDownHorzNormal enumeration member 4644
tpDownHorzPressed enumeration member 4644
tpDownHot enumeration member 4644
tpDownNormal enumeration member 4644
tpDownPressed enumeration member 4644
TPort class 6970
about TPort class 6970
GetBinding 6976
GetName 6976
SetBinding 6976
SetName 6976
TPort Members 6971

TPort methods 6975
TPorts class 6977
about TPorts class 6977
Add 6982
GetPort 6983
Insert 6983
TPorts Members 6977
TPorts methods 6982
TPortType class 6983
about TPortType class 6983
AfterConstruction 6988
GetName 6989
GetOperations 6989
SetName 6989
TPortType Members 6983
TPortType methods 6988
TPortTypes class 6990
about TPortTypes class 6990
Add 6995
GetPortType 6996
Insert 6996
TPortTypes Members 6990
TPortTypes methods 6995
tpPageDontCare enumeration member 4644
tpPageRoot enumeration member 4644
tpProgressDontCare enumeration member 4645
tpProgressRoot enumeration member 4645
TPrintDlg class 1814
about TPrintDlg class 1814
BeforeExecute 1821
Caption 1816
Dialog 1816
Enabled 1817
GetDialogClass 1815
HelpContext 1817
HelpKeyword 1818
HelpType 1818
Hint 1819
ImageIndex 1819
OnAccept 1821
OnCancel 1821
OnHint 1821
OnUpdate 1822
SecondaryShortCuts 1819
ShortCut 1820
TPrintDlg events 1820
TPrintDlg Members 1814
TPrintDlg methods 1815
TPrintDlg properties 1815
Visible 1820
TPublishableVariantType class 4728
about TPublishableVariantType class 4728
GetInstance 4732
GetProperty 4732
SetProperty 4733
TPublishableVariantType Members 4729
TPublishableVariantType methods 4731
tpUpDisabled enumeration member 4644
tpUpHorzDisabled enumeration member 4644
tpUpHorzHot enumeration member 4644
tpUpHorzNormal enumeration member 4644
tpUpHorzPressed enumeration member 4644
tpUpHot enumeration member 4644
tpUpNormal enumeration member 4644
tpUpPressed enumeration member 4644
TPushButtonActionLink class 2722
about TPushButtonActionLink class 2722
IsImageIndexLinked 2723
SetImageIndex 2724
TPushButtonActionLink Members 2723
TPushButtonActionLink methods 2723
TRadioButton class 2724
about TRadioButton class 2724
Action 2730
Align 2730
Alignment 2734
Anchors 2734
BiDiMode 2735

Caption 2735
Checked 2738
Color 2738
Constraints 2739
Create 2726
CreateParams 2727
CreateWnd 2728
Ctl3D 2740
DoubleBuffered 2740
DragCursor 2742
DragKind 2742
DragMode 2743
Enabled 2744
Font 2745
GetChecked 2728
GetControlsAlignment 2728
OnClick 2758
OnContextPopup 2760
OnDblClick 2761
OnDragDrop 2761
OnDragOver 2764
OnEndDock 2767
OnEndDrag 2768
OnEnter 2769
OnExit 2771
OnKeyDown 2774
OnKeyPress 2777
OnKeyUp 2779
OnMouseActivate 2780
OnMouseDown 2781
OnMouseEnter 2784
OnMouseLeave 2784
OnMouseMove 2784
OnMouseUp 2786
OnStartDock 2789
OnStartDrag 2789
ParentBiDiMode 2746
ParentColor 2746
ParentCtl3D 2747
ParentDoubleBuffered 2747
ParentFont 2747
ParentShowHint 2748
PopupMenu 2748
SetChecked 2729
ShowHint 2753
TabOrder 2755
TabStop 2755
TRadioButton events 2758
TRadioButton Members 2725
TRadioButton methods 2727
TRadioButton properties 2729
Visible 2756
WordWrap 2757
TranslateException class 7670
about TranslateException class 7670
TRawSocket class 1353
about TRawSocket class 1353
Create 1355
SockType 1355
TRawSocket Members 1353
TRawSocket properties 1355
trBandChecked enumeration member 4646
trBandDisabled enumeration member 4646
trBandHot enumeration member 4646
trBandHotChecked enumeration member 4646
trBandNormal enumeration member 4646
trBandPressed enumeration member 4646
trChevronDisabled enumeration member 4646
trChevronHot enumeration member 4646
trChevronNormal enumeration member 4646
trChevronPressed enumeration member 4646
trChevronVertDisabled enumeration member 4646
trChevronVertHot enumeration member 4646
trChevronVertNormal enumeration member 4646
trChevronVertPressed enumeration member 4646
trGripper enumeration member 4646
trGripperVert enumeration member 4646
trRebarDontCare enumeration member 4646

trRebarRoot enumeration member 4646
tsArrowBtnDownDisabled enumeration member 4647
tsArrowBtnDownHot enumeration member 4647
tsArrowBtnDownNormal enumeration member 4647
tsArrowBtnDownPressed enumeration member 4647
tsArrowBtnLeftDisabled enumeration member 4647
tsArrowBtnLeftHot enumeration member 4647
tsArrowBtnLeftNormal enumeration member 4647
tsArrowBtnLeftPressed enumeration member 4647
tsArrowBtnRightDisabled enumeration member 4647
tsArrowBtnRightHot enumeration member 4647
tsArrowBtnRightNormal enumeration member 4647
tsArrowBtnRightPressed enumeration member 4647
tsArrowBtnUpDisabled enumeration member 4647
tsArrowBtnUpHot enumeration member 4647
tsArrowBtnUpNormal enumeration member 4647
tsArrowBtnUpPressed enumeration member 4647
TScreenTipBorderStyle class 280
- about TScreenTipBorderStyle class 280
- Color 281
- CornerSize 281
- Create 280
- Style 281
- TScreenTipBorderStyle Members 280
- TScreenTipBorderStyle properties 281

TScreenTipItem class 282
- about TScreenTipItem class 282
- Action 286
- Assign 284
- Create 283
- Description 286
- Destroy 283
- DisabledDescription 286
- DisabledHeader 286
- GetDisplayName 284
- Header 286
- Height 287
- Image 287
- IsEnabled 285
- Manager 287
- SetManager 285
- ShowFooter 287
- ShowImage 288
- TScreenTipItem Members 282
- TScreenTipItem methods 283
- TScreenTipItem properties 285
- Width 288

TScreenTipsCollection class 288
- about TScreenTipsCollection class 288
- Add 289
- Create 289
- GetOwner 290
- Insert 290
- Items 291
- TScreenTipsCollection Members 288
- TScreenTipsCollection methods 289
- TScreenTipsCollection properties 291

TScreenTipsManager class 291
- about TScreenTipsManager class 291
- BorderStyle 293
- DisplayOffset 293
- Font 293
- Footer 294
- FooterImage 294
- GradientEndColor 294
- GradientStartColor 294
- LinkedActionLists 294
- OnCreateScreenTip 295
- OnDuplicateCreateError 296
- OnGetScreenTipWinClass 296
- OnScreenTipsLoaded 296
- OnShowScreenTip 296
- Options 295
- ScreenTips 295
- TScreenTipsManager events 295
- TScreenTipsManager Members 292
- TScreenTipsManager properties 293

TScreenTipsPopup class 297

about TScreenTipsPopup class 297
Align 298
Anchors 301
Associate 302
Glyph 302
PopupType 302
ScreenTip 303
ScreenTipManager 303
Transparent 303
TScreenTipsPopup Members 297
TScreenTipsPopup properties 298
Visible 303
TScreenTipsWindow class 305
about TScreenTipsWindow class 305
ActivateToolTip 307
ActiveItem 311
Body 312
CMMouseenter 307
CMMouseLeave 308
CornerSize 312
Create 306
CreateParams 308
CreateRoundedStyle 308
CreateTipBorderStyle 309
CreateWnd 309
Destroy 306
DrawBackground 309
DrawBorder 310
DrawFooter 310
DrawHeader 310
Footer 312
FooterImage 312
GradientEndColor 313
GradientStartColor 313
HideHint 310
HintControl 313
Image 313
Paint 310
TipBorderStyle 313
TScreenTipsWindow Members 305
TScreenTipsWindow methods 307
TScreenTipsWindow properties 311
WMNCHitTest 311
TScriptCodeItem class 6637
about TScriptCodeItem class 6637
AssignString 6638
GetScript 6639
TScriptCodeItem Members 6637
TScriptCodeItem methods 6638
TScriptComponent class 915
about TScriptComponent class 915
Create 917
GetComponent 917
Obj 917
TScriptComponent Members 915
TScriptComponent methods 917
TScriptComponent properties 917
TScriptEnginesList class 6639
about TScriptEnginesList class 6639
AddEngine 6641
AddObjectFactory 6641
Count 6642
Create 6640
Destroy 6640
Find 6641
FindScriptProducerClass 6642
Items 6643
ListLanguageNames 6642
RemoveEngine 6642
TScriptEnginesList Members 6639
TScriptEnginesList methods 6641
TScriptEnginesList properties 6642
TScriptError class 6643
about TScriptError class 6643
Create 6644
FCharPos 6644
FDescription 6644
FFileName 6645

FLine 6645
FSourceLine 6645
GetCharPos 6645
GetDescription 6646
GetFileName 6646
GetLine 6646
GetSourceLine 6646
SetDescription 6647
TScriptError fields 6644
TScriptError Members 6643
TScriptError methods 6645
TScriptErrors class 6647
about TScriptErrors class 6647
Add 6649
Create 6647
Destroy 6648
GetError 6649
GetErrorCount 6649
TScriptErrors Members 6647
TScriptErrors methods 6648
TScriptFile class 6649
about TScriptFile class 6649
ApplicationName 6652
Create 6650
Destroy 6650
FLineNumberMap 6651
LongName 6652
ParseStream 6651
ParseString 6651
Script 6652
ShortName 6653
Source 6653
TScriptFile fields 6651
TScriptFile Members 6650
TScriptFile methods 6651
TScriptFile properties 6652
TScriptGlobalObjects class 6653
about TScriptGlobalObjects class 6653
AddNamedIntf 6656
Create 6654
Destroy 6655
FindNamedItemIntf 6656
GetNamedItemCount 6656
GetNamedItemName 6656
GetScriptEngine 6656
NamedItemCount 6657
NamedItemName 6657
Producer 6657
ScriptEngine 6658
TScriptGlobalObjects Members 6653
TScriptGlobalObjects methods 6655
TScriptGlobalObjects properties 6657
TScriptIncludeFile class 6658
about TScriptIncludeFile class 6658
Create 6659
LoadFile 6659
TScriptIncludeFile Members 6658
TScriptIncludeFile methods 6659
TScriptIncludeFiles class 6659
about TScriptIncludeFiles class 6659
Create 6660
GetScriptIncludeFile 6660
Items 6661
TScriptIncludeFiles Members 6660
TScriptIncludeFiles methods 6660
TScriptIncludeFiles properties 6661
TScriptIncludeItem class 6662
about TScriptIncludeItem class 6662
Create 6663
GetScript 6663
TScriptIncludeItem Members 6662
TScriptIncludeItem methods 6663
TScriptItem class 6664
about TScriptItem class 6664
Create 6665
GetScript 6665
Script 6666
TScriptItem Members 6664

TScriptItem methods 6665
TScriptItem properties 6665
TScriptItems class 6666
about TScriptItems class 6666
Create 6667
GetScriptItem 6667
Items 6667
TScriptItems Members 6666
TScriptItems methods 6667
TScriptItems properties 6667
TScriptObject class 918
about TScriptObject class 918
TScriptObjectFactories class 6668
about TScriptObjectFactories class 6668
Count 6672
Create 6670
CreateGlobalObject 6671
Destroy 6670
Factories 6672
GetCount 6671
GetFactory 6671
GetObjectList 6672
TScriptObjectFactories Members 6669
TScriptObjectFactories methods 6671
TScriptObjectFactories properties 6672
TScriptProducer class 6673
about TScriptProducer class 6673
Component 6681
Content 6681
Create 6674
Designing 6681
Destroy 6674
Errors 6681
Evaluate 6676
EvaluateExpression 6676
FErrors 6675
FScriptEngine 6675
FScriptFile 6676
GetContent 6677
GetErrors 6677
GetHTMLBlock 6677
GetHTMLBlockCount 6677
GetWebModuleContext 6678
HandleParserError 6678
HandleScriptError 6678
ImplHandleScriptError 6678
ParseStream 6678
ParseString 6679
ReplaceTags 6679
Script 6682
SetContent 6679
TScriptProducer fields 6675
TScriptProducer Members 6673
TScriptProducer methods 6676
TScriptProducer properties 6680
Write 6680
Write method 6679
WriteItem 6680
TScrollBar class 2790
about TScrollBar class 2790
Align 2795
Anchors 2799
BiDiMode 2799
Change 2793
Constraints 2800
Create 2792
CreateParams 2793
CreateWnd 2793
Ctl3D 2800
DoubleBuffered 2801
DragCursor 2802
DragKind 2803
DragMode 2803
Enabled 2805
Kind 2805
LargeChange 2806
Max 2808
Min 2810

OnChange 2829
OnContextPopup 2829
OnDragDrop 2830
OnDragOver 2833
OnEndDock 2836
OnEndDrag 2836
OnEnter 2837
OnExit 2840
OnKeyDown 2842
OnKeyPress 2846
OnKeyUp 2847
OnMouseEnter 2849
OnMouseLeave 2849
OnScroll 2849
OnStartDock 2851
OnStartDrag 2852
PageSize 2812
ParentBiDiMode 2814
ParentCtl3D 2814
ParentDoubleBuffered 2814
ParentShowHint 2815
PopupMenu 2815
Position 2820
Scroll 2794
SetParams 2794
ShowHint 2822
SmallChange 2824
TabOrder 2826
TabStop 2826
TScrollBar events 2828
TScrollBar Members 2791
TScrollBar methods 2792
TScrollBar properties 2794
Visible 2827
TScroller class 4494
about TScroller class 4494
Change 4497
Create 4495
Destroy 4496
Max 4497
Min 4498
OnClick 4499
Paint 4496
Position 4498
ScrollOrientation 4498
TScroller events 4499
TScroller Members 4494
TScroller methods 4496
TScroller properties 4496
tsDownDisabled enumeration member 4649
tsDownHorzDisabled enumeration member 4649
tsDownHorzHot enumeration member 4649
tsDownHorzNormal enumeration member 4649
tsDownHorzPressed enumeration member 4649
tsDownHot enumeration member 4649
tsDownNormal enumeration member 4649
tsDownPressed enumeration member 4649
TSearchAction class 1823
about TSearchAction class 1823
Create 1824
Destroy 1825
ExecuteTarget 1826
FControl 1825
FFindFirst 1826
HandlesTarget 1827
Notification 1827
Search 1827
TSearchAction fields 1825
TSearchAction Members 1824
TSearchAction methods 1826
UpdateTarget 1828
TSearchFind class 1828
about TSearchFind class 1828
BeforeExecute 1835
Caption 1831
Dialog 1831
Enabled 1832
GetDialogClass 1830

HelpContext 1832
HelpKeyword 1832
HelpType 1833
Hint 1833
ImageIndex 1834
OnAccept 1836
OnCancel 1836
OnHint 1836
OnUpdate 1837
SecondaryShortCuts 1834
ShortCut 1835
TSearchFind events 1835
TSearchFind Members 1829
TSearchFind methods 1830
TSearchFind properties 1831
Visible 1835
TSearchFindFirst class 1838
about TSearchFindFirst class 1838
Create 1840
TSearchFindFirst Members 1839
TSearchFindNext class 1840
about TSearchFindNext class 1840
Caption 1844
Create 1842
Enabled 1844
ExecuteTarget 1842
HandlesTarget 1843
HelpContext 1844
HelpKeyword 1845
HelpType 1845
Hint 1846
ImageIndex 1846
OnHint 1848
OnUpdate 1849
SearchFind 1847
SecondaryShortCuts 1847
ShortCut 1847
TSearchFindNext events 1848
TSearchFindNext Members 1841
TSearchFindNext methods 1842
TSearchFindNext properties 1843
UpdateTarget 1843
Visible 1848
TSearchReplace class 1850
about TSearchReplace class 1850
BeforeExecute 1858
Caption 1853
Dialog 1853
Enabled 1854
ExecuteTarget 1852
GetDialogClass 1852
HelpContext 1854
HelpKeyword 1855
HelpType 1855
Hint 1856
ImageIndex 1856
OnAccept 1858
OnCancel 1858
OnHint 1858
OnUpdate 1859
SecondaryShortCuts 1856
ShortCut 1857
TSearchReplace events 1857
TSearchReplace Members 1850
TSearchReplace methods 1852
TSearchReplace properties 1853
Visible 1857
TSemaphore class 3086
about TSemaphore class 3086
Acquire 3089
Create 3088, 3089
Create constructor 3088
Release 3090
Release method 3089
TSemaphore Members 3087
TSemaphore methods 3089
TServerAcceptThread class 80
about TServerAcceptThread class 80

Create 80
Execute 81
ServerSocket 85
TServerAcceptThread Members 80
TServerAcceptThread methods 80
TServerAcceptThread properties 84
TServerClientThread class 85
about TServerClientThread class 85
ClientExecute 88
ClientSocket 91
Create 86
Data 92
Destroy 87
DoTerminate 88
EndConnect 89
Error 89
Event 90
Execute 90
HandleException 90
KeepInCache 92
ReActivate 91
ServerSocket 93
StartConnect 91
TServerClientThread Members 86
TServerClientThread methods 87
TServerClientThread properties 91
TServerClientWinSocket class 93
about TServerClientWinSocket class 93
Create 95
Destroy 96
ServerWinSocket 96
TServerClientWinSocket Members 94
TServerClientWinSocket properties 96
TServerSocket class 97
about TServerSocket class 97
Active 100
Create 99
OnAccept 103
OnClientConnect 104
OnClientDisconnect 105
OnClientError 105
OnClientRead 106
OnClientWrite 106
OnGetSocket 107
OnGetThread 107
OnListen 108
OnThreadEnd 108
OnThreadStart 109
Port 100
ServerType 101
Service 101
Socket 102
ThreadCacheSize 102
TServerSocket events 103
TServerSocket Members 97
TServerSocket properties 99
TServerSocketThread class 1356
about TServerSocketThread class 1356
AddClientSocketThread 1358
ClearThreadPool 1358
Create 1357
CreateThread 1359
Destroy 1357
Execute 1359
FetchClientSocketThread 1359
OnGetThread 1361
RemoveClientSocketThread 1359
ServerSocket 1360
ThreadCacheSize 1360
ThreadPool 1361
TServerSocketThread events 1361
TServerSocketThread Members 1356
TServerSocketThread methods 1358
TServerSocketThread properties 1360
TServerWinSocket class 109
about TServerWinSocket class 109
Accept 113
ActiveConnections 118

ActiveThreads 119
ClientConnect 114
ClientDisconnect 114
ClientErrorEvent 114
ClientRead 114
ClientWrite 115
Connections 119
Create 112
Destroy 112
Disconnect 115
DoCreateThread 115
GetClientSocket 116
GetClientThread 116
GetServerThread 116
IdleThreads 120
Listen 116
OnClientConnect 122
OnClientDisconnect 123
OnClientError 124
OnClientRead 124
OnClientWrite 125
OnGetSocket 126
OnGetThread 126
OnThreadEnd 127
OnThreadStart 127
ServerType 120
SetServerType 117
SetThreadCacheSize 117
ThreadCacheSize 121
ThreadEnd 118
ThreadStart 118
TServerWinSocket events 121
TServerWinSocket Members 110
TServerWinSocket methods 113
TServerWinSocket properties 118
TService class 3017, 6996
about TService class 3017, 6996
AfterConstruction 7001
AfterInstall 3031
AfterUninstall 3032
AllowPause 3024
AllowStop 3024
BeforeInstall 3032
BeforeUninstall 3033
Controller 3020
CreateNew 3019
Dependencies 3024
Destroy 3019
DisplayName 3025
DoContinue 3020
DoCustomControl 3020
DoInterrogate 3020
DoPause 3021
DoShutdown 3021
DoStart 3021
DoStop 3021
ErrCode 3025
ErrorSeverity 3026
GetName 7002
GetPorts 7002
GetServiceController 3022
Interactive 3026
LoadGroup 3026
LogMessage 3022
Main 3023
OnContinue 3033
OnExecute 3034
OnPause 3034
OnShutdown 3034
OnStart 3035
OnStop 3035
Param 3027
ParamCount 3027
Password 3027
ReportStatus 3023
ServiceStartName 3028
ServiceThread 3028
ServiceType 3029

SetName 7002
StartType 3029
Status 3029
TagID 3030
Terminated 3030
TService events 3031
TService Members 3017, 6996
TService methods 3019, 7001
TService properties 3023
WaitHint 3030
Win32ErrCode 3031
TServiceApplication class 3035
about TServiceApplication class 3035
Create 3037
CreateForm 3038
DelayInitialize 3040
Destroy 3037
DispatchServiceMain 3038
DoHandleException 3038
Hook 3039
Initialize 3039
Installing 3039
RegisterServices 3039
Run 3040
ServiceCount 3040
Title 3041
TServiceApplication Members 3036
TServiceApplication methods 3037
TServiceApplication properties 3040
TServices class 7003
about TServices class 7003
Add 7008
GetService 7009
Insert 7009
TServices Members 7003
TServices methods 7008
TServiceThread class 3041
about TServiceThread class 3041
Create 3042
Execute 3043
ProcessRequests 3046
TServiceThread Members 3042
TServiceThread methods 3042
TSessionIDGen class 331
about TSessionIDGen class 331
Create 331
NextID 332
TSessionIDGen Members 331
TSessionIDGen methods 332
TSessionItem class 332
about TSessionItem class 332
Assign 334
Create 333
Destroy 333
ExpirationTime 335
Expired 335
ID 336
IsActive 335
Items 336
StartTime 336
StatusFlags 336
Terminated 337
TimeoutMinutes 337
TouchTime 337
TSessionItem Members 332
TSessionItem methods 334
TSessionItem properties 335
TSessions class 337
about TSessions class 337
ActiveCount 347
CheckStatusChange 340
ConnectActiveSession 340
ConnectSession 340
Create 338
CreateSessionIDGen 341
Destroy 339
DisconnectSession 341
FileName 347

GetExpirationTime 341
GetExpired 341
GetIDCount 341
GetIDs 342
GetIsActive 342
GetItemCount 342
GetItems 342
GetSession 343
GetSessionIDGen 343
GetStartTime 343
GetTerminated 343
GetTimeoutMinutes 343
GetTouchTime 344
ItemValue 344
LoadFromFile 344
LoadFromStream 344
MaxSessions 347
NotifyStatusChange 345
SaveToFile 345
SaveToStream 345
SessionExists 345
SetItemValue 345
SetTimeoutMinutes 346
StartSession 346
TerminateSession 346
TouchSession 346
TSessions Members 338
TSessions methods 339
TSessions properties 346
TSessionsService class 6722
about TSessionsService class 6722
DefaultTimeout 6724
MaxSessions 6724
OnEndSession 6725
OnStartSession 6725
TSessionsService events 6725
TSessionsService Members 6723
TSessionsService properties 6724
tsGripper enumeration member 4650
tsGripperHorzDisabled enumeration member 4647
tsGripperHorzHot enumeration member 4647
tsGripperHorzNormal enumeration member 4647
tsGripperHorzPressed enumeration member 4647
tsGripperPane enumeration member 4650
tsGripperVertDisabled enumeration member 4647
tsGripperVertHot enumeration member 4647
tsGripperVertNormal enumeration member 4647
tsGripperVertPressed enumeration member 4647
TShadowWindow class 352
about TShadowWindow class 352
Control 357
Create 353
CreateParams 354
CreateShadow 353
Destroy 353
Notification 355
Paint 355
SetBounds 356
Side 357
TShadowWindow Members 352
TShadowWindow methods 354
TShadowWindow properties 357
WMEraseBkgnd 356
TShellNameSpace class 424
about TShellNameSpace class 424
Columns 438
Connect 428
ConnectTo 428
CountViewTypes 438
Create 425
CreateSubscriptionForSelection 429
DefaultInterface 438
DeleteSubscriptionForSelection 429
Depth 438
Destroy 426
Disconnect 429
EnumOptions 439
Expand 430

Export 430
Flags 439
Get_Columns 430
Get_CountViewTypes 430
Get_Depth 430
Get_EnumOptions 431
Get_Flags 431
Get_Mode 431
Get_Root 431
Get_SelectedItem 432
Get_SubscriptionsEnabled 432
Get_TVFlags 432
Import 432
InitServerData 432
InvokeContextMenuCommand 433
InvokeEvent 433
Mode 439
MoveSelectionDown 433
MoveSelectionTo 434
MoveSelectionUp 434
NewFolder 434
OnDoubleClick 440
OnFavoritesSelectionChange 441
OnInitialized 441
OnSelectionChange 441
ResetSort 434
Root 439
SelectedItem 440
SelectedItems 435
Set_Columns 435
Set_Depth 435
Set_EnumOptions 435
Set_Flags 435
Set_Mode 436
Set_Root 436
Set_SelectedItem 436
Set_TVFlags 436
SetRoot 437
SetViewType 437
SubscriptionsEnabled 440
Synchronize 437
TShellNameSpace events 440
TShellNameSpace Members 424
TShellNameSpace methods 427
TShellNameSpace properties 437
TVFlags 440
UnselectAll 437
TShellResources class 828
about TShellResources class 828
TShellUIHelper class 441
about TShellUIHelper class 441
AddChannel 444
AddDesktopComponent 445, 446
AddDesktopComponent method 445
AddFavorite 446, 447
AddFavorite method 446
AutoCompleteAttach 447
AutoCompleteAttach method 447
AutoCompleteSaveForm 448
AutoCompleteSaveForm method 448
AutoScan 448, 449
AutoScan method 448
Connect 449
ConnectTo 449
Create 442
DefaultInterface 452
Destroy 443
Disconnect 450
ImportExportFavorites 450
InitServerData 450
IsSubscribed 451
NavigateAndFind 451
RefreshOfflineDesktop 451
ResetFirstBootMode 451
ResetSafeMode 452
ShowBrowserUI 452
TShellUIHelper Members 442
TShellUIHelper methods 444

TShellUIHelper properties 452
TShellWindows class 452
_NewEnum 456
about TShellWindows class 452
Connect 456
ConnectTo 456
Count 461
Create 453
DefaultInterface 461
Destroy 454
Disconnect 457
FindWindowSW 457
Get_Count 457
InitServerData 457
InvokeEvent 458
Item 458, 459
Item method 458
OnActivated 459
OnCreated 459
OnNavigate 459
OnWindowRegistered 461
OnWindowRevoked 462
ProcessAttachDetach 460
Register 460
RegisterPending 460
Revoke 460
TShellWindows events 461
TShellWindows Members 453
TShellWindows methods 455
TShellWindows properties 460
TSimpleDataSet class 838
about TSimpleDataSet class 838
Active 846
AfterApplyUpdates 873
AfterCancel 873
AfterClose 875
AfterDelete 877
AfterEdit 879
AfterExecute 880
AfterGetParams 881
AfterGetRecords 882
AfterInsert 882
AfterOpen 883
AfterPost 884
AfterRefresh 885
AfterRowRequest 885
AfterScroll 886
Aggregates 848
AggregatesActive 849
AllocConnection 842
AllocDataSet 842
AllocProvider 843
AutoCalcFields 849
BeforeApplyUpdates 886
BeforeCancel 887
BeforeClose 888
BeforeDelete 888
BeforeEdit 889
BeforeExecute 889
BeforeGetParams 890
BeforeGetRecords 891
BeforeInsert 892
BeforeOpen 894
BeforePost 894
BeforeRefresh 895
BeforeRowRequest 895
BeforeScroll 896
Connection 850
Constraints 850
Create 841
DataSet 851
Destroy 842
DisableStringTrim 851
FetchOnDemand 852
FieldDefs 853
FileName 856
Filter 857
Filtered 859

FilterOptions 859
IndexDefs 860
IndexFieldNames 861
IndexName 862
Loaded 843
MasterFields 863
MasterSource 863
Notification 843
ObjectView 864
OnCalcFields 897
OnDeleteError 897
OnEditError 898
OnFilterRecord 898
OnNewRecord 899
OnPostError 899
OnReconcileError 900
OpenCursor 844
PacketRecords 865
Params 866
PSGetCommandText 844
ReadOnly 871
SetConnection 845
StoreDefs 871
TSimpleDataSet events 872
TSimpleDataSet Members 839
TSimpleDataSet methods 842
TSimpleDataSet properties 845
TSimpleEvent class 3090
about TSimpleEvent class 3090
TSimpleRWSync class 4066
about TSimpleRWSync class 4066
BeginRead 4069
BeginWrite 4069
Create 4068
Destroy 4068
EndRead 4070
EndWrite 4070
TSimpleRWSync Members 4067
TSimpleRWSync methods 4069
TSiteModuleHelper class 6531
about TSiteModuleHelper class 6531
Create 6533
Destroy 6533
EndIterator 6534
FindNamedVariable 6534
FindVariable 6535
FindVariableContainer 6535
GetCurrentObject 6535
GetVariable 6535
GetVariableCount 6535
Module 6537
NextIteration 6536
Notification 6536
NotifyActivate 6536
NotifyDeactivate 6536
OnActivate 6539
OnDeactivate 6539
Request 6537
Response 6538
Session 6538
StartIterator 6537
TSiteModuleHelper events 6539
TSiteModuleHelper Members 6532
TSiteModuleHelper methods 6534
TSiteModuleHelper properties 6537
WebModuleContext 6538
TSitePageElements class 1107
about TSitePageElements class 1107
ServerFunctionsContent 1108
ServerScriptContent 1109
ServerVariablesContent 1109
StylesContent 1109
TSitePageElements fields 1108
TSitePageElements Members 1107
WarningsContent 1109
TSitePageModuleHelper class 6539
about TSitePageModuleHelper class 6539
Create 6542

DefaultAction 6548
Destroy 6542
DispatchPage 6543
GetDefaultAction 6543
GetPageProducer 6544
GetProducerComponent 6544
GetRedirectParams 6544
GetTemplateFileType 6544
GetTemplateStream 6545
GetXMLData 6545
GetXSLData 6545
HasHTMLBrowserView 6545
HasScriptView 6545
HasXMLBrowserView 6546
HasXSLBrowserView 6546
HTMLFromTemplate 6546
ImplDispatchPage 6546
ImplIncludePage 6547
ImplRedirectToPage 6547
ImplSetWebContentOptions 6547
IncludePage 6547
OnAfterDispatchPage 6549
OnAfterRedirectToPage 6549
OnBeforeDispatchPage 6550
OnBeforeRedirectToPage 6550
RedirectToPage 6547
SetDefaultPageFileName 6548
SetWebContentOptions 6548
TSitePageModuleHelper events 6549
TSitePageModuleHelper Members 6540
TSitePageModuleHelper methods 6543
TSitePageModuleHelper properties 6548
tsLowerTrackHorzDisabled enumeration member 4647
tsLowerTrackHorzHot enumeration member 4647
tsLowerTrackHorzNormal enumeration member 4647
tsLowerTrackHorzPressed enumeration member 4647
tsLowerTrackVertDisabled enumeration member 4647
tsLowerTrackVertHot enumeration member 4647
tsLowerTrackVertNormal enumeration member 4647
tsLowerTrackVertPressed enumeration member 4647
tsModernPopout enumeration member 4575
tsModernTabs enumeration member 4575
TSoapConnection class 1128
about TSoapConnection class 1128
AfterConnect 1141
AfterDisconnect 1141
Agent 1136
BeforeConnect 1142
BeforeDisconnect 1142
Connected 1136
Create 1130
Destroy 1131
DoConnect 1131
DoDisconnect 1132
GetConnected 1132
GetProviderNames 1132
GetServer 1133
GetServerList 1133
GetSOAPServer 1134
GetSOAPServerIID 1134
Notification 1134
OnAfterExecute 1143
OnBeforeExecute 1143
Password 1137
Proxy 1137
ProxyByPass 1137
RIO 1138
SetSOAPServerIID 1135
SetUseSoapAdapter 1135
SOAPServerIID 1138
TSoapConnection events 1141
TSoapConnection Members 1129
TSoapConnection methods 1131
TSoapConnection properties 1135
URL 1139
UserName 1140
UseSOAPAdapter 1140
TSoapDataList class 1112

about TSoapDataList class 1112
Add 1113
Assign 1113
Clear 1114
Insert 1114
Remove 1114
TSoapDataList Members 1112
TSoapDataList methods 1113
TSoapDataModule class 1146
_AddRef 1149
_Release 1150
about TSoapDataModule class 1146
AfterConstruction 1150
ApplyUpdates 1151
AS_ApplyUpdates 1151
AS_DataRequest 1152
AS_Execute 1153
AS_GetParams 1154
AS_GetProviderNames 1155
AS_GetRecords 1155
AS_RowRequest 1156
BeforeDestruction 1157
Create 1148
DataRequest 1157
Destroy 1148
Execute 1158
GetParams 1158
GetProvider 1159
GetProviderNames 1159
GetRecords 1160
Lock 1161
NewInstance 1161
ProviderCount 1168
Providers 1168
RefCount 1168
RegisterProvider 1162
RowRequest 1162
SafeCallException 1162
SAS_ApplyUpdates 1163
SAS_DataRequest 1163
SAS_Execute 1164
SAS_GetParams 1165
SAS_GetProviderNames 1165
SAS_GetRecords 1166
SAS_RowRequest 1166
TSoapDataModule Members 1147
TSoapDataModule methods 1149
TSoapDataModule properties 1167
Unlock 1167
UnRegisterProvider 1167
TSOAPDOMProcessor class 1172
_AddRef 1174
_Release 1175
about TSOAPDOMProcessor class 1172
AddHeaderProcessor 1175
AfterConstruction 1176
DefaultProcessHeader 1176
Destroy 1173
FHeaderProcessors 1173
FindHeaderProcessor 1177
MakeHeaderNodes 1177
NewInstance 1177
TSOAPDOMProcessor fields 1173
TSOAPDOMProcessor Members 1172
TSOAPDOMProcessor methods 1174
TSoapPascalInvoker class 1284
about TSoapPascalInvoker class 1284
AfterDispatchEvent 1290
BeforeDispatchEvent 1291
BeforeDispatchEvent2 1291
Converter 1289
Create 1285
Destroy 1286
FConverter 1286
FDomConverter 1286
FHeadersIn 1287
FHeadersOut 1287
FSOAPHeaders 1287

GetDomConverter 1288
GetInvoker 1288
Invoke 1288
Notification 1289
OnExceptionEvent 1291
OnGetAttachment 1292
OnSendAttachment 1292
TSoapPascalInvoker events 1290
TSoapPascalInvoker fields 1286
TSoapPascalInvoker Members 1284
TSoapPascalInvoker methods 1287
TSoapPascalInvoker properties 1289
TSocketConnection class 176
about TSocketConnection class 176
Address 180
Create 178
CreateTransport 179
DoConnect 179
Host 181
InterceptGUID 181
InterceptName 182
ObjectBroker 183
Port 183
SupportCallbacks 184
TSocketConnection Members 177
TSocketConnection methods 179
TSocketConnection properties 180
TSocketTransport class 185
about TSocketTransport class 185
Address 190
Create 186
Destroy 187
GetConnected 188
GetWaitEvent 188
Host 190
InterceptGUID 190
Port 190
Receive 188
Send 189
SetConnected 189
Socket 191
TSocketTransport Members 185
TSocketTransport methods 187
TSocketTransport properties 190
tsOwnerDraw enumeration member 4575
tsPane enumeration member 4650
tspLogOff enumeration member 4650
tspLogOffButtonsHot enumeration member 4650
tspLogOffButtonsNormal enumeration member 4650
tspLogOffButtonsPressed enumeration member 4650
tspMorePrograms enumeration member 4650
tspMoreProgramsArrowHot enumeration member 4650
tspMoreProgramsArrowNormal enumeration member 4650
tspMoreProgramsArrowPressed enumeration member 4650
tspPlacesList enumeration member 4650
tspPlacesListSeparator enumeration member 4650
tspPreview enumeration member 4650
tspProgList enumeration member 4650
tspProgListSeparator enumeration member 4650
tspStartPanelDontCare enumeration member 4650
tspStartPanelRoot enumeration member 4650
tspUserPane enumeration member 4650
tspUserPicture enumeration member 4650
TSQLBlobStream class 1507
about TSQLBlobStream class 1507
Create 1508
Destroy 1508
ReadBlobData 1509
TSQLBlobStream Members 1508
TSQLBlobStream methods 1509
TSQLConnection class 1510
about TSQLConnection class 1510
ActiveStatements 1532
AfterConnect 1550
AfterDisconnect 1551
AutoClone 1532
BeforeConnect 1551
BeforeDisconnect 1552

Check 1514
CheckConnection 1514
CheckDisconnect 1515
CloneConnection 1515
CloseDataSets 1515
Commit 1516
Connected 1533
Connection 1534
ConnectionName 1534
ConnectionOptions 1516
ConnectionRegistryFile 1535
ConnectionState 1535
Create 1513
DataSets 1535
Destroy 1513
DoConnect 1516
DoDisconnect 1517
Driver 1536
DriverName 1536
DriverRegistryFile 1537
Execute 1517
ExecuteDirect 1518
GetConnected 1518
GetDataSet 1519
GetDefaultSchemaName 1519
GetDriverFunc 1537
GetFieldNames 1519
GetFieldNames method 1519
GetIndexNames 1521
GetIndexNames method 1520
GetLoginUsername 1521
GetPackageNames 1522
GetPackageNames method 1521
GetProcedureNames 1523
GetProcedureNames method 1522
GetProcedureParams 1524
GetProcedureParams method 1524
GetSchemaNames 1525
GetSchemaNames method 1525
GetTableNames 1525
GetTableNames method 1525
InTransaction 1537
KeepConnection 1538
LastError 1538
LibraryName 1538
Loaded 1526
LoadParamsFromIniFile 1527
LoadParamsOnConnect 1539
LoadSQLDll 1527
LocaleCode 1540
LoginPrompt 1540
MaxStmtsPerConn 1544
MetaData 1545
MultipleTransactionsSupported 1545
Notification 1528
OnLogin 1552
OpenSchema 1528, 1529
OpenSchema method 1528
Params 1546
ParamsLoaded 1547
QuoteChar 1547
Rollback 1529
SetTraceCallbackEvent 1529
SQLConnection 1547
SQLDllHandle 1547
SQLError 1530
SQLHourGlass 1548
StartTransaction 1530
TableScope 1548
TraceCallbackEvent 1549
TransactionsSupported 1549
TSQLConnection events 1550
TSQLConnection Members 1511
TSQLConnection methods 1513
TSQLConnection properties 1531
VendorLib 1550
TSQLDataLink class 1553
about TSQLDataLink class 1553

ActiveChanged 1554
CheckBrowseMode 1555
Create 1554
GetDetailDataSet 1555
RecordChanged 1555
TSQLDataLink Members 1553
TSQLDataLink methods 1554
TSQLDataSet class 1556
about TSQLDataSet class 1556
CommandText 1561
CommandType 1562
Create 1560
DataSource 1563
ExecSQL 1560
MaxBlobSize 1564
ParamCheck 1564
Params 1565
SortFieldNames 1566
SQLConnection 1566
TSQLDataSet Members 1557
TSQLDataSet methods 1560
TSQLDataSet properties 1561
TSQLMonitor class 1567
about TSQLMonitor class 1567
Active 1573
AutoSave 1573
Create 1568
Destroy 1569
FileName 1574
InvokeCallBack 1569
LoadFromFile 1570
MaxTraceCount 1574
OnLogTrace 1575
OnTrace 1575
SaveToFile 1570
SetActive 1571
SetFileName 1571
SetSQLConnection 1571
SetStreamedActive 1571
SetTraceList 1571
SQLConnection 1576
SwitchConnection 1572
Trace 1572
TraceCount 1576
TraceList 1577
TSQLMonitor Members 1567
TSQLMonitor methods 1569
TSQLMonitor properties 1572
UpdateTraceCallBack 1572
TSQLQuery class 1577
about TSQLQuery class 1577
Create 1581
DataSource 1584
Destroy 1581
ExecSQL 1582
GetMetadata 1584
MaxBlobSize 1585
ParamCheck 1585
Params 1586
PrepareStatement 1583
RowsAffected 1586
SQL 1587
SQLConnection 1588
Text 1588
TSQLQuery Members 1578
TSQLQuery methods 1582
TSQLQuery properties 1583
TSqlServerMethod class 1589
about TSqlServerMethod class 1589
Create 1592
ExecuteMethod 1593
Params 1593
ServerMethodName 1594
SQLConnection 1594
TSqlServerMethod Members 1589
TSqlServerMethod methods 1592
TSqlServerMethod properties 1593
TSQLStoredProc class 1594

about TSQLStoredProc class 1594
Create 1598
ExecProc 1599
MaxBlobSize 1600
NextRecordSet 1599
PackageName 1601
ParamCheck 1601
Params 1602
PrepareStatement 1600
SQLConnection 1603
StoredProcName 1603
TSQLStoredProc Members 1595
TSQLStoredProc methods 1599
TSQLStoredProc properties 1600
TSQLTable class 1604
about TSQLTable class 1604
Active 1611
Create 1608
DeleteRecords 1609
Destroy 1608
GetIndexNames 1609
IndexFieldCount 1613
IndexFieldNames 1613
IndexFields 1614
IndexName 1614
MasterFields 1615
MasterLink 1615
MasterSource 1615
MaxBlobSize 1616
OpenCursor 1610
PrepareStatement 1610
SetIndexField 1610
SQLConnection 1617
TableName 1617
TSQLTable Members 1605
TSQLTable methods 1609
TSQLTable properties 1611
tsScrollBarDontCare enumeration member 4647
tsScrollBarRoot enumeration member 4647
tsSizeBoxLeftAlign enumeration member 4647
tsSizeBoxRightAlign enumeration member 4647
tsSoftTabs enumeration member 4575
tsSpinDontCare enumeration member 4649
tsSpinRoot enumeration member 4649
tsStandard enumeration member 4575
tsStatusDontCare enumeration member 4650
tsStatusRoot enumeration member 4650
tsSynonym enumeration member 1630
tsSysTable enumeration member 1630
tsTable enumeration member 1630
TStandardAddRemoveItem class 1674
about TStandardAddRemoveItem class 1674
DrawBackground 1675
DrawGlyph 1675
TStandardAddRemoveItem Members 1674
TStandardAddRemoveItem methods 1675
TStandardButtonControl class 1676
about TStandardButtonControl class 1676
DrawFrame 1676
DrawGlyph 1677
DrawText 1677
TStandardButtonControl Members 1676
TStandardButtonControl methods 1676
TStandardCustomizePopup class 1677
about TStandardCustomizePopup class 1677
CanAutoSize 1678
NCPaint 1678
PositionPopup 1679
TStandardCustomizePopup Members 1677
TStandardCustomizePopup methods 1678
TStandardDropDownButton class 1679
about TStandardDropDownButton class 1679
DrawFrame 1680
GetPopupClass 1680
TStandardDropDownButton Members 1679
TStandardDropDownButton methods 1679
TStandardMenuButton class 1680
about TStandardMenuButton class 1680

DrawBackground 1681
DrawDesignFocus 1681
DrawFrame 1682
DrawText 1682
TStandardMenuButton Members 1681
TStandardMenuButton methods 1681
TStandardMenuExpandBtn class 1682
about TStandardMenuExpandBtn class 1682
DrawBackground 1683
DrawFrame 1683
TStandardMenuExpandBtn Members 1682
TStandardMenuExpandBtn methods 1683
TStandardMenuItem class 1683
about TStandardMenuItem class 1683
CMTextchanged 1685
DrawBackground 1685
DrawDesignFocus 1685
DrawEdge 1686
DrawGlyph 1686
DrawGlyphFrame 1686
DrawLargeGlyph 1687
DrawSeparator 1687
DrawText 1687
TStandardMenuItem Members 1684
TStandardMenuItem methods 1684
TStandardMenuPopup class 1688
about TStandardMenuPopup class 1688
CanAutoSize 1689
GetExpandBtnClass 1689
NCPaint 1689
PositionPopup 1690
TStandardMenuPopup Members 1688
TStandardMenuPopup methods 1688
TStandardStyleActionBars class 2922
about TStandardStyleActionBars class 2922
GetAddRemoveItemClass 2923
GetColorMapClass 2923
GetControlClass 2923
GetPopupClass 2924
GetScrollBtnClass 2924
GetStyleName 2924
TStandardStyleActionBars Members 2922
TStandardStyleActionBars methods 2922
TStandardToolScrollBtn class 1690
about TStandardToolScrollBtn class 1690
DrawFrame 1691
TStandardToolScrollBtn Members 1690
TStandardToolScrollBtn methods 1690
TStaticText class 2853
about TStaticText class 2853
Align 2855
Alignment 2859
Anchors 2859
AutoSize 2860
BevelEdges 2860
BevelInner 2861
BevelKind 2861
BevelOuter 2862
BiDiMode 2862
BorderStyle 2862
Caption 2863
Color 2865
Constraints 2866
DoubleBuffered 2867
DragCursor 2868
DragKind 2869
DragMode 2869
Enabled 2871
FocusControl 2871
Font 2872
OnClick 2885
OnContextPopup 2887
OnDblClick 2887
OnDragDrop 2888
OnDragOver 2891
OnEndDock 2894
OnEndDrag 2894
OnMouseActivate 2895

OnMouseDown 2896
OnMouseEnter 2899
OnMouseLeave 2899
OnMouseMove 2899
OnMouseUp 2901
OnStartDock 2904
OnStartDrag 2904
ParentBiDiMode 2873
ParentColor 2873
ParentDoubleBuffered 2874
ParentFont 2874
ParentShowHint 2874
PopupMenu 2875
ShowAccelChar 2880
ShowHint 2880
TabOrder 2882
TabStop 2882
Transparent 2883
TStaticText events 2884
TStaticText Members 2853
TStaticText properties 2855
Visible 2883
tsThumbBtnHorzDisabled enumeration member 4647
tsThumbBtnHorzHot enumeration member 4647
tsThumbBtnHorzNormal enumeration member 4647
tsThumbBtnHorzPressed enumeration member 4647
tsThumbBtnVertDisabled enumeration member 4647
tsThumbBtnVertHot enumeration member 4647
tsThumbBtnVertNormal enumeration member 4647
tsThumbBtnVertPressed enumeration member 4647
TStreamedConnection class 191
_AddRef 194
_Release 194
about TStreamedConnection class 191
Create 193
CreateTransport 195
Destroy 193
DoConnect 195
DoDisconnect 196
DoError 196
GetInterceptorList 196
GetInterpreter 197
GetServerList 197
Handle 201
InterceptGUID 201
InterceptName 202
InternalClose 198
InternalOpen 198
Interpreter 203
QueryInterface 199
Send 199
SupportCallbacks 203
ThreadException 199
ThreadReceivedStream 200
TStreamedConnection Members 192
TStreamedConnection methods 193
TStreamedConnection properties 201
WndProc 200
TStringBuilder class 4070
about TStringBuilder class 4070
Append 4078
Append method 4077
AppendFormat 4079
AppendLine 4080
AppendLine method 4079
Capacity 4085
Chars 4085
CopyTo 4080
Create 4075
Create constructor 4074
EnsureCapacity 4080
Equals 4081
FData 4076
FLength 4076
FMaxCapacity 4076
Insert 4082
Insert method 4081
Length 4086

MaxCapacity 4086
Remove 4083
Replace 4084
Replace method 4083
ToString 4084
ToString method 4084
TStringBuilder fields 4075
TStringBuilder Members 4073
TStringBuilder methods 4076
TStringBuilder properties 4085
TStringsValuesList class 5843
about TStringsValuesList class 5843
Actions 5848
Data 5849
OnPrepareStrings 5849
Strings 5849
TStringsValuesList events 5849
TStringsValuesList Members 5844
TStringsValuesList properties 5848
tsUpDisabled enumeration member 4649
tsUpHorzDisabled enumeration member 4649
tsUpHorzHot enumeration member 4649
tsUpHorzNormal enumeration member 4649
tsUpHorzPressed enumeration member 4649
tsUpHot enumeration member 4649
tsUpNormal enumeration member 4649
tsUpperTrackHorzDisabled enumeration member 4647
tsUpperTrackHorzHot enumeration member 4647
tsUpperTrackHorzNormal enumeration member 4647
tsUpperTrackHorzPressed enumeration member 4647
tsUpperTrackVertDisabled enumeration member 4647
tsUpperTrackVertHot enumeration member 4647
tsUpperTrackVertNormal enumeration member 4647
tsUpperTrackVertPressed enumeration member 4647
tsUpPressed enumeration member 4649
tsView enumeration member 1630
TSynchroObject class 3091
about TSynchroObject class 3091
Acquire 3092
Release 3093
TSynchroObject Members 3091
TSynchroObject methods 3092
TTabbedNotebook class 4442
about TTabbedNotebook class 4442
ActivePage 4451
Align 4452
AlignControls 4445
Anchors 4455
Change 4445
Click 4445
CMTabFontChanged 4446
Constraints 4456
Create 4443
CreateHandle 4446
CreateParams 4447
Destroy 4444
DoubleBuffered 4456
Enabled 4458
Font 4458
GetChildOwner 4447
GetChildren 4447
GetIndexForPage 4448
KeyDown 4448
Loaded 4449
OnChange 4473
OnClick 4473
OnContextPopup 4474
OnEnter 4474
OnExit 4477
PageIndex 4459
Pages 4460
ParentDoubleBuffered 4460
ParentShowHint 4461
PopupMenu 4461
ReadState 4449
SetPageIndex 4450
ShowControl 4450
ShowHint 4466

TabFont 4468
TabFontChanged 4450
TabOrder 4468
Tabs 4469
TabsPerRow 4470
TabStop 4471
TopFont 4471
TTabbedNotebook events 4473
TTabbedNotebook Members 4442
TTabbedNotebook methods 4444
TTabbedNotebook properties 4451
Visible 4471
TTabList class 4499
about TTabList class 4499
Add 4500
AddStrings 4503
Clear 4504
Create 4500
Delete 4505
Insert 4505
Put 4506
TTabList Members 4500
TTabList methods 4500
TTabPage class 4480
about TTabPage class 4480
Caption 4482
Create 4481
Enabled 4485
Height 4485
ReadState 4481
TabOrder 4486
TTabPage Members 4480
TTabPage methods 4481
TTabPage properties 4482
Visible 4487
Width 4488
TTabSet class 4506
about TTabSet class 4506
Align 4517
Anchors 4521
AutoScroll 4521
BackgroundColor 4522
CanChange 4510
Canvas 4522
CMHintShow 4510
Constraints 4524
Create 4509
CreateParams 4511
Destroy 4509
DitherBackground 4525
DoubleBuffered 4525
DragCursor 4527
DragKind 4527
DragMode 4528
DrawTab 4511
Enabled 4529
EndMargin 4530
FirstIndex 4530
Font 4530
GetChildren 4511
GetImageIndex 4512
Images 4532
ItemAtPos 4512
ItemRect 4512
ItemWidth 4513
MeasureTab 4513
MinClientRect 4513
MinClientRect method 4513
MouseDown 4514
Notification 4515
OnAlignInsertBefore 4545
OnAlignPosition 4546
OnChange 4547
OnClick 4547
OnDragDrop 4549
OnDragOver 4552
OnDrawTab 4555
OnEndDock 4555

OnEndDrag 4556
OnEnter 4557
OnExit 4559
OnGetImageIndex 4562
OnMeasureTab 4562
OnMouseActivate 4563
OnMouseDown 4564
OnMouseEnter 4566
OnMouseLeave 4566
OnMouseMove 4566
OnMouseUp 4569
OnStartDock 4571
OnStartDrag 4572
Paint 4515
ParentBackground 4532
ParentDoubleBuffered 4532
ParentShowHint 4532
PopupMenu 4533
Scroller 4538
SelectedColor 4538
SelectNext 4516
SetTabIndex 4516
ShowHint 4538
ShrinkToFit 4540
SoftTop 4540
StartMargin 4540
Style 4541
TabHeight 4541
TabIndex 4542
TabPosition 4542
Tabs 4542
TTabSet events 4545
TTabSet Members 4507
TTabSet methods 4510
TTabSet properties 4516
UnselectedColor 4543
Visible 4543
VisibleTabs 4544
ttbButtonChecked enumeration member 4654
ttbButtonCheckedHot enumeration member 4654
ttbButtonDisabled enumeration member 4654
ttbButtonHot enumeration member 4654
ttbButtonNormal enumeration member 4654
ttbButtonPressed enumeration member 4654
ttbDropDownButtonChecked enumeration member 4654
ttbDropDownButtonCheckedHot enumeration member 4654
ttbDropDownButtonDisabled enumeration member 4654
ttbDropDownButtonHot enumeration member 4654
ttbDropDownButtonNormal enumeration member 4654
ttbDropDownButtonPressed enumeration member 4654
ttbFlashButton enumeration member 4653
ttbGroupCount enumeration member 4653
ttBody enumeration member 4651
ttBranch enumeration member 4658
ttbSeparatorChecked enumeration member 4654
ttbSeparatorCheckedHot enumeration member 4654
ttbSeparatorDisabled enumeration member 4654
ttbSeparatorHot enumeration member 4654
ttbSeparatorNormal enumeration member 4654
ttbSeparatorPressed enumeration member 4654
ttbSeparatorVertChecked enumeration member 4654
ttbSeparatorVertCheckedHot enumeration member 4654
ttbSeparatorVertDisabled enumeration member 4654
ttbSeparatorVertHot enumeration member 4654
ttbSeparatorVertNormal enumeration member 4654
ttbSeparatorVertPressed enumeration member 4654
ttbSplitButtonChecked enumeration member 4654
ttbSplitButtonCheckedHot enumeration member 4654
ttbSplitButtonDisabled enumeration member 4654
ttbSplitButtonDropDownChecked enumeration member 4654
ttbSplitButtonDropDownCheckedHot enumeration member 4654
ttbSplitButtonDropDownDisabled enumeration member 4654
ttbSplitButtonDropDownHot enumeration member 4654
ttbSplitButtonDropDownNormal enumeration member 4654
ttbSplitButtonDropDownPressed enumeration member 4654
ttbSplitButtonHot enumeration member 4654
ttbSplitButtonNormal enumeration member 4654
ttbSplitButtonPressed enumeration member 4654

ttbTaskBandDontCare enumeration member 4653
ttbTaskBandRoot enumeration member 4653
ttbThumbBottomDisabled enumeration member 4656
ttbThumbBottomFocused enumeration member 4656
ttbThumbBottomHot enumeration member 4656
ttbThumbBottomNormal enumeration member 4656
ttbThumbBottomPressed enumeration member 4656
ttbThumbDisabled enumeration member 4656
ttbThumbFocused enumeration member 4656
ttbThumbHot enumeration member 4656
ttbThumbLeftDisabled enumeration member 4656
ttbThumbLeftFocused enumeration member 4656
ttbThumbLeftHot enumeration member 4656
ttbThumbLeftNormal enumeration member 4656
ttbThumbLeftPressed enumeration member 4656
ttbThumbNormal enumeration member 4656
ttbThumbPressed enumeration member 4656
ttbThumbRightDisabled enumeration member 4656
ttbThumbRightFocused enumeration member 4656
ttbThumbRightHot enumeration member 4656
ttbThumbRightNormal enumeration member 4656
ttbThumbRightPressed enumeration member 4656
ttbThumbTics enumeration member 4656
ttbThumbTicsVert enumeration member 4656
ttbThumbTopDisabled enumeration member 4656
ttbThumbTopFocused enumeration member 4656
ttbThumbTopHot enumeration member 4656
ttbThumbTopNormal enumeration member 4656
ttbThumbTopPressed enumeration member 4656
ttbThumbVertDisabled enumeration member 4656
ttbThumbVertFocused enumeration member 4656
ttbThumbVertHot enumeration member 4656
ttbThumbVertNormal enumeration member 4656
ttbThumbVertPressed enumeration member 4656
ttbTimeNormal enumeration member 4653
ttbToolBarDontCare enumeration member 4654
ttbToolBarRoot enumeration member 4654
ttbTrack enumeration member 4656
ttbTrackBarDontCare enumeration member 4656
ttbTrackBarRoot enumeration member 4656
ttbTrackVert enumeration member 4656
TTcpClient class 1362
  about TTcpClient class 1362
  Active 1365
  BlockMode 1366
  Connected 1366
  OnConnect 1370
  OnCreateHandle 1370
  OnDestroyHandle 1371
  OnDisconnect 1373
  OnError 1373
  OnReceive 1373
  OnSend 1374
  RemoteHost 1367
  RemotePort 1368
  TTcpClient events 1369
  TTcpClient Members 1363
  TTcpClient properties 1365
TTcpServer class 1375
  about TTcpServer class 1375
  Active 1379
  BlockMode 1380
  LocalHost 1380
  LocalPort 1381
  OnAccept 1382
  OnCreateHandle 1384
  OnDestroyHandle 1385
  OnGetThread 1387
  OnListening 1387
  TTcpServer events 1382
  TTcpServer Members 1377
  TTcpServer properties 1378
ttGlyphClosed enumeration member 4658
ttGlyphOpened enumeration member 4658
TThemedAddRemoveItem class 4594
  about TThemedAddRemoveItem class 4594
  CalcBounds 4595
  DrawBackground 4595

DrawGlyph 4596
DrawSeparator 4596
DrawText 4596
DrawUnusedEdges 4597
GetImageSize 4597
TThemedAddRemoveItem Members 4594
TThemedAddRemoveItem methods 4595
TThemedButtonControl class 4597
about TThemedButtonControl class 4597
DrawBackground 4598
DrawFrame 4598
DrawGlyph 4598
DrawLargeGlyph 4599
DrawText 4599
TThemedButtonControl Members 4597
TThemedButtonControl methods 4598
TThemedCustomizePopup class 4599
about TThemedCustomizePopup class 4599
Create 4600
GetAddRemoveItemClass 4600
GetDefaultColorMapClass 4601
NCPaint 4601
TThemedCustomizePopup Members 4600
TThemedCustomizePopup methods 4600
TThemedDropDownButton class 4601
about TThemedDropDownButton class 4601
DrawBackground 4602
DrawFrame 4602
DrawGlyph 4603
DrawLargeGlyph 4603
GetDropDownButtonWidth 4603
GetPopupClass 4603
TThemedDropDownButton Members 4601
TThemedDropDownButton methods 4602
TThemedExpandBtn class 4604
about TThemedExpandBtn class 4604
DrawBackground 4604
DrawFrame 4605
GetImageSize 4605
TThemedExpandBtn Members 4604
TThemedExpandBtn methods 4604
TThemedMenuButton class 4605
about TThemedMenuButton class 4605
CalcBounds 4606
DrawBackground 4606
DrawText 4607
TThemedMenuButton Members 4605
TThemedMenuButton methods 4606
TThemedMenuItem class 4607
about TThemedMenuItem class 4607
CalcBounds 4608
DrawBackground 4608
DrawGlyph 4609
DrawLargeGlyph 4609
DrawSeparator 4609
DrawShadowedText 4610
DrawSubMenuGlyph 4610
DrawText 4610
DrawUnusedEdges 4610
GetImageSize 4611
TThemedMenuItem Members 4607
TThemedMenuItem methods 4608
TThemedPopupMenu class 4611
about TThemedPopupMenu class 4611
Create 4611
GetExpandBtnClass 4612
NCPaint 4612
PositionPopup 4612
TThemedPopupMenu Members 4611
TThemedPopupMenu methods 4612
VisibleChanging 4613
TThemedToolScrollBtn class 4613
about TThemedToolScrollBtn class 4613
DrawBackground 4614
TThemedToolScrollBtn Members 4613
TThemedToolScrollBtn methods 4614
TThemeServices class 4617
about TThemeServices class 4617

ApplyThemeChange 4620
ColorToRGB 4620, 4621
ColorToRGB method 4620
ContentRect 4621
Create 4618
Destroy 4619
DoOnThemeChange 4621
DrawEdge 4622
DrawEdge method 4621
DrawElement 4622, 4623
DrawElement method 4622
DrawIcon 4623
DrawParentBackground 4624
DrawParentBackground method 4623
DrawText 4625
GetElementDetails 4626, 4627, 4628, 4629, 4630, 4631
GetElementDetails method 4625
HasTransparentParts 4631
OnThemeChange 4633
PaintBorder 4631
Theme 4632
ThemesAvailable 4632
ThemesEnabled 4632
TThemeServices events 4633
TThemeServices Members 4617
TThemeServices methods 4619
TThemeServices properties 4632
UnloadThemeData 4631
UpdateThemes 4632
TThreadLocalCounter class 4087
about TThreadLocalCounter class 4087
Close 4088
Delete 4088
Destroy 4087
Open 4088
TThreadLocalCounter Members 4087
TThreadLocalCounter methods 4087
TTimeZone record 1659
about TTimeZone record 1659
FInfo 1660
FUsesDayligtSavings 1660
GetTimeZone 1660
TTimeZone fields 1659
TTimeZone Members 1659
TTimeZone methods 1660
ttItemDisabled enumeration member 4658
ttItemHot enumeration member 4658
ttItemNormal enumeration member 4658
ttItemSelected enumeration member 4658
ttItemSelectedNotFocus enumeration member 4658
ttnAnimBackground enumeration member 4657
ttnBackground enumeration member 4657
ttnTrayNotifyDontCare enumeration member 4657
ttnTrayNotifyRoot enumeration member 4657
TToolDockForm class 4668
about TToolDockForm class 4668
CanResize 4669
Create 4669
CreateParams 4670
DoAddDockClient 4670
TToolDockForm Members 4668
TToolDockForm methods 4669
WMNCCreate 4670
WMNCHitTest 4670
WMNCLButtonDown 4671
WMSize 4671
WMSysCommand 4671
TToolDockObject class 4671
about TToolDockObject class 4671
AdjustDockRect 4673
Create 4672
DrawDragDockImage 4673
EraseDragDockImage 4673
TToolDockObject Members 4672
TToolDockObject methods 4672
TToolWindow class 4673
about TToolWindow class 4673
CMBorderChanged 4675

CMCtl3DChanged 4675
Create 4674
EdgeBorders 4676
EdgeInner 4676
EdgeOuter 4677
NCPaint 4675
TToolWindow Members 4674
TToolWindow methods 4674
TToolWindow properties 4676
WMNCCalcSize 4675
WMNCPaint 4676
ttPane enumeration member 4651
ttpFlashButtonGroupMenu enumeration member 4653
TTransactionItem class 1618
about TTransactionItem class 1618
FNext 1618
FTransaction 1619
FTransactionDesc 1619
TTransactionItem fields 1618
TTransactionItem Members 1618
TTransportThread class 204
about TTransportThread class 204
Create 204
Destroy 204
Execute 205
Semaphore 209
TTransportThread Members 204
TTransportThread methods 205
TTransportThread properties 209
ttTabDontCare enumeration member 4651
ttTabItemBothEdgeDisabled enumeration member 4651
ttTabItemBothEdgeFocused enumeration member 4651
ttTabItemBothEdgeHot enumeration member 4651
ttTabItemBothEdgeNormal enumeration member 4651
ttTabItemBothEdgeSelected enumeration member 4651
ttTabItemDisabled enumeration member 4651
ttTabItemFocused enumeration member 4651
ttTabItemHot enumeration member 4651
ttTabItemLeftEdgeDisabled enumeration member 4651
ttTabItemLeftEdgeFocused enumeration member 4651
ttTabItemLeftEdgeHot enumeration member 4651
ttTabItemLeftEdgeNormal enumeration member 4651
ttTabItemLeftEdgeSelected enumeration member 4651
ttTabItemNormal enumeration member 4651
ttTabItemRightEdgeDisabled enumeration member 4651
ttTabItemRightEdgeFocused enumeration member 4651
ttTabItemRightEdgeHot enumeration member 4651
ttTabItemRightEdgeNormal enumeration member 4651
ttTabItemRightEdgeSelected enumeration member 4651
ttTabItemSelected enumeration member 4651
ttTabRoot enumeration member 4651
ttTaskBarDontCare enumeration member 4653
ttTaskBarRoot enumeration member 4653
tttBaloonLink enumeration member 4655
tttBaloonNormal enumeration member 4655
tttBaloonStemDownLeft enumeration member 4655
tttBaloonStemUpLeft enumeration member 4655
tttBaloonTitleLink enumeration member 4655
tttBaloonTitleNormal enumeration member 4655
tttCloseHot enumeration member 4655
tttCloseNormal enumeration member 4655
tttClosePressed enumeration member 4655
ttTopTabItemBothEdgeDisabled enumeration member 4651
ttTopTabItemBothEdgeFocused enumeration member 4651
ttTopTabItemBothEdgeHot enumeration member 4651
ttTopTabItemBothEdgeNormal enumeration member 4651
ttTopTabItemBothEdgeSelected enumeration member 4651
ttTopTabItemDisabled enumeration member 4651
ttTopTabItemFocused enumeration member 4651
ttTopTabItemHot enumeration member 4651
ttTopTabItemLeftEdgeDisabled enumeration member 4651
ttTopTabItemLeftEdgeFocused enumeration member 4651
ttTopTabItemLeftEdgeHot enumeration member 4651
ttTopTabItemLeftEdgeNormal enumeration member 4651
ttTopTabItemLeftEdgeSelected enumeration member 4651
ttTopTabItemNormal enumeration member 4651
ttTopTabItemRightEdgeDisabled enumeration member 4651
ttTopTabItemRightEdgeFocused enumeration member 4651

ttTopTabItemRightEdgeHot enumeration member 4651
ttTopTabItemRightEdgeNormal enumeration member 4651
ttTopTabItemRightEdgeSelected enumeration member 4651
ttTopTabItemSelected enumeration member 4651
ttTreeviewDontCare enumeration member 4658
ttTreeviewRoot enumeration member 4658
tttStandardLink enumeration member 4655
tttStandardNormal enumeration member 4655
tttStandardTitleLink enumeration member 4655
tttStandardTitleNormal enumeration member 4655
tttToolTipDontCare enumeration member 4655
tttToolTipRoot enumeration member 4655
TTypes class 7009
about TTypes class 7009
AfterConstruction 7014
GetSchemaDefs 7015
TTypes Members 7009
TTypes methods 7014
TTypeTranslator class 4714
about TTypeTranslator class 4714
Base64ToVar 4716
Base64ToVar method 4716
CastNativeToSoap 4716
CastSoapToNative 4717
CastSoapToVariant 4717
CastSoapToVariant method 4717
Create 4715
Destroy 4715
TTypeTranslator Members 4715
TTypeTranslator methods 4715
TUdpSocket class 1387
about TUdpSocket class 1387
Active 1390
BlockMode 1391
Create 1389
LocalHost 1392
LocalPort 1392
OnConnect 1396
OnCreateHandle 1397
OnDestroyHandle 1398
OnDisconnect 1399
OnError 1400
OnReceive 1400
OnSend 1401
RemoteHost 1394
RemotePort 1395
TUdpSocket events 1396
TUdpSocket Members 1388
TUdpSocket properties 1390
TUnicodeEncoding class 4088
about TUnicodeEncoding class 4088
Create 4090
GetByteCount 4090
GetBytes 4091
GetCharCount 4091
GetChars 4092
GetMaxByteCount 4092
GetMaxCharCount 4092
GetPreamble 4093
TUnicodeEncoding Members 4089
TUnicodeEncoding methods 4090
TUpdateFileList class 5850
about TUpdateFileList class 5850
Count 5851
Create 5850
Files 5851
TUpdateFileList Members 5850
TUpdateFileList properties 5851
TUTF7Encoding class 4093
about TUTF7Encoding class 4093
Create 4094
GetByteCount 4095
GetBytes 4095
GetCharCount 4096
GetChars 4096
GetMaxByteCount 4097
GetMaxCharCount 4097
TUTF7Encoding Members 4093

TUTF7Encoding methods 4095
TUTF8Encoding class 4098
about TUTF8Encoding class 4098
Create 4099
GetMaxByteCount 4099
GetMaxCharCount 4100
GetPreamble 4100
TUTF8Encoding Members 4098
TUTF8Encoding methods 4099
TValueListEditor class 4814
about TValueListEditor class 4814
AdjustColWidths 4819
AdjustRowCount 4819
Align 4831
Anchors 4835
BiDiMode 4835
BorderStyle 4836
CanEditModify 4820
Cells 4836
ColCount 4836
Color 4837
ColWidthsChanged 4820
Constraints 4838
Create 4817
CreateEditor 4820
CreateWnd 4821
Ctl3D 4838
DefaultColWidth 4839
DefaultDrawing 4840
DefaultRowHeight 4841
DeleteRow 4821
Destroy 4818
DisplayOptions 4841
DoExit 4822
DoOnValidate 4822
DoubleBuffered 4842
DragCursor 4843
DragKind 4844
DragMode 4844
DrawCell 4822
DropDownRows 4846
EditList 4846
EditListGetItems 4823
Enabled 4847
FindRow 4823
FixedColor 4847
FixedCols 4847
Font 4848
GetCell 4823
GetColCount 4824
GetEditLimit 4824
GetEditMask 4824
GetEditStyle 4824
GetEditText 4825
GetPickList 4825
GetRowCount 4825
GridLineWidth 4849
InsertRow 4825
IsEmptyRow 4826
ItemProps 4851
KeyDown 4826
KeyOptions 4851
Keys 4851
MouseDown 4827
OnClick 4868
OnContextPopup 4870
OnDblClick 4871
OnDragDrop 4871
OnDragOver 4874
OnDrawCell 4878
OnEditButtonClick 4879
OnEndDock 4880
OnEndDrag 4880
OnEnter 4881
OnExit 4883
OnGetEditMask 4886
OnGetEditText 4887
OnGetPickList 4889

OnKeyDown 4889
OnKeyPress 4893
OnKeyUp 4894
OnMouseActivate 4896
OnMouseDown 4897
OnMouseEnter 4899
OnMouseLeave 4899
OnMouseMove 4899
OnMouseUp 4902
OnMouseWheelDown 4904
OnMouseWheelUp 4904
OnRowMoved 4905
OnSelectCell 4906
OnSetEditText 4908
OnStartDock 4908
OnStartDrag 4909
OnStringsChange 4909
OnStringsChanging 4910
OnTopLeftChanged 4910
OnValidate 4910
Options 4852
ParentBiDiMode 4853
ParentColor 4854
ParentCtl3D 4854
ParentDoubleBuffered 4855
ParentFont 4855
ParentShowHint 4855
PopupMenu 4856
Refresh 4827
Resize 4828
RestoreCurrentRow 4828
RowCount 4861
RowMoved 4828
ScrollBars 4861
SelectCell 4829
SetCell 4829
SetEditText 4829
SetStrings 4829
ShowHint 4862
Strings 4863
StringsChanging 4830
TabOrder 4864
TitleCaptions 4864
TitleCaptionsStored 4830
TValueListEditor events 4868
TValueListEditor Members 4815
TValueListEditor methods 4818
TValueListEditor properties 4830
Values 4865
Visible 4865
VisibleColCount 4867
VisibleRowCount 4867
TValueListStrings class 4911
about TValueListStrings class 4911
Assign 4913
Changed 4916
Changing 4916
Clear 4916
Create 4912
CustomSort 4916
Delete 4917
Destroy 4912
Exchange 4917
FindItemProp 4918
InsertItem 4918
ItemProps 4919
KeyIsValid 4918
Put 4919
TValueListStrings Members 4911
TValueListStrings methods 4913
TValueListStrings properties 4919
UpdateCount 4919
TValuesListNameField class 5852
about TValuesListNameField class 5852
GetDefaultFieldName 5855
ImplGetValue 5855
TValuesListNameField Members 5852
TValuesListNameField methods 5855

TValuesListValueField class 5855
about TValuesListValueField class 5855
GetDefaultFieldName 5859
ImplGetValue 5859
TValuesListValueField Members 5856
TValuesListValueField methods 5858
TVariableLookup class 6682
about TVariableLookup class 6682
Create 6683
FindGlobalVariable 6684
FindGlobalVariableContainer 6684
Ident 6684
LookupObject 6684
Obj 6685
TVariableLookup Members 6682
TVariableLookup methods 6683
TVariableLookup properties 6684
twCaptionActive enumeration member 4658
twCaptionDisabled enumeration member 4658
twCaptionInactive enumeration member 4658
twCaptionSizingTemplate enumeration member 4658
twCloseButtonDisabled enumeration member 4658
twCloseButtonHot enumeration member 4658
twCloseButtonNormal enumeration member 4658
twCloseButtonPushed enumeration member 4658
twDialog enumeration member 4658
TWebAppComponents class 6217
about TWebAppComponents class 6217
AdapterDispatcher 6219
ApplicationAdapter 6220
DispatchActions 6220
EndUserAdapter 6221
LocateFileService 6221
OnAfterDispatch 6223
OnBeforeDispatch 6224
OnException 6224
PageDispatcher 6222
Sessions 6222
TWebAppComponents events 6223
TWebAppComponents Members 6218
TWebAppComponents properties 6219
UserListService 6223
TWebAppDataModule class 6550
about TWebAppDataModule class 6550
AppServices 6552
OnActivate 6553
OnDeactivate 6553
TWebAppDataModule events 6553
TWebAppDataModule Members 6551
TWebAppDataModule properties 6552
TWebAppDataModuleFactory class 6265
about TWebAppDataModuleFactory class 6265
Create 6266
GetIsAppModule 6266
TWebAppDataModuleFactory Members 6265
TWebAppDataModuleFactory methods 6266
TWebApplication class 5912
about TWebApplication class 5912
Create 5913
CreateForm 5914
Destroy 5914
Initialize 5915
Run 5915
Title 5916
TWebApplication Members 5913
TWebApplication methods 5914
TWebApplication properties 5915
TWebAppPageModule class 6554
about TWebAppPageModule class 6554
AppServices 6556
DefaultAction 6557
OnActivate 6558
OnAfterDispatchPage 6558
OnBeforeDispatchPage 6559
OnDeactivate 6559
PageProducer 6557
TWebAppPageModule events 6558
TWebAppPageModule Members 6554

TWebAppPageModule properties 6556
TWebAppPageModuleFactory class 6267
about TWebAppPageModuleFactory class 6267
Create 6268
GetIsAppModule 6269
TWebAppPageModuleFactory Members 6268
TWebAppPageModuleFactory methods 6268
TWebAppPageModuleHelper class 6560
about TWebAppPageModuleHelper class 6560
GetPageProducer 6562
TWebAppPageModuleHelper Members 6560
TWebAppPageModuleHelper methods 6562
TWebBrowser class 462
about TWebBrowser class 462
AddressBar 478
Align 478
Anchors 481
Application 482
Busy 482
ClientToWindow 465
Container 483
ControlInterface 483
CreateControl 466
DefaultInterface 484
Document 484
DragCursor 484
DragMode 485
ExecWB 466
ExecWB method 466
FullName 486
FullScreen 487
Get_Application 467
Get_Container 467
Get_Document 467
Get_Parent 467
GetProperty 468
GoBack 468
GoForward 468
GoHome 469
GoSearch 469
HWnd 487
InitControlData 470
LocationName 487
LocationURL 488
MenuBar 488
Name 488
Navigate 470
Navigate method 470
Navigate2 472
Navigate2 method 472
Offline 489
OnBeforeNavigate2 503
OnClientToHostWindow 503
OnCommandStateChange 503
OnDocumentComplete 504
OnDownloadBegin 504
OnDownloadComplete 505
OnDragDrop 505
OnDragOver 508
OnEndDrag 512
OnEnter 513
OnExit 515
OnFileDownload 518
OnFullScreen 518
OnMenuBar 518
OnNavigateComplete2 519
OnNavigateError 519
OnNewWindow2 519
OnPrintTemplateInstantiation 520
OnPrintTemplateTeardown 520
OnPrivacyImpactedStateChange 520
OnProgressChange 520
OnPropertyChange 521
OnQuit 521
OnSetSecureLockIcon 521
OnStartDrag 522
OnStatusBar 522
OnStatusTextChange 523

OnTheaterMode 523
OnTitleChange 523
OnToolBar 524
OnUpdatePageStatus 524
OnVisible 524
OnWindowClosing 524
OnWindowSetHeight 525
OnWindowSetLeft 525
OnWindowSetResizable 525
OnWindowSetTop 525
OnWindowSetWidth 525
Parent 489
ParentShowHint 490
Path 490
PopupMenu 491
PutProperty 473
QueryStatusWB 474
Quit 474
ReadyState 496
Refresh 474
Refresh2 475
Refresh2 method 475
RegisterAsBrowser 496
RegisterAsDropTarget 496
Resizable 497
ShowBrowserBar 476
ShowBrowserBar method 476
ShowHint 497
Silent 499
StatusBar 499
StatusText 499
Stop 476
TabOrder 499
TabStop 500
TheaterMode 500
ToolBar 501
TopLevelContainer 501
TWebBrowser events 502
TWebBrowser Members 462
TWebBrowser methods 465
TWebBrowser properties 477
type_ 501
Visible 502
TWebBrowser_V1 class 526
about TWebBrowser_V1 class 526
Align 534
Anchors 537
Application 538
Busy 538
Container 538
ControlInterface 538
CreateControl 528
DefaultInterface 538
Document 539
DragCursor 539
DragMode 539
Get_Application 528
Get_Container 528
Get_Document 529
Get_Parent 529
GoBack 529
GoForward 529
GoHome 529
GoSearch 530
InitControlData 530
LocationName 541
LocationURL 541
Navigate 530, 531
Navigate method 530
OnBeforeNavigate 554
OnCommandStateChange 554
OnDownloadBegin 554
OnDownloadComplete 554
OnDragDrop 554
OnDragOver 557
OnEndDrag 561
OnEnter 562
OnExit 564

OnFrameBeforeNavigate 567
OnFrameNavigateComplete 567
OnFrameNewWindow 567
OnNavigateComplete 568
OnNewWindow 568
OnProgressChange 568
OnPropertyChange 568
OnQuit 568
OnStartDrag 569
OnStatusTextChange 569
OnTitleChange 570
OnWindowActivate 570
OnWindowMove 570
OnWindowResize 570
Parent 541
ParentShowHint 543
PopupMenu 543
Refresh 532
Refresh2 532
Refresh2 method 532
ShowHint 548
Stop 533
TabOrder 550
TabStop 551
TopLevelContainer 551
TWebBrowser_V1 events 553
TWebBrowser_V1 Members 526
TWebBrowser_V1 methods 527
TWebBrowser_V1 properties 533
type_ 551
Visible 551
TWebComponentClassDescription class 5996
about TWebComponentClassDescription class 5996
ComponentClass 5998
ComponentClassName 5998
Editor 5998
FrameClass 5999
TWebComponentClassDescription fields 5998
TWebComponentClassDescription Members 5997
TWebComponentList class 5999
_AddRef 6002
_Release 6002
about TWebComponentList class 5999
Add 6002
Changed 6003
Clear 6003
Count 6007
Create 6001
Destroy 6001
GetComponent 6003
GetCount 6004
GetOwner 6004
IndexOf 6004
LockNotify 6005
NewComponent 6005
NotifyLocked 6005
OnChange 6009
ParentComponent 6008
QueryInterface 6005
Remove 6005
SetComponentIndex 6006
TWebComponentList events 6009
TWebComponentList Members 6000
TWebComponentList methods 6001
TWebComponentList properties 6007
UnlockNotify 6006
WebComponentContainer_GetComponent 6007
WebComponentContainer_GetComponentCount 6007
WebComponents 6008
TWebComponentsEditorHelper class 6009
_AddRef 6011
_Release 6011
about TWebComponentsEditorHelper class 6009
CanAddClassHelper 6011
ImplCanAddClassHelper 6011
QueryInterface 6011
TWebComponentsEditorHelper Members 6009
TWebComponentsEditorHelper methods 6010

TWebConnection class 209
about TWebConnection class 209
Agent 217
Create 212
CreateTransport 213
Destroy 212
DoConnect 213
GetInterpreter 214
GetWaitEvent 214
ObjectBroker 217
Password 218
Proxy 218
ProxyByPass 219
Receive 214
Send 215
SetConnected 215
SupportCallbacks 219
Transport_GetConnected 216
Transport_SetConnected 216
TWebConnection Members 210
TWebConnection methods 212
TWebConnection properties 217
URL 220
UserName 220
TWebContainedComponent class 6012
about TWebContainedComponent class 6012
Create 6013
Destroy 6014
GetContainerList 6014
GetParentComponent 6015
HasParent 6015
IsDefaultField 6015
ParentChanged 6016
ReadState 6016
SetParentComponent 6017
SetWebParent 6017
TWebContainedComponent Members 6012
TWebContainedComponent methods 6014
TWebContainedComponent properties 6017
WebContainedComponentHelper 6018
WebParent 6018
TWebContainedComponentHelper class 6018
about TWebContainedComponentHelper class 6018
ContainerList 6025
Create 6020
Destroy 6021
GetContainer 6021
GetIndex 6022
IsDefaultField 6022
OnParentChanged 6025
ParentChanged 6023
SetComponentList 6023
SetContainer 6023
SetIndex 6024
SetWebParent 6024
TWebContainedComponentHelper events 6025
TWebContainedComponentHelper Members 6019
TWebContainedComponentHelper methods 6021
TWebContainedComponentHelper properties 6024
WebParent 6025
TWebContainedContainerComponent class 6026
about TWebContainedContainerComponent class 6026
Create 6028
Destroy 6028
GetChildren 6029
GetContainerList 6029
GetDefaultWebComponents 6030
GetParentComponent 6030
GetWebComponents 6031
HasParent 6031
ImplGetDefaultComponentList 6031
IsDefaultField 6032
ParentChanged 6032
ReadState 6033
SetChildOrder 6033
SetDefaultWebComponents 6033
SetParentComponent 6034
SetWebParent 6034

TWebContainedContainerComponent Members 6027
TWebContainedContainerComponent methods 6028
TWebContainedContainerComponent properties 6035
WebContainedComponentHelper 6035
WebContainerComponentHelper 6035
WebParent 6036
TWebContainerComponent class 6036
about TWebContainerComponent class 6036
Create 6037
Destroy 6038
GetChildren 6038
GetDefaultWebComponents 6039
GetWebComponents 6039
ImplGetDefaultComponentList 6039
SetChildOrder 6040
SetDefaultWebComponents 6040
TWebContainerComponent Members 6037
TWebContainerComponent methods 6038
TWebContainerComponent properties 6041
WebContainerComponentHelper 6041
TWebContainerComponentHelper class 6041
about TWebContainerComponentHelper class 6041
Create 6043
DefaultWebComponents 6047
Destroy 6044
GetChildren 6044
GetComponentList 6045
GetDefaultComponentList 6045
GetDefaultWebComponents 6046
OnGetDefaultComponentList 6048
OnSetDefaultComponentList 6049
SetChildOrder 6046
SetDefaultWebComponents 6046
TWebContainerComponentHelper events 6048
TWebContainerComponentHelper Members 6042
TWebContainerComponentHelper methods 6044
TWebContainerComponentHelper properties 6047
Webcomponents 6048
TWebContentOptions class 6049
about TWebContentOptions class 6049
Create 6050
Flags 6051
TWebContentOptions Members 6050
TWebContentOptions properties 6051
TWebContext class 5950
about TWebContext class 5950
Create 5952
Destroy 5953
FindApplicationModule 5954
FindDefaultPageInfo 5955
FindModuleClass 5955
FindModuleName 5956
FindPage 5956
FindPageInfo 5957
FindVariableContainer 5957
GetAdapterRequest 5958
GetAdapterResponse 5958
GetDispatchedPageName 5958
GetEndUser 5958
GetPageCount 5959
GetPageInfo 5959
GetRequestedPageName 5959
GetSession 5959
GetWebModules 5960
GetWebRequest 5960
GetWebResponse 5960
SetAdapterRequest 5960
SetAdapterResponse 5961
SetDispatchedPageName 5961
SetRequestedPageName 5961
TWebContext Members 5951
TWebContext methods 5954
TWebDataModule class 6562
about TWebDataModule class 6562
OnActivate 6564
OnDeactivate 6564
TWebDataModule events 6564
TWebDataModule Members 6563

TWebDataModuleFactory class 6269
about TWebDataModuleFactory class 6269
Create 6270
TWebDataModuleFactory Members 6270
TWebModuleFactoryList class 6588
about TWebModuleFactoryList class 6588
AddFactory 6590
AppModuleFactory 6591
Create 6589
Destroy 6590
GetItem 6591
ItemCount 6591
Items 6592
TWebModuleFactoryList Members 6588
TWebModuleFactoryList methods 6590
TWebModuleFactoryList properties 6591
TWebModuleList class 6592
about TWebModuleList class 6592
AddModule 6594
AddModuleClass 6595
AddModuleName 6595
AutoCreateModules 6596
AutoDestroyModules 6596
Create 6593
Destroy 6594
Factories 6599
FindModuleClass 6597
FindModuleName 6597
GetFactory 6598
GetFactoryCount 6598
GetItem 6598
GetItemCount 6599
GetOnModuleAdded 6599
SetOnModuleAdded 6599
TWebModuleList Members 6593
TWebModuleList methods 6594
TWebModuleList properties 6599
TWebNotifyContainer class 6565
about TWebNotifyContainer class 6565
Add 6568
Create 6566
Destroy 6566
FList 6567
GetCount 6568
NotifyWebActivate 6568
NotifyWebDeactivate 6568
TWebNotifyContainer fields 6567
TWebNotifyContainer Members 6565
TWebNotifyContainer methods 6567
TWebPageInfo class 6271
about TWebPageInfo class 6271
Create 6272
TWebPageInfo Members 6271
TWebPageModule class 6569
about TWebPageModule class 6569
DefaultAction 6571
OnActivate 6572
OnAfterDispatchPage 6573
OnBeforeDispatchPage 6573
OnDeactivate 6574
PageProducer 6572
TWebPageModule events 6572
TWebPageModule Members 6570
TWebPageModule properties 6571
TWebPageModuleFactory class 6273
about TWebPageModuleFactory class 6273
Create 6274
TWebPageModuleFactory Members 6273
TWebPageModuleHelper class 6574
about TWebPageModuleHelper class 6574
GetPageProducer 6577
TWebPageModuleHelper Members 6574
TWebPageModuleHelper methods 6576
TWebRequestHandler class 6600
about TWebRequestHandler class 6600
ActivateWebModules 6602
ActiveCount 6604
AddWebModuleFactory 6602

CacheConnections 6605
Create 6601
DeactivateWebModules 6603
Destroy 6601
HandleException 6603
HandleRequest 6603
InactiveCount 6605
MaxConnections 6606
TWebRequestHandler Members 6600
TWebRequestHandler methods 6602
TWebRequestHandler properties 6604
WebModuleClass 6606
WebModuleFactory 6607
WebModuleFactoryCount 6607
TWebServExp class 6693
about TWebServExp class 6693
ArrayAsComplexContent 6697
BindingType 6698
Create 6695
Destroy 6695
GenerateXMLSchema 6696
GetWebServExp 6697
GetWSDLForInterface 6697
ImportLocations 6698
ImportNames 6698
OnAfterPublishingWSDL 6699
OnBeforePublishingTypes 6699
OnPublishingType 6699
ServiceAddress 6698
TargetNameSpace 6698
TWebServExp events 6699
TWebServExp Members 6693
TWebServExp methods 6696
TWebServExp properties 6697
WSDLElements 6699
TWebSession class 6225
about TWebSession class 6225
Create 6226
Create constructor 6226
GetSessionID 6227
GetTimeoutMinutes 6227
GetValue 6228
SetTimeoutMinutes 6228
SetValue 6228
Terminate 6229
TWebSession Members 6225
TWebSession methods 6226
UpdateResponse 6229
TWebSessionIDsImpl class 6726
about TWebSessionIDsImpl class 6726
Create 6727
Destroy 6728
GetID 6728
GetIDCount 6729
TWebSessionIDsImpl Members 6726
TWebSessionIDsImpl methods 6728
TWebSessionImpl class 6729
about TWebSessionImpl class 6729
Create 6731
Destroy 6731
ExpirationTime 6736
Expired 6736
GetExpirationTime 6732
GetExpired 6733
GetID 6733
GetIsActive 6733
GetItemCount 6733
GetItems 6734
GetStartTime 6734
GetTerminated 6734
GetTimeoutMinutes 6734
GetTouchTime 6734
GetValue 6735
ID 6736
SetTimeoutMinutes 6735
SetValue 6735
StartTime 6737
Terminate 6735

Terminated 6737
TimeoutMinutes 6737
TouchTime 6737
TWebSessionImpl Members 6730
TWebSessionImpl methods 6732
TWebSessionImpl properties 6736
Values 6738
TWebUserItem class 6750
about TWebUserItem class 6750
AccessRights 6753
AccessRightsStrings 6754
Assign 6752
CheckRights 6753
Create 6751
Destroy 6752
Password 6754
TWebUserItem Members 6751
TWebUserItem methods 6752
TWebUserItem properties 6753
UserID 6754
UserName 6754
TWebUserItems class 6755
about TWebUserItems class 6755
Create 6756
FindUserID 6756
FindUserName 6756
GetAttr 6757
GetAttrCount 6757
GetItemAttr 6758
GetOwner 6758
Items 6759
TWebUserItems Members 6755
TWebUserItems methods 6756
TWebUserItems properties 6759
TWebUserList class 6759
about TWebUserList class 6759
OnAfterCheckAccessRights 6761
OnAfterValidateUser 6761
OnBeforeCheckAccessRights 6762
OnBeforeValidateUser 6762
OnUserIDNotFound 6762
OnValidateUserError 6763
TWebUserList events 6761
TWebUserList Members 6760
TWebUserList properties 6760
UserItems 6761
twFrameBottomActive enumeration member 4658
twFrameBottomInactive enumeration member 4658
twFrameBottomSizingTemplate enumeration member 4658
twFrameLeftActive enumeration member 4658
twFrameLeftInactive enumeration member 4658
twFrameLeftSizingTemplate enumeration member 4658
twFrameRightActive enumeration member 4658
twFrameRightInactive enumeration member 4658
twFrameRightSizingTemplate enumeration member 4658
twHelpButtonDisabled enumeration member 4658
twHelpButtonHot enumeration member 4658
twHelpButtonNormal enumeration member 4658
twHelpButtonPushed enumeration member 4658
twHorzScrollDisabled enumeration member 4658
twHorzScrollHot enumeration member 4658
twHorzScrollNormal enumeration member 4658
twHorzScrollPushed enumeration member 4658
twHorzThumbDisabled enumeration member 4658
twHorzThumbHot enumeration member 4658
twHorzThumbNormal enumeration member 4658
twHorzThumbPushed enumeration member 4658
TWideStringList class 6768
about TWideStringList class 6768
Add 6772
AddObject 6773
CaseSensitive 6782
Changed 6773
Changing 6774
Clear 6774
CompareStrings 6774
Create 6771
Create constructor 6770

CustomSort 6775
Delete 6775
Destroy 6771
Duplicates 6782
Exchange 6776
Find 6776
Get 6777
GetCapacity 6777
GetCount 6777
GetObject 6778
IndexOf 6778
Insert 6779
InsertItem 6779
InsertObject 6779
OnChange 6784
OnChanging 6785
OwnsObjects 6783
Put 6780
PutObject 6780
SetCapacity 6780
SetUpdateState 6781
Sort 6781
Sorted 6783
TWideStringList events 6784
TWideStringList Members 6769
TWideStringList methods 6772
TWideStringList properties 6782
TWideStrings class 6785
about TWideStrings class 6785
Add 6789
AddObject 6790
AddStrings 6790
AddStrings method 6790
Append 6791
Assign 6791
AssignTo 6792
BeginUpdate 6792
Capacity 6807
Clear 6793
CommaText 6807
CompareStrings 6793
Count 6809
DefineProperties 6794
Delete 6794
DelimitedText 6809
Delimiter 6810
Destroy 6788
EndUpdate 6795
Equals 6795
Error 6796
Error method 6795
Exchange 6796
ExtractName 6797
Get 6797
GetCapacity 6797
GetCount 6798
GetEnumerator 6798
GetObject 6798
GetText 6799
GetTextStr 6799
IndexOf 6800
IndexOfName 6800
IndexOfObject 6800
Insert 6801
InsertObject 6801
LineBreak 6810
LoadFromFile 6802
LoadFromStream 6802
Move 6803
Names 6810
NameValueSeparator 6811
Objects 6811
Put 6803
PutObject 6804
QuoteChar 6812
SaveToFile 6804
SaveToStream 6804
SetCapacity 6805

SetText 6805
SetTextStr 6806
SetUpdateState 6806
StrictDelimiter 6812
Strings 6813
StringsAdapter 6814
Text 6814
TWideStrings Members 6786
TWideStrings methods 6788
TWideStrings properties 6806
UpdateCount 6815
ValueFromIndex 6815
Values 6816
TWideStringsEnumerator class 6816
about TWideStringsEnumerator class 6816
Create 6817
Current 6818
GetCurrent 6817
MoveNext 6817
TWideStringsEnumerator Members 6816
TWideStringsEnumerator methods 6817
TWideStringsEnumerator properties 6818
TWindowAction class 1860
about TWindowAction class 1860
Create 1861
Form 1863
GetForm 1862
HandlesTarget 1862
Notification 1862
TWindowAction Members 1861
TWindowAction methods 1861
TWindowAction properties 1863
UpdateTarget 1863
TWindowArrange class 1864
about TWindowArrange class 1864
ExecuteTarget 1865
TWindowArrange Members 1864
TWindowArrange methods 1865
TWindowCascade class 1865
about TWindowCascade class 1865
ExecuteTarget 1866
TWindowCascade Members 1866
TWindowCascade methods 1866
TWindowClose class 1867
about TWindowClose class 1867
ExecuteTarget 1868
TWindowClose Members 1868
TWindowClose methods 1868
UpdateTarget 1869
TWindowMinimizeAll class 1869
about TWindowMinimizeAll class 1869
ExecuteTarget 1870
TWindowMinimizeAll Members 1870
TWindowMinimizeAll methods 1870
TWindowTileHorizontal class 1871
about TWindowTileHorizontal class 1871
ExecuteTarget 1872
TWindowTileHorizontal Members 1871
TWindowTileHorizontal methods 1872
TWindowTileVertical class 1873
about TWindowTileVertical class 1873
ExecuteTarget 1874
TWindowTileVertical Members 1873
TWindowTileVertical methods 1874
TWinHandleUser class 1266
about TWinHandleUser class 1266
TWinHandleUser Members 1266
TWinHandleUser methods 1267
UseHandle1 1267
UseHandle2 1267
TWinSocketStream class 127
about TWinSocketStream class 127
Create 129
Destroy 129
Read 130
Seek 130
TimeOut 132
TWinSocketStream Members 128

TWinSocketStream methods 130
TWinSocketStream properties 132
WaitForData 131
Write 131
twMaxButtonDisabled enumeration member 4658
twMaxButtonHot enumeration member 4658
twMaxButtonNormal enumeration member 4658
twMaxButtonPushed enumeration member 4658
twMaxCaptionActive enumeration member 4658
twMaxCaptionDisabled enumeration member 4658
twMaxCaptionInactive enumeration member 4658
twMDICloseButtonDisabled enumeration member 4658
twMDICloseButtonHot enumeration member 4658
twMDICloseButtonNormal enumeration member 4658
twMDICloseButtonPushed enumeration member 4658
twMDIHelpButtonDisabled enumeration member 4658
twMDIHelpButtonHot enumeration member 4658
twMDIHelpButtonNormal enumeration member 4658
twMDIHelpButtonPushed enumeration member 4658
twMDIMinButtonDisabled enumeration member 4658
twMDIMinButtonHot enumeration member 4658
twMDIMinButtonNormal enumeration member 4658
twMDIMinButtonPushed enumeration member 4658
twMDIRestoreButtonDisabled enumeration member 4658
twMDIRestoreButtonHot enumeration member 4658
twMDIRestoreButtonNormal enumeration member 4658
twMDIRestoreButtonPushed enumeration member 4658
twMDISysButtonDisabled enumeration member 4658
twMDISysButtonHot enumeration member 4658
twMDISysButtonNormal enumeration member 4658
twMDISysButtonPushed enumeration member 4658
twMinButtonDisabled enumeration member 4658
twMinButtonHot enumeration member 4658
twMinButtonNormal enumeration member 4658
twMinButtonPushed enumeration member 4658
twMinCaptionActive enumeration member 4658
twMinCaptionDisabled enumeration member 4658
twMinCaptionInactive enumeration member 4658
twRestoreButtonDisabled enumeration member 4658
twRestoreButtonHot enumeration member 4658
twRestoreButtonNormal enumeration member 4658
twRestoreButtonPushed enumeration member 4658
TWSDLClientNode class 7218
about TWSDLClientNode class 7218
BeforeExecute 7219
Execute 7220, 7221
Execute method 7220
GetMimeBoundary 7221
Notification 7221
SetMimeBoundary 7222
TransportNode 7222
TWSDLClientNode Members 7219
TWSDLClientNode methods 7219
TWSDLClientNode properties 7222
WSDLView 7223
TWSDLDocument class 7015
about TWSDLDocument class 7015
AfterConstruction 7020
Definition 7022
GetDefinition 7021
LoadData 7021
SetDefinition 7021
TWSDLDocument Members 7016
TWSDLDocument methods 7020
TWSDLDocument properties 7021
TWSDLHTMLPublish class 7232
about TWSDLHTMLPublish class 7232
AddInterfaceList 7235
AddPortList 7235
AdminEnabled 7239
Create 7233
Destroy 7234
DispatchEnabled 7235
DispatchMask 7235
DispatchMethodType 7236
DispatchRequest 7236
GetHostScriptBaseURL 7237
GetTargetNamespace 7237

HandleRequest 7237
OnAfterPublishingWSDL 7241
OnBeforePublishingTypes 7241
OnBeforePublishingWSDL 7242
OnPublishingType 7242
PublishOptions 7240
ServiceInfo 7238
TargetNamespace 7240
TWSDLHTMLPublish events 7241
TWSDLHTMLPublish Members 7233
TWSDLHTMLPublish methods 7234
TWSDLHTMLPublish properties 7239
UpdatePortList 7238
WebDispatch 7240
WSILInfo 7238
TWSDLItems class 7163
about TWSDLItems class 7163
CompareName 7171
Create 7169
Create constructor 7169
GetBindingForServicePort 7172
GetBindingOfType 7173
GetBindingOperationNode 7173
GetBindings 7174
GetBindingType 7174
GetHttpBindingAttribute 7175
GetHttpBindingAttribute method 7175
GetImports 7175
GetLocationForImport 7176
GetMessageNode 7176
GetMessages 7177
GetName 7177
GetOperationNode 7177
GetOperations 7178
GetOperationsForBinding 7178
GetPartNode 7179
GetParts 7179
GetPartsForOperation 7180
GetPortsForService 7180
GetPortTypeNode 7181
GetPortTypes 7181
GetSchemaNode 7182
GetSchemas 7182
GetServiceAndPortOfBinding 7183
GetServiceNode 7183
GetServices 7184
GetSoapAction 7184
GetSoapActionList 7185
GetSoapAddressForServicePort 7185
GetSoapBindingAttribute 7186
GetSoapBindingInputNode 7186
GetSoapBindingIONode 7187
GetSoapBindingOutputNode 7188
GetSoapBodyAttribute 7188
GetSoapBodyInputAttribute 7189
GetSoapBodyNamespace 7190
GetSoapBodyOutputAttribute 7190
GetSoapHeaders 7191
GetSoapInputHeaders 7192
GetSoapOperationAttribute 7192
GetSoapOperationStyle 7193
GetSoapOutputHeaders 7193
GetTargetNamespace 7194
GetTargetNamespacePrefix 7195
GetWSDLDoc 7195
GetWSDLItems 7195
HasTypesNode 7196
IsPortTypeHTTPBound 7196
IterateImports 7196
Load 7197
StreamLoader 7197
TargetNamespace 7198
TWSDLItems Members 7164
TWSDLItems methods 7170
TWSDLItems properties 7197
TWSDLPublish class 7242
about TWSDLPublish class 7242
GetPortTypeEntries 7243

GetPortTypeList 7244
GetTypeSystemsList 7244
GetWSDLForPortType 7244
GetXSDForTypeSystem 7244
OnAfterPublishingWSDL 7245
OnBeforePublishingTypes 7245
OnBeforePublishingWSDL 7246
OnPublishingType 7246
TargetNamespace 7245
TWSDLPublish events 7245
TWSDLPublish Members 7243
TWSDLPublish methods 7243
TWSDLPublish properties 7244
TWSDLView class 7223
about TWSDLView class 7223
Activate 7225
IntfInfo 7224
Operation 7225
Password 7226
Port 7226
PortType 7226
Proxy 7227
Service 7227
SetDesignState 7225
TWSDLView fields 7224
TWSDLView Members 7223
TWSDLView methods 7224
TWSDLView properties 7225
UserName 7227
WSDL 7228
twSmallCaptionActive enumeration member 4658
twSmallCaptionDisabled enumeration member 4658
twSmallCaptionInactive enumeration member 4658
twSmallCaptionSizingTemplate enumeration member 4658
twSmallCloseButtonDisabled enumeration member 4658
twSmallCloseButtonHot enumeration member 4658
twSmallCloseButtonNormal enumeration member 4658
twSmallCloseButtonPushed enumeration member 4658
twSmallFrameBottomActive enumeration member 4658
twSmallFrameBottomInactive enumeration member 4658
twSmallFrameBottomSizingTemplate enumeration member 4658
twSmallFrameLeftActive enumeration member 4658
twSmallFrameLeftInactive enumeration member 4658
twSmallFrameLeftSizingTemplate enumeration member 4658
twSmallFrameRightActive enumeration member 4658
twSmallFrameRightInactive enumeration member 4658
twSmallFrameRightSizingTemplate enumeration member 4658
twSmallMaxCaptionActive enumeration member 4658
twSmallMaxCaptionDisabled enumeration member 4658
twSmallMaxCaptionInactive enumeration member 4658
twSmallMinCaptionActive enumeration member 4658
twSmallMinCaptionDisabled enumeration member 4658
twSmallMinCaptionInactive enumeration member 4658
twSysButtonDisabled enumeration member 4658
twSysButtonHot enumeration member 4658
twSysButtonNormal enumeration member 4658
twSysButtonPushed enumeration member 4658
twVertScrollDisabled enumeration member 4658
twVertScrollHot enumeration member 4658
twVertScrollNormal enumeration member 4658
twVertScrollPushed enumeration member 4658
twVertThumbDisabled enumeration member 4658
twVertThumbHot enumeration member 4658
twVertThumbNormal enumeration member 4658
twVertThumbPushed enumeration member 4658
twWindowDontCare enumeration member 4658
twWindowRoot enumeration member 4658
TXMLBroker class 7255
about TXMLBroker class 7255
AddNotify 7259
AfterDispatch 7281
ApplyXMLUpdates 7260
AppServer 7274
AS_ApplyUpdates 7260
AS_FetchParams 7261
AS_GetRecords 7261
BeforeDispatch 7282
CheckInactive 7261

Connected 7274
Create 7257
Destroy 7258
DispatchEnabled 7261
DispatchMask 7261
DispatchMethodType 7262
DispatchRequest 7262
FetchParams 7262
GetAppServer 7262
GetConnected 7263
GetDelta 7263
GetErrorCount 7263
GetErrorResponse 7263
GetErrors 7264
GetHasAppServer 7264
GetNotify 7264
GetNotifyCount 7265
GetProducerName 7265
GetRedirect 7265
GetResponse 7265
GetXMLRecords 7265
GetXMLStream 7266
HasAppServer 7275
HTMLSubmitFormName 7266
MasterRowSetVarName 7267
MaxErrors 7275
MaxRecords 7276
Notification 7267
Notify 7276
NotifyCount 7277
OnGetErrorResponse 7282
OnGetResponse 7283
OnRequestRecords 7284
OnRequestUpdate 7284
Params 7277
ProviderName 7278
ReconcileProducer 7279
RemoteServer 7280
RemoveNotify 7268
RequestRecords 7268
RequestUpdate 7269
RowSetVarName 7270
SendConnectEvent 7271
SetAppServer 7271
SetConnected 7271
SetParams 7271
SetProvider 7272
SetProviderName 7272
SetReconcileProducer 7272
SetRemoteServer 7272
SetWebDispatch 7273
SubmitFormVarName 7273
TXMLBroker events 7281
TXMLBroker Members 7256
TXMLBroker methods 7258
TXMLBroker properties 7273
WebDispatch 7280
TXMLData class 7741
about TXMLData class 7741
Create 7742
Destroy 7742
LoadFromXML 7743
LoadFromXML method 7743
ObjectToSOAP 7743
SOAPToObject 7744
TXMLData Members 7741
TXMLData methods 7742
TXMLData properties 7745
XMLNode 7745
TXMLDocument class 7299
_AddRef 7306
_Release 7306
about TXMLDocument class 7299
Active 7333
AddChild 7307
AddChild method 7307
AfterClose 7344
AfterConstruction 7307

AfterNodeChange 7345
AfterOpen 7345
AssignParseOptions 7308
AsyncLoadState 7334
BeforeClose 7346
BeforeNodeChange 7346
BeforeOpen 7347
CheckActive 7308
CheckAutoSave 7308
CheckDOM 7309
ChildNodes 7335
Create 7303
Create constructor 7303
CreateElement 7309
CreateNode 7309
DefineProperties 7311
Destroy 7304
DoAfterClose 7311
DoAfterOpen 7311
DoBeforeClose 7312
DoBeforeOpen 7312
DocBindingInfo 7335
DocSource 7335
DocumentElement 7335
DOMDocument 7336
DOMImplementation 7337
DOMParseOptions 7337
DOMPersist 7337
DOMVendor 7337
DoNodeChange 7312
Encoding 7338
FileName 7339
GeneratePrefix 7312
GetActive 7313
GetAsyncLoadState 7313
GetChildNodeClass 7313
GetChildNodes 7313
GetDocBinding 7314
GetDocumentElement 7314
GetDocumentNode 7315
GetDocumentObject 7315
GetDOMDocument 7315
GetDOMPersist 7315
GetEncoding 7316
GetFileName 7316
GetModified 7316
GetNodeIndentStr 7317
GetOptions 7317
GetParseOptions 7317
GetPrologNode 7317
GetPrologValue 7318
GetSchemaRef 7318
GetStandAlone 7318
GetVersion 7318
GetXML 7319
InternalSetPrologValue 7319
IsEmptyDoc 7319
LoadData 7320
Loaded 7320
LoadFromFile 7321
LoadFromStream 7321
LoadFromXML 7322
LoadFromXML method 7322
Modified 7339
NewInstance 7322
Node 7340
NodeIndentStr 7340
NSPrefixBase 7341
OnAsyncLoad 7347
Options 7341
ParseOptions 7342
PrefixID 7342
ReadDOMVendor 7323
Refresh 7323
RegisterDocBinding 7324
ReleaseDoc 7324
Resync 7325
SaveToFile 7325

SaveToStream 7325
SaveToUTF8String 7326
SaveToXML 7326
SaveToXML method 7326
SaveToXMLStrings 7327
SchemaRef 7343
SetActive 7327
SetDocumentElement 7327
SetDOMDocument 7328
SetEncoding 7328
SetFileName 7328
SetModified 7329
SetNodeIndentStr 7329
SetOnAsyncLoad 7329
SetOptions 7330
SetParseOptions 7330
SetPrologValue 7330
SetStandAlone 7330
SetVersion 7331
SetXML 7331
SetXMLStrings 7331
StandAlone 7343
TXMLDocument events 7344
TXMLDocument Members 7300
TXMLDocument methods 7304
TXMLDocument properties 7332
Version 7343
WriteDOMVendor 7332
XML 7344
XMLStringsChanging 7332
TXMLNode class 7348
_AddRef 7355
_Release 7356
about TXMLNode class 7348
AddChild 7356
AddChild method 7356
AddHostedNode 7357
AttributeListNotify 7358
AttributeNodes 7390
CheckNotHosted 7358
CheckReadOnly 7359
CheckTextNode 7359
ChildListNotify 7360
ChildNodeClasses 7390
ChildNodes 7391
ClearDocumentRef 7360
CloneNode 7361
Collection 7391
Create 7352
CreateAttributeList 7361
CreateAttributeNode 7362
CreateChildList 7362
CreateChildNode 7362
CreateCollection 7363
CreateHosted 7353
DeclareNamespace 7364
Destroy 7353
DOMElement 7364
DOMNode 7392
DoNodeChange 7365
FindHostedNode 7365
FindNamespaceDecl 7366
FindNamespaceURI 7366
GetAttribute 7367
GetAttributeNodes 7367
GetAttributeNS 7368
GetChildNodeClasses 7368
GetChildNodes 7369
GetChildValue 7369
GetCollection 7370
GetDOMNode 7370
GetHasChildNodes 7370
GetHostedNodes 7371
GetHostNode 7371
GetIsTextElement 7371
GetLocalName 7372
GetNamespaceURI 7373
GetNodeName 7373

GetNodeObject 7374
GetNodeType 7374
GetNodeValue 7375
GetOwnerDocument 7375
GetParentNode 7376
GetPrefix 7376
GetPrefixedName 7377
GetReadOnly 7377
GetText 7378
GetXML 7378
HasAttribute 7379
HasAttribute method 7378
HasChildNode 7379
HasChildNode method 7379
HostedNodes 7393
HostNode 7393
InternalAddChild 7380
NestingLevel 7380
NextSibling 7381
Normalize 7381
OnHostAttrNotify 7395
OnHostChildNotify 7396
OwnerDocument 7394
ParentNode 7395
PreviousSibling 7382
RegisterChildNode 7382
RegisterChildNodes 7383
RemoveHostedNode 7383
Resync 7384
SetAttribute 7384
SetAttributeNodes 7384
SetAttributeNS 7385
SetChildNodes 7385
SetChildValue 7386
SetCollection 7386
SetNodeValue 7387
SetParentNode 7387
SetReadOnly 7388
SetText 7388
TransformNode 7389
TransformNode method 7389
TXMLNode events 7395
TXMLNode Members 7349
TXMLNode methods 7354
TXMLNode properties 7389
TXMLNodeCollection class 7396
about TXMLNodeCollection class 7396
AddItem 7403
AfterConstruction 7403
ChildListNotify 7404
Clear 7404
Count 7409
CreateItemList 7405
Delete 7405
GetCount 7406
GetList 7406
GetNode 7406
InsertInCollection 7407
IsCollectionItem 7407
ItemInterface 7410
ItemNS 7410
ItemTag 7411
List 7411
Remove 7408
SetChildNodes 7408
TXMLNodeCollection Members 7397
TXMLNodeCollection methods 7402
TXMLNodeCollection properties 7409
UpdateCollectionList 7409
TXMLNodeList class 7411
about TXMLNodeList class 7411
Add 7416
BeginUpdate 7416
Clear 7417
Count 7426
Create 7414
DefaultNamespaceURI 7426
Delete 7417

Delete method 7417
Destroy 7415
DoNotify 7418
EndUpdate 7419
FindNode 7419
FindNode method 7419
FindSibling 7420
First 7420
Get 7421
GetCount 7421
GetNode 7422
GetUpdateCount 7422
IndexOf 7423
IndexOf method 7422
Insert 7423
InternalInsert 7424
Last 7424
List 7426
NotificationProc 7428
Owner 7427
Remove 7424
ReplaceNode 7425
TXMLNodeList events 7428
TXMLNodeList Members 7412
TXMLNodeList methods 7415
TXMLNodeList properties 7425
UpdateCount 7427
TXMLParams class 7285
about TXMLParams class 7285
AssignStrings 7287
AssignTo 7287
TXMLParams Members 7286
TXMLParams methods 7286
TXMLSchemaDefs class 7022
about TXMLSchemaDefs class 7022
Add 7028
GetXMLSchemaDef 7028
TXMLSchemaDefs Members 7023
TXMLSchemaDefs methods 7028
TXMLTransform class 7671
about TXMLTransform class 7671
AfterEachRow 7681
AfterEachRowSet 7681
BeforeEachRow 7682
BeforeEachRowSet 7682
Data 7675
DoTransform 7673
EmptyDestinationDocument 7676
GetData 7673
GetResultString 7673
OnTranslate 7683
ResultDocument 7677
ResultString 7677
SourceXml 7678
SourceXmlDocument 7678
SourceXmlFile 7679
Transform 7673
TransformationDocument 7679
TransformationFile 7680
TransformXML 7674
Translate 7674
TXMLTransform events 7681
TXMLTransform Members 7671
TXMLTransform methods 7672
TXMLTransform properties 7675
TXMLTransformClient class 7683
about TXMLTransformClient class 7683
ApplyUpdates 7686
Create 7685
Destroy 7685
GetDataAsXml 7686
ProviderName 7689
RemoteServer 7689
SetParams 7687
SetupAppServer 7688
TransformApplyUpdates 7690
TransformGetData 7690
TransformSetParams 7691

TXMLTransformClient Members 7684
TXMLTransformClient methods 7685
TXMLTransformClient properties 7688
TXMLTransformProvider class 7691
about TXMLTransformProvider class 7691
AfterApplyUpdates 7698
AfterGetRecords 7699
AfterRowRequest 7700
BeforeApplyUpdates 7701
BeforeGetRecords 7701
BeforeRowRequest 7703
CacheData 7696
Create 7693
Destroy 7694
InternalApplyUpdates 7694
InternalGetRecords 7695
OnDataRequest 7703
TransformRead 7696
TransformWrite 7697
TXMLTransformProvider events 7698
TXMLTransformProvider Members 7693
TXMLTransformProvider methods 7694
TXMLTransformProvider properties 7695
XMLDataFile 7697
TXPManifest class 7730
about TXPManifest class 7730
TXPStyleActionBars class 7732
about TXPStyleActionBars class 7732
GetAddRemoveItemClass 7733
GetColorMapClass 7733
GetControlClass 7733
GetPopupClass 7734
GetScrollBtnClass 7734
GetStyleName 7734
TXPStyleActionBars Members 7732
TXPStyleActionBars methods 7732
TXPStyleAddRemoveItem class 7708
about TXPStyleAddRemoveItem class 7708
CalcBounds 7709
DrawBackground 7709
DrawGlyph 7710
DrawSeparator 7710
DrawText 7710
DrawUnusedEdges 7710
TXPStyleAddRemoveItem Members 7708
TXPStyleAddRemoveItem methods 7709
TXPStyleButton class 7711
about TXPStyleButton class 7711
DrawBackground 7711
DrawFrame 7712
DrawGlyph 7712
DrawText 7712
TXPStyleButton Members 7711
TXPStyleButton methods 7711
TXPStyleCustomizePopup class 7713
about TXPStyleCustomizePopup class 7713
Create 7713
GetAddRemoveItemClass 7714
GetDefaultColorMapClass 7714
NCPaint 7714
TXPStyleCustomizePopup Members 7713
TXPStyleCustomizePopup methods 7714
TXPStyleDropDownBtn class 7714
about TXPStyleDropDownBtn class 7714
DrawBackground 7715
DrawFrame 7716
DrawGlyph 7716
GetPopupClass 7716
TXPStyleDropDownBtn Members 7715
TXPStyleDropDownBtn methods 7715
TXPStyleExpandBtn class 7716
about TXPStyleExpandBtn class 7716
DrawBackground 7717
DrawFrame 7717
TXPStyleExpandBtn Members 7717
TXPStyleExpandBtn methods 7717
TXPStyleMenuButton class 7718
about TXPStyleMenuButton class 7718

DrawBackground 7718
DrawText 7719
TXPStyleMenuButton Members 7718
TXPStyleMenuButton methods 7718
TXPStyleMenuItem class 7719
about TXPStyleMenuItem class 7719
CalcBounds 7720
CalcLayout 7720
DrawBackground 7721
DrawGlyph 7721
DrawSeparator 7721
DrawShadowedText 7721
DrawSubMenuGlyph 7722
DrawText 7722
DrawUnusedEdges 7722
TXPStyleMenuItem Members 7719
TXPStyleMenuItem methods 7720
TXPStylePopupMenu class 7722
about TXPStylePopupMenu class 7722
Create 7723
Destroy 7723
DisplayShadow 7724
GetExpandBtnClass 7724
HideShadow 7725
NCPaint 7725
PositionPopup 7725
TXPStylePopupMenu Members 7723
TXPStylePopupMenu methods 7724
VisibleChanging 7725
TXPStyleToolScrollBtn class 7726
about TXPStyleToolScrollBtn class 7726
DrawBackground 7726
TXPStyleToolScrollBtn Members 7726
TXPStyleToolScrollBtn methods 7726
TXSBoolean class 7745
about TXSBoolean class 7745
AsBoolean 7747
NativeToXS 7746
TXSBoolean Members 7745
TXSBoolean methods 7746
TXSBoolean properties 7747
XSToNative 7746
TXSCustomDateTime class 7747
about TXSCustomDateTime class 7747
AsDateTime 7759
AsUTCDateTime 7760
Create 7749
Day 7760
Destroy 7749
FDateTime 7750
FractionalSeconds 7761
GetAsDateTime 7751
GetAsUTCDateTime 7751
GetDay 7751
GetFractionalSeconds 7752
GetHour 7752
GetHourOffset 7752
GetMillisecond 7752
GetMinute 7753
GetMinuteOffset 7753
GetMonth 7753
GetSecond 7754
GetUseZeroMilliseconds 7754
GetYear 7754
Hour 7761
HourOffset 7762
Millisecond 7762
Minute 7763
MinuteOffset 7763
Month 7764
Second 7764
SetAsDateTime 7755
SetAsUTCDateTime 7755
SetDay 7755
SetFractionalSeconds 7755
SetHour 7756
SetHourOffset 7756
SetMillisecond 7756

SetMinute 7757
SetMinuteOffset 7757
SetMonth 7757
SetSecond 7758
SetUseZeroMilliseconds 7758
SetYear 7758
TXSCustomDateTime fields 7750
TXSCustomDateTime Members 7748
TXSCustomDateTime methods 7750
TXSCustomDateTime properties 7759
UseZeroMilliseconds 7765
Year 7765
TXSDate class 7766
about TXSDate class 7766
AsDate 7771
Clone 7768
Create 7767
Day 7771
GetAsDate 7768
GetDay 7768
GetMonth 7768
GetYear 7769
Month 7771
NativeToXS 7769
SetAsDate 7769
SetDay 7769
SetMonth 7770
SetYear 7770
TXSDate Members 7766
TXSDate methods 7767
TXSDate properties 7770
XSToNative 7770
Year 7772
TXSDateTime class 7772
about TXSDateTime class 7772
Clone 7774
CompareDateTimeParam 7775
Millisecond 7776
NativeToXS 7775
TXSDateTime Members 7773
TXSDateTime methods 7774
TXSDateTime properties 7776
XSToNative 7775
TXSDecimal class 7776
about TXSDecimal class 7776
AsBcd 7779
DecimalString 7779
NativeToXS 7778
TXSDecimal Members 7777
TXSDecimal methods 7777
TXSDecimal properties 7778
XSToNative 7778
TXSDuration class 7780
about TXSDuration class 7780
Create 7782
DecimalSecond 7784
GetDecimalSecond 7782
NativeToXS 7783
SetDecimalSecond 7783
TXSDuration Members 7780
TXSDuration methods 7782
TXSDuration properties 7783
XSToNative 7783
TXSHexBinary class 7784
about TXSHexBinary class 7784
AsByteArray 7786
HexBinaryString 7787
NativeToXS 7785
TXSHexBinary Members 7785
TXSHexBinary methods 7785
TXSHexBinary properties 7786
XSToNative 7786
TXSInteger class 7787
about TXSInteger class 7787
AsInteger 7789
NativeToXS 7788
TXSInteger Members 7787
TXSInteger methods 7787

TXSInteger properties 7788
XSToNative 7788
TXSLong class 7789
about TXSLong class 7789
AsLong 7791
NativeToXS 7790
TXSLong Members 7789
TXSLong methods 7789
TXSLong properties 7790
XSToNative 7790
TXSLPageProducer class 7837
about TXSLPageProducer class 7837
TXSLPageProducer Members 7838
TXSLPageProducer properties 7843
XMLData 7843
TXSString class 7791
about TXSString class 7791
NativeToXS 7792
TXSString Members 7791
TXSString methods 7791
XSToNative 7792
TXSTime class 7792
about TXSTime class 7792
AsTime 7800
Clone 7795
FractionalSeconds 7801
FractionalSecondString 7801
GetAsTime 7795
GetFractionalSeconds 7795
GetFractionalSecondString 7796
GetHour 7796
GetHourOffset 7796
GetMillisecond 7796
GetMinute 7796
GetMinuteOffset 7797
GetSecond 7797
Hour 7801
HourOffset 7802
Millisecond 7802
Minute 7802
MinuteOffset 7803
NativeToXS 7797
Second 7803
SetAsTime 7797
SetFractionalSeconds 7798
SetHour 7798
SetHourOffset 7798
SetMillisecond 7798
SetMinute 7799
SetMinuteOffset 7799
SetSecond 7799
TXSTime Members 7793
TXSTime methods 7794
TXSTime properties 7800
UseZeroMilliseconds 7804
XSToNative 7799
Types 4683
Types namespace 4683
Constants 4709
Functions 4685
Structs, Records, Enums 4693
Types 4697
Types.Bounds 4685
Types.Bounds function 4685
Types.CenteredRect 4686
Types.CenteredRect function 4686
Types.CenterPoint 4686
Types.CenterPoint function 4686
Types.DWORD 4697
Types.DWORD type 4697
Types.EqualRect 4686
Types.EqualRect function 4686
Types.EqualsValue 4709
Types.EqualsValue constant 4709
Types.GreaterThanValue 4709
Types.GreaterThanValue constant 4709
Types.IntersectRect 4687
Types.IntersectRect function 4687

Types.IsRectEmpty 4687
Types.IsRectEmpty function 4687
Types.LessThanValue 4710
Types.LessThanValue constant 4710
Types.OffsetRect 4688
Types.OffsetRect function 4688
Types.PByte 4698
Types.PByte type 4698
Types.PDouble 4698
Types.PDouble type 4698
Types.PInteger 4699
Types.PInteger type 4699
Types.PLongint 4699
Types.PLongint type 4699
Types.Point 4688
Types.Point function 4688
Types.PPoint 4699
Types.PPoint type 4699
Types.PRect 4700
Types.PRect type 4700
Types.PSize 4700
Types.PSize type 4700
Types.PSmallInt 4701
Types.PSmallInt type 4701
Types.PSmallPoint 4701
Types.PSmallPoint type 4701
Types.PtInRect 4689
Types.PtInRect function 4689
Types.Rect 4689
Types.Rect function 4689
Types.RectHeight 4690
Types.RectHeight function 4690
Types.RectWidth 4690
Types.RectWidth function 4690
Types.RT_RCDATA 4710
Types.RT_RCDATA constant 4710
Types.SIZE 4701
Types.SIZE type 4701
Types.SmallPoint 4690
Types.SmallPoint function 4690
Types.SplitRect 4691
Types.SplitRect function 4691
Types.tagPOINT 4702
Types.tagPOINT type 4702
Types.tagSIZE 4693
Types.tagSIZE record 4693
Types.TBooleanDynArray 4702
Types.TBooleanDynArray type 4702
Types.TByteDynArray 4702
Types.TByteDynArray type 4702
Types.TCardinalDynArray 4703
Types.TCardinalDynArray type 4703
Types.TDoubleDynArray 4703
Types.TDoubleDynArray type 4703
Types.TInt64DynArray 4704
Types.TInt64DynArray type 4704
Types.TIntegerDynArray 4704
Types.TIntegerDynArray type 4704
Types.TLongWordDynArray 4704
Types.TLongWordDynArray type 4704
Types.TPoint 4694
Types.TPoint record 4694
Types.TRect 4694
Types.TRect record 4694
Types.TShortIntDynArray 4705
Types.TShortIntDynArray type 4705
Types.TSingleDynArray 4705
Types.TSingleDynArray type 4705
Types.TSize 4706
Types.TSize type 4706
Types.TSmallIntDynArray 4706
Types.TSmallIntDynArray type 4706
Types.TSmallPoint 4695
Types.TSmallPoint record 4695
Types.TSplitRectType 4696
Types.TSplitRectType enumeration 4696
Types.TStringDynArray 4706
Types.TStringDynArray type 4706

Types.TValueRelationship 4707
Types.TValueRelationship type 4707
Types.TWideStringDynArray 4707
Types.TWideStringDynArray type 4707
Types.TWordDynArray 4708
Types.TWordDynArray type 4708
Types.UnionRect 4692
Types.UnionRect function 4692
TypeTrans 4713
TypeTrans namespace 4713
Classes 4714
Functions 4718
Variables 4721
TypeTrans.ETypeTransException 4714
TypeTrans.FloatToStrEx 4718
TypeTrans.FloatToStrEx function 4718
TypeTrans.GetEnumValueEx 4718
TypeTrans.GetEnumValueEx function 4718
TypeTrans.GetEnumValueExW 4719
TypeTrans.GetEnumValueExW function 4719
TypeTrans.SetEnumPropEx 4719
TypeTrans.SetEnumPropEx function 4719
TypeTrans.SetEnumPropExW 4719
TypeTrans.SetEnumPropExW function 4719
TypeTrans.StrToFloatEx 4720
TypeTrans.StrToFloatEx function 4720
TypeTrans.TTypeTranslator 4714
TypeTrans.TTypeTranslator.Base64ToVar 4716
TypeTrans.TTypeTranslator.CastNativeToSoap 4716
TypeTrans.TTypeTranslator.CastSoapToNative 4717
TypeTrans.TTypeTranslator.CastSoapToVariant 4717
TypeTrans.TTypeTranslator.Create 4715
TypeTrans.TTypeTranslator.Destroy 4715
TypeTrans.TypeTranslator 4721
TypeTrans.TypeTranslator variable 4721
TypInfo 4723
TypInfo namespace 4723
Classes 4727
Constants 4802
Functions 4735
Structs, Records, Enums 4779
Types 4789
Variables 4801
TypInfo.BooleanIdents 4801
TypInfo.BooleanIdents variable 4801
TypInfo.DotSep 4801
TypInfo.DotSep variable 4801
TypInfo.EPropertyConvertError 4727
TypInfo.EPropertyError 4727
TypInfo.FreeAndNilProperties 4737
TypInfo.FreeAndNilProperties function 4737
TypInfo.GetAnsiStrProp 4737, 4738
TypInfo.GetAnsiStrProp function 4737, 4738
TypInfo.GetDynArrayProp 4738
TypInfo.GetDynArrayProp function 4738
TypInfo.GetEnumName 4739
TypInfo.GetEnumName function 4739
TypInfo.GetEnumProp 4742
TypInfo.GetEnumProp function 4742
TypInfo.GetEnumValue 4744
TypInfo.GetEnumValue function 4744
TypInfo.GetFloatProp 4745
TypInfo.GetFloatProp function 4745
TypInfo.GetInt64Prop 4746
TypInfo.GetInt64Prop function 4746
TypInfo.GetInterfaceProp 4746
TypInfo.GetInterfaceProp function 4746
TypInfo.GetMethodProp 4747
TypInfo.GetMethodProp function 4747
TypInfo.GetObjectProp 4748
TypInfo.GetObjectProp function 4748
TypInfo.GetObjectPropClass 4749
TypInfo.GetObjectPropClass function 4749
TypInfo.GetOrdProp 4750
TypInfo.GetOrdProp function 4750
TypInfo.GetPropInfo 4751
TypInfo.GetPropInfo function 4751
TypInfo.GetPropInfos 4751

TypInfo.GetPropInfos function 4751
TypInfo.GetPropList 4752
TypInfo.GetPropList function 4752
TypInfo.GetPropName 4753
TypInfo.GetPropName function 4753
TypInfo.GetPropValue 4753
TypInfo.GetPropValue function 4753
TypInfo.GetSetElementName 4754
TypInfo.GetSetElementName function 4754
TypInfo.GetSetElementValue 4754
TypInfo.GetSetElementValue function 4754
TypInfo.GetSetProp 4755
TypInfo.GetSetProp function 4755
TypInfo.GetStrProp 4756
TypInfo.GetStrProp function 4756
TypInfo.GetTypeData 4756
TypInfo.GetTypeData function 4756
TypInfo.GetTypeName 4757
TypInfo.GetTypeName function 4757
TypInfo.GetUnicodeStrProp 4757, 4758
TypInfo.GetUnicodeStrProp function 4757, 4758
TypInfo.GetVariantProp 4758
TypInfo.GetVariantProp function 4758
TypInfo.GetWideStrProp 4759
TypInfo.GetWideStrProp function 4759
TypInfo.IsPublishedProp 4759
TypInfo.IsPublishedProp function 4759
TypInfo.IsStoredProp 4760
TypInfo.IsStoredProp function 4760
TypInfo.PPropInfo 4789
TypInfo.PPropInfo type 4789
TypInfo.PPropList 4790
TypInfo.PPropList type 4790
TypInfo.PPTypeInfo 4790
TypInfo.PPTypeInfo type 4790
TypInfo.PropIsType 4761
TypInfo.PropIsType function 4761
TypInfo.PropType 4761
TypInfo.PropType function 4761
TypInfo.PTypeData 4791
TypInfo.PTypeData type 4791
TypInfo.PTypeInfo 4794
TypInfo.PTypeInfo type 4794
TypInfo.SamePropTypeName 4762
TypInfo.SamePropTypeName function 4762
TypInfo.SetAnsiStrProp 4762, 4763
TypInfo.SetAnsiStrProp function 4762, 4763
TypInfo.SetDynArrayProp 4763
TypInfo.SetDynArrayProp function 4763
TypInfo.SetEnumProp 4764
TypInfo.SetEnumProp function 4764
TypInfo.SetFloatProp 4766
TypInfo.SetFloatProp function 4766
TypInfo.SetInt64Prop 4767
TypInfo.SetInt64Prop function 4767
TypInfo.SetInterfaceProp 4768
TypInfo.SetInterfaceProp function 4768
TypInfo.SetMethodProp 4769
TypInfo.SetMethodProp function 4769
TypInfo.SetObjectProp 4770
TypInfo.SetObjectProp function 4770
TypInfo.SetOrdProp 4770
TypInfo.SetOrdProp function 4770
TypInfo.SetPropValue 4771
TypInfo.SetPropValue function 4771
TypInfo.SetSetProp 4772
TypInfo.SetSetProp function 4772
TypInfo.SetStrProp 4773
TypInfo.SetStrProp function 4773
TypInfo.SetToString 4774
TypInfo.SetToString function 4774
TypInfo.SetUnicodeStrProp 4775
TypInfo.SetUnicodeStrProp function 4775
TypInfo.SetVariantProp 4775
TypInfo.SetVariantProp function 4775
TypInfo.SetWideStrProp 4776
TypInfo.SetWideStrProp function 4776
TypInfo.ShortStringBase 4797

TypInfo.ShortStringBase type 4797
TypInfo.SortPropList 4777
TypInfo.SortPropList function 4777
TypInfo.StringToSet 4778
TypInfo.StringToSet function 4778
TypInfo.TFloatType 4779
TypInfo.TFloatType enumeration 4779
TypInfo.TIntfFlag 4780
TypInfo.TIntfFlag enumeration 4780
TypInfo.TIntfFlags 4797
TypInfo.TIntfFlags type 4797
TypInfo.TIntfFlagsBase 4798
TypInfo.TIntfFlagsBase type 4798
TypInfo.tkAny 4802
TypInfo.tkAny constant 4802
TypInfo.tkMethods 4802
TypInfo.tkMethods constant 4802
TypInfo.tkProperties 4802
TypInfo.tkProperties constant 4802
TypInfo.TManagedField 4780
TypInfo.TManagedField record 4780
TypInfo.TMethodKind 4780
TypInfo.TMethodKind enumeration 4780
TypInfo.TOrdType 4782
TypInfo.TOrdType enumeration 4782
TypInfo.TParamFlag 4783
TypInfo.TParamFlag enumeration 4783
TypInfo.TParamFlags 4799
TypInfo.TParamFlags type 4799
TypInfo.TParamFlagsBase 4799
TypInfo.TParamFlagsBase type 4799
TypInfo.TPropData 4783
TypInfo.TPropData record 4783
TypInfo.TPropInfo 4784
TypInfo.TPropInfo record 4784
TypInfo.TPropInfoProc 4799
TypInfo.TPropInfoProc type 4799
TypInfo.TPropList 4800
TypInfo.TPropList type 4800
TypInfo.TPublishableVariantType 4728
TypInfo.TPublishableVariantType.GetInstance 4732
TypInfo.TPublishableVariantType.GetProperty 4732
TypInfo.TPublishableVariantType.SetProperty 4733
TypInfo.TTypeData 4785
TypInfo.TTypeData record 4785
TypInfo.TTypeInfo 4786
TypInfo.TTypeInfo record 4786
TypInfo.TTypeKind 4787
TypInfo.TTypeKind enumeration 4787
TypInfo.TTypeKinds 4800
TypInfo.TTypeKinds type 4800
TZCompressionStream class 7851
about TZCompressionStream class 7851
CompressionRate 7854
Create 7852
Create constructor 7852
Destroy 7852
OnProgress 7855
Read 7853
Seek 7854
TZCompressionStream events 7855
TZCompressionStream Members 7851
TZCompressionStream methods 7853
TZCompressionStream properties 7854
Write 7854
TZDecompressionStream class 7855
about TZDecompressionStream class 7855
Create 7856
Destroy 7856
OnProgress 7858
Read 7857
Seek 7857
TZDecompressionStream events 7858
TZDecompressionStream Members 7855
TZDecompressionStream methods 7857
Write 7857

U

UDDIHelper 4805
UDDIHelper namespace 4805
Functions 4806
UDDIHelper.GetBindingkeyAccessPoint 4806
UDDIHelper.GetBindingkeyAccessPoint function 4806
UnicodeString class 3393
!= 3423
[] 3424
[] operator 3423
~UnicodeString 3396
+ 3424
+= 3425
< 3426
<= 3426
= 3427
== 3427
> 3428
>= 3428
about UnicodeString class 3393
BytesOf 3401
ByteType 3401
c_str 3402
cat_printf 3402
cat_sprintf 3403
cat_vprintf 3403
CodePage 3404
Compare 3404
CompareIC 3405
CurrToStr 3405
CurrToStrF 3405
data 3406
Delete 3406
ElementSize 3407
EnsureUnicode 3407
FloatToStrF 3407
FmtLoadStr 3408
Format 3409
FormatFloat 3409
GetRec 3411
GetRec method 3411
Insert 3411
IntToHex 3412
IsDelimiter 3412
IsEmpty 3412
IsLeadSurrogate 3412
IsPathDelimiter 3413
IsTrailSurrogate 3413
LastChar 3413
LastDelimiter 3413
Length 3414
LoadStr 3414
LoadStr method 3414
LoadString 3415
LowerCase 3415
Pos 3416
printf 3416
RefCount 3416
SetLength 3417
sprintf 3417
StringOfChar 3417
SubString 3418
swap 3418
t_str 3418
ThrowIfOutOfRange 3419
ToDouble 3419
ToInt 3419
ToIntDef 3420
Trim 3420
TrimLeft 3420
TrimRight 3421
UnicodeString 3397
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) 3429
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) 3430
UnicodeString constructor 3396
UnicodeString enumerations 3399

UnicodeString friends 3429
UnicodeString Members 3394
UnicodeString methods 3400
UnicodeString operators 3423
UnicodeString structures 3398
Unique 3421
UpperCase 3421
vprintf 3422
w_str 3422

V

ValEdit 4807
ValEdit namespace 4807
Classes 4808
Structs, Records, Enums 4921
Types 4923
ValEdit.TDisplayOption 4921
ValEdit.TDisplayOption enumeration 4921
ValEdit.TDisplayOptions 4923
ValEdit.TDisplayOptions type 4923
ValEdit.TGetPickListEvent 4923
ValEdit.TGetPickListEvent type 4923
ValEdit.TItemProp 4808
ValEdit.TItemProp.AssignTo 4810
ValEdit.TItemProp.Create 4809
ValEdit.TItemProp.Destroy 4809
ValEdit.TItemProp.EditMask 4811
ValEdit.TItemProp.EditStyle 4812
ValEdit.TItemProp.HasPickList 4810
ValEdit.TItemProp.KeyDesc 4812
ValEdit.TItemProp.MaxLength 4813
ValEdit.TItemProp.PickList 4813
ValEdit.TItemProp.ReadOnly 4814
ValEdit.TItemProp.UpdateEdit 4811
ValEdit.TItemProps 4924
ValEdit.TItemProps type 4924
ValEdit.TKeyOption 4922
ValEdit.TKeyOption enumeration 4922
ValEdit.TKeyOptions 4924
ValEdit.TKeyOptions type 4924
ValEdit.TOnValidateEvent 4924
ValEdit.TOnValidateEvent type 4924
ValEdit.TValueListEditor 4814
ValEdit.TValueListEditor.AdjustColWidths 4819
ValEdit.TValueListEditor.AdjustRowCount 4819
ValEdit.TValueListEditor.Align 4831
ValEdit.TValueListEditor.Anchors 4835
ValEdit.TValueListEditor.BiDiMode 4835
ValEdit.TValueListEditor.BorderStyle 4836
ValEdit.TValueListEditor.CanEditModify 4820
ValEdit.TValueListEditor.Cells 4836
ValEdit.TValueListEditor.ColCount 4836
ValEdit.TValueListEditor.Color 4837
ValEdit.TValueListEditor.ColWidthsChanged 4820
ValEdit.TValueListEditor.Constraints 4838
ValEdit.TValueListEditor.Create 4817
ValEdit.TValueListEditor.CreateEditor 4820
ValEdit.TValueListEditor.CreateWnd 4821
ValEdit.TValueListEditor.Ctl3D 4838
ValEdit.TValueListEditor.DefaultColWidth 4839
ValEdit.TValueListEditor.DefaultDrawing 4840
ValEdit.TValueListEditor.DefaultRowHeight 4841
ValEdit.TValueListEditor.DeleteRow 4821
ValEdit.TValueListEditor.Destroy 4818
ValEdit.TValueListEditor.DisplayOptions 4841
ValEdit.TValueListEditor.DoExit 4822
ValEdit.TValueListEditor.DoOnValidate 4822
ValEdit.TValueListEditor.DoubleBuffered 4842
ValEdit.TValueListEditor.DragCursor 4843
ValEdit.TValueListEditor.DragKind 4844
ValEdit.TValueListEditor.DragMode 4844
ValEdit.TValueListEditor.DrawCell 4822
ValEdit.TValueListEditor.DropDownRows 4846
ValEdit.TValueListEditor.EditList 4846
ValEdit.TValueListEditor.EditListGetItems 4823
ValEdit.TValueListEditor.Enabled 4847
ValEdit.TValueListEditor.FindRow 4823
ValEdit.TValueListEditor.FixedColor 4847

ValEdit.TValueListEditor.FixedCols 4847
ValEdit.TValueListEditor.Font 4848
ValEdit.TValueListEditor.GetCell 4823
ValEdit.TValueListEditor.GetColCount 4824
ValEdit.TValueListEditor.GetEditLimit 4824
ValEdit.TValueListEditor.GetEditMask 4824
ValEdit.TValueListEditor.GetEditStyle 4824
ValEdit.TValueListEditor.GetEditText 4825
ValEdit.TValueListEditor.GetPickList 4825
ValEdit.TValueListEditor.GetRowCount 4825
ValEdit.TValueListEditor.GridLineWidth 4849
ValEdit.TValueListEditor.InsertRow 4825
ValEdit.TValueListEditor.IsEmptyRow 4826
ValEdit.TValueListEditor.ItemProps 4851
ValEdit.TValueListEditor.KeyDown 4826
ValEdit.TValueListEditor.KeyOptions 4851
ValEdit.TValueListEditor.Keys 4851
ValEdit.TValueListEditor.MouseDown 4827
ValEdit.TValueListEditor.OnClick 4868
ValEdit.TValueListEditor.OnContextPopup 4870
ValEdit.TValueListEditor.OnDblClick 4871
ValEdit.TValueListEditor.OnDragDrop 4871
ValEdit.TValueListEditor.OnDragOver 4874
ValEdit.TValueListEditor.OnDrawCell 4878
ValEdit.TValueListEditor.OnEditButtonClick 4879
ValEdit.TValueListEditor.OnEndDock 4880
ValEdit.TValueListEditor.OnEndDrag 4880
ValEdit.TValueListEditor.OnEnter 4881
ValEdit.TValueListEditor.OnExit 4883
ValEdit.TValueListEditor.OnGetEditMask 4886
ValEdit.TValueListEditor.OnGetEditText 4887
ValEdit.TValueListEditor.OnGetPickList 4889
ValEdit.TValueListEditor.OnKeyDown 4889
ValEdit.TValueListEditor.OnKeyPress 4893
ValEdit.TValueListEditor.OnKeyUp 4894
ValEdit.TValueListEditor.OnMouseActivate 4896
ValEdit.TValueListEditor.OnMouseDown 4897
ValEdit.TValueListEditor.OnMouseEnter 4899
ValEdit.TValueListEditor.OnMouseLeave 4899
ValEdit.TValueListEditor.OnMouseMove 4899
ValEdit.TValueListEditor.OnMouseUp 4902
ValEdit.TValueListEditor.OnMouseWheelDown 4904
ValEdit.TValueListEditor.OnMouseWheelUp 4904
ValEdit.TValueListEditor.OnRowMoved 4905
ValEdit.TValueListEditor.OnSelectCell 4906
ValEdit.TValueListEditor.OnSetEditText 4908
ValEdit.TValueListEditor.OnStartDock 4908
ValEdit.TValueListEditor.OnStartDrag 4909
ValEdit.TValueListEditor.OnStringsChange 4909
ValEdit.TValueListEditor.OnStringsChanging 4910
ValEdit.TValueListEditor.OnTopLeftChanged 4910
ValEdit.TValueListEditor.OnValidate 4910
ValEdit.TValueListEditor.Options 4852
ValEdit.TValueListEditor.ParentBiDiMode 4853
ValEdit.TValueListEditor.ParentColor 4854
ValEdit.TValueListEditor.ParentCtl3D 4854
ValEdit.TValueListEditor.ParentDoubleBuffered 4855
ValEdit.TValueListEditor.ParentFont 4855
ValEdit.TValueListEditor.ParentShowHint 4855
ValEdit.TValueListEditor.PopupMenu 4856
ValEdit.TValueListEditor.Refresh 4827
ValEdit.TValueListEditor.Resize 4828
ValEdit.TValueListEditor.RestoreCurrentRow 4828
ValEdit.TValueListEditor.RowCount 4861
ValEdit.TValueListEditor.RowMoved 4828
ValEdit.TValueListEditor.ScrollBars 4861
ValEdit.TValueListEditor.SelectCell 4829
ValEdit.TValueListEditor.SetCell 4829
ValEdit.TValueListEditor.SetEditText 4829
ValEdit.TValueListEditor.SetStrings 4829
ValEdit.TValueListEditor.ShowHint 4862
ValEdit.TValueListEditor.Strings 4863
ValEdit.TValueListEditor.StringsChanging 4830
ValEdit.TValueListEditor.TabOrder 4864
ValEdit.TValueListEditor.TitleCaptions 4864
ValEdit.TValueListEditor.TitleCaptionsStored 4830
ValEdit.TValueListEditor.Values 4865
ValEdit.TValueListEditor.Visible 4865

ValEdit.TValueListEditor.VisibleColCount 4867
ValEdit.TValueListEditor.VisibleRowCount 4867
ValEdit.TValueListStrings 4911
ValEdit.TValueListStrings.Assign 4913
ValEdit.TValueListStrings.Changed 4916
ValEdit.TValueListStrings.Changing 4916
ValEdit.TValueListStrings.Clear 4916
ValEdit.TValueListStrings.Create 4912
ValEdit.TValueListStrings.CustomSort 4916
ValEdit.TValueListStrings.Delete 4917
ValEdit.TValueListStrings.Destroy 4912
ValEdit.TValueListStrings.Exchange 4917
ValEdit.TValueListStrings.FindItemProp 4918
ValEdit.TValueListStrings.InsertItem 4918
ValEdit.TValueListStrings.ItemProps 4919
ValEdit.TValueListStrings.KeyIsValid 4918
ValEdit.TValueListStrings.Put 4919
ValEdit.TValueListStrings.UpdateCount 4919
VarCmplx 4925
VarCmplx namespace 4925
Functions 4927
Variables 4971
VarCmplx.ComplexNumberDefuzzAtZero 4971
VarCmplx.ComplexNumberDefuzzAtZero variable 4971
VarCmplx.ComplexNumberSymbol 4971
VarCmplx.ComplexNumberSymbol variable 4971
VarCmplx.ComplexNumberSymbolBeforeImaginary 4972
VarCmplx.ComplexNumberSymbolBeforeImaginary variable 4972
VarCmplx.VarAsComplex 4928
VarCmplx.VarAsComplex function 4928
VarCmplx.VarComplex 4929
VarCmplx.VarComplex function 4929
VarCmplx.VarComplexAbs 4929
VarCmplx.VarComplexAbs function 4929
VarCmplx.VarComplexAbsSqr 4930
VarCmplx.VarComplexAbsSqr function 4930
VarCmplx.VarComplexAngle 4931
VarCmplx.VarComplexAngle function 4931
VarCmplx.VarComplexArcCos 4931
VarCmplx.VarComplexArcCos function 4931
VarCmplx.VarComplexArcCosH 4932
VarCmplx.VarComplexArcCosH function 4932
VarCmplx.VarComplexArcCot 4933
VarCmplx.VarComplexArcCot function 4933
VarCmplx.VarComplexArcCotH 4934
VarCmplx.VarComplexArcCotH function 4934
VarCmplx.VarComplexArcCsc 4935
VarCmplx.VarComplexArcCsc function 4935
VarCmplx.VarComplexArcCscH 4937
VarCmplx.VarComplexArcCscH function 4937
VarCmplx.VarComplexArcSec 4938
VarCmplx.VarComplexArcSec function 4938
VarCmplx.VarComplexArcSecH 4939
VarCmplx.VarComplexArcSecH function 4939
VarCmplx.VarComplexArcSin 4940
VarCmplx.VarComplexArcSin function 4940
VarCmplx.VarComplexArcSinH 4941
VarCmplx.VarComplexArcSinH function 4941
VarCmplx.VarComplexArcTan 4942
VarCmplx.VarComplexArcTan function 4942
VarCmplx.VarComplexArcTanH 4943
VarCmplx.VarComplexArcTanH function 4943
VarCmplx.VarComplexConjugate 4944
VarCmplx.VarComplexConjugate function 4944
VarCmplx.VarComplexCos 4945
VarCmplx.VarComplexCos function 4945
VarCmplx.VarComplexCosH 4946
VarCmplx.VarComplexCosH function 4946
VarCmplx.VarComplexCot 4947
VarCmplx.VarComplexCot function 4947
VarCmplx.VarComplexCotH 4948
VarCmplx.VarComplexCotH function 4948
VarCmplx.VarComplexCreate 4949
VarCmplx.VarComplexCreate function 4949
VarCmplx.VarComplexCsc 4950
VarCmplx.VarComplexCsc function 4950
VarCmplx.VarComplexCscH 4951
VarCmplx.VarComplexCscH function 4951

VarCmplx.VarComplexExp 4952
VarCmplx.VarComplexExp function 4952
VarCmplx.VarComplexFromPolar 4953
VarCmplx.VarComplexFromPolar function 4953
VarCmplx.VarComplexInverse 4954
VarCmplx.VarComplexInverse function 4954
VarCmplx.VarComplexLn 4954
VarCmplx.VarComplexLn function 4954
VarCmplx.VarComplexLog10 4955
VarCmplx.VarComplexLog10 function 4955
VarCmplx.VarComplexLog2 4956
VarCmplx.VarComplexLog2 function 4956
VarCmplx.VarComplexLogN 4956
VarCmplx.VarComplexLogN function 4956
VarCmplx.VarComplexPower 4957
VarCmplx.VarComplexPower function 4957
VarCmplx.VarComplexSec 4958
VarCmplx.VarComplexSec function 4958
VarCmplx.VarComplexSecH 4959
VarCmplx.VarComplexSecH function 4959
VarCmplx.VarComplexSign 4960
VarCmplx.VarComplexSign function 4960
VarCmplx.VarComplexSimplify 4960
VarCmplx.VarComplexSimplify function 4960
VarCmplx.VarComplexSin 4961
VarCmplx.VarComplexSin function 4961
VarCmplx.VarComplexSinH 4962
VarCmplx.VarComplexSinH function 4962
VarCmplx.VarComplexSqr 4963
VarCmplx.VarComplexSqr function 4963
VarCmplx.VarComplexSqrt 4964
VarCmplx.VarComplexSqrt function 4964
VarCmplx.VarComplexTan 4964
VarCmplx.VarComplexTan function 4964
VarCmplx.VarComplexTanH 4965
VarCmplx.VarComplexTanH function 4965
VarCmplx.VarComplexTimesImaginary 4967
VarCmplx.VarComplexTimesImaginary function 4967
VarCmplx.VarComplexTimesNegI 4967
VarCmplx.VarComplexTimesNegI function 4967
VarCmplx.VarComplexTimesPosI 4968
VarCmplx.VarComplexTimesPosI function 4968
VarCmplx.VarComplexTimesReal 4968
VarCmplx.VarComplexTimesReal function 4968
VarCmplx.VarComplexToPolar 4969
VarCmplx.VarComplexToPolar function 4969
VarCmplx.VarIsComplex 4969
VarCmplx.VarIsComplex function 4969
VarConv 4973
VarConv namespace 4973
Functions 4974
VarConv.VarAsConvert 4974
VarConv.VarAsConvert function 4974
VarConv.VarConvert 4975
VarConv.VarConvert function 4975
VarConv.VarConvertCreate 4976
VarConv.VarConvertCreate function 4976
VarConv.VarIsConvert 4977
VarConv.VarIsConvert function 4977
VarHlpr 4979
VarHlpr namespace 4979
Functions 4981
VarHlpr.VariantAdd 4981
VarHlpr.VariantAdd function 4981
VarHlpr.VariantAdd2 4982
VarHlpr.VariantAdd2 function 4982
VarHlpr.VariantAnd 4982
VarHlpr.VariantAnd function 4982
VarHlpr.VariantAnd2 4983
VarHlpr.VariantAnd2 function 4983
VarHlpr.VariantArrayRedim 4983
VarHlpr.VariantArrayRedim function 4983
VarHlpr.VariantCast 4984
VarHlpr.VariantCast function 4984
VarHlpr.VariantClear 4984
VarHlpr.VariantClear function 4984
VarHlpr.VariantCpy 4985
VarHlpr.VariantCpy function 4985

VarHlpr.VariantDiv 4985
VarHlpr.VariantDiv function 4985
VarHlpr.VariantDiv2 4986
VarHlpr.VariantDiv2 function 4986
VarHlpr.VariantFromUnicodeString 4986
VarHlpr.VariantFromUnicodeString function 4986
VarHlpr.VariantGetElement 4987
VarHlpr.VariantGetElement function 4987
VarHlpr.VariantMod 4988
VarHlpr.VariantMod function 4988
VarHlpr.VariantMod2 4988
VarHlpr.VariantMod2 function 4988
VarHlpr.VariantMul 4989
VarHlpr.VariantMul function 4989
VarHlpr.VariantMul2 4989
VarHlpr.VariantMul2 function 4989
VarHlpr.VariantNeg 4990
VarHlpr.VariantNeg function 4990
VarHlpr.VariantNot 4990
VarHlpr.VariantNot function 4990
VarHlpr.VariantOr 4990
VarHlpr.VariantOr function 4990
VarHlpr.VariantOr2 4991
VarHlpr.VariantOr2 function 4991
VarHlpr.VariantPutElement 4992
VarHlpr.VariantPutElement function 4992
VarHlpr.VariantShl 4992
VarHlpr.VariantShl function 4992
VarHlpr.VariantShl2 4993
VarHlpr.VariantShl2 function 4993
VarHlpr.VariantShr 4993
VarHlpr.VariantShr function 4993
VarHlpr.VariantShr2 4994
VarHlpr.VariantShr2 function 4994
VarHlpr.VariantSub 4994
VarHlpr.VariantSub function 4994
VarHlpr.VariantSub2 4995
VarHlpr.VariantSub2 function 4995
VarHlpr.VariantToUnicodeString 4995
VarHlpr.VariantToUnicodeString function 4995
VarHlpr.VariantXor 4996
VarHlpr.VariantXor function 4996
VarHlpr.VariantXor2 4996
VarHlpr.VariantXor2 function 4996
Variant class 3430
- 3446, 3447
- operator 3446
! 3447
!= 3448
!= operator 3448
% 3449
% operator 3448
%= 3449
& 3450
& operator 3449
&= 3450
* 3451
* operator 3450
*= 3451
/ 3452
/ operator 3452
/= 3452
^ 3453
^ operator 3453
^= 3453
__int64 3454
| 3454
| operator 3454
|= 3454
~Variant 3433
+ 3455
+ operator 3455
+= 3455
< 3456
< operator 3456
<< 3457
<< operator 3456
<<= 3458

<= 3458
<= operator 3458
= 3461
-= 3463
= operator 3459
== 3463
== operator 3463
> 3464, 3465
> operator 3464
>= 3466
>= operator 3465
>> 3467
>> operator 3467
>>= 3468
about Variant class 3430
AnsiString 3468
ArrayDimCount 3437
ArrayHighBound 3438
ArrayLock 3438
ArrayLowBound 3438
ArrayRedim 3439
ArrayUnlock 3439
AsType 3439
bool 3468
ChangeType 3439
char 3468
Clear 3440
Currency 3469
Currency* 3469
double 3469
double* 3470
Empty 3440
float 3470
float* 3470
GetBaseVariant 3440
GetElement 3441
GetElement method 3441
GetError 3441
IDispatch* 3470
IInterface* 3471
int 3471
int* 3471
IsArray 3442
IsEmpty 3442
IsNull 3442
IUnknown* 3471
long 3472
long* 3472
NoParam 3443
PutElement 3443
PutElement method 3443
Sanitized 3444
SetError 3444
short 3472
short* 3472
signed char 3473
signed char* 3473
TDateTime 3473
TDateTime* 3473
Type 3444
UnicodeString 3474
unsigned char 3474
unsigned char* 3474
unsigned int 3474
unsigned int* 3474
unsigned long 3475
unsigned long* 3475
unsigned short 3475
unsigned short* 3475
Variant 3435
Variant constructor 3434
Variant Members 3431
Variant methods 3437
Variant operators 3445
WideString 3476
Variants 4997
Variants namespace 4997
Classes 5002

Functions 5048
Interfaces 5045
Structs, Records, Enums 5105
Types 5108
Variables 5110
Variants._DispInvoke 5050
Variants._DispInvoke function 5050
Variants._OleVarFromInt 5051
Variants._OleVarFromInt function 5051
Variants._OleVarFromInt64 5051
Variants._OleVarFromInt64 function 5051
Variants._OleVarFromLStr 5051
Variants._OleVarFromLStr function 5051
Variants._OleVarFromPStr 5052
Variants._OleVarFromPStr function 5052
Variants._OleVarFromUInt64 5052
Variants._OleVarFromUInt64 function 5052
Variants._OleVarFromUStr 5052
Variants._OleVarFromUStr function 5052
Variants._OleVarFromVar 5053
Variants._OleVarFromVar function 5053
Variants._VarAdd 5053
Variants._VarAdd function 5053
Variants._VarAddRef 5053
Variants._VarAddRef function 5053
Variants._VarAnd 5054
Variants._VarAnd function 5054
Variants._VarArrayGet 5054
Variants._VarArrayGet function 5054
Variants._VarArrayPut 5054
Variants._VarArrayPut function 5054
Variants._VarArrayRedim 5055
Variants._VarArrayRedim function 5055
Variants._VarCast 5055
Variants._VarCast function 5055
Variants._VarCastOle 5055
Variants._VarCastOle function 5055
Variants._VarClear 5056
Variants._VarClear function 5056
Variants._VarClr 5056
Variants._VarClr function 5056
Variants._VarCmpEQ 5056
Variants._VarCmpEQ function 5056
Variants._VarCmpGE 5057
Variants._VarCmpGE function 5057
Variants._VarCmpGT 5057
Variants._VarCmpGT function 5057
Variants._VarCmpLE 5057
Variants._VarCmpLE function 5057
Variants._VarCmpLT 5058
Variants._VarCmpLT function 5058
Variants._VarCmpNE 5058
Variants._VarCmpNE function 5058
Variants._VarCopy 5058
Variants._VarCopy function 5058
Variants._VarCopyNoInd 5059
Variants._VarCopyNoInd function 5059
Variants._VarDiv 5059
Variants._VarDiv function 5059
Variants._VarFromBool 5059
Variants._VarFromBool function 5059
Variants._VarFromCurr 5060
Variants._VarFromCurr function 5060
Variants._VarFromDisp 5060
Variants._VarFromDisp function 5060
Variants._VarFromDynArray 5060
Variants._VarFromDynArray function 5060
Variants._VarFromInt 5061
Variants._VarFromInt function 5061
Variants._VarFromInt64 5061
Variants._VarFromInt64 function 5061
Variants._VarFromIntf 5061
Variants._VarFromIntf function 5061
Variants._VarFromLStr 5062
Variants._VarFromLStr function 5062
Variants._VarFromPStr 5062
Variants._VarFromPStr function 5062
Variants._VarFromReal 5062

Variants._VarFromReal function 5062
Variants._VarFromTDateTime 5063
Variants._VarFromTDateTime function 5063
Variants._VarFromUInt64 5063
Variants._VarFromUInt64 function 5063
Variants._VarFromUStr 5063
Variants._VarFromUStr function 5063
Variants._VarFromWChar 5064
Variants._VarFromWChar function 5064
Variants._VarFromWStr 5064
Variants._VarFromWStr function 5064
Variants._VarMod 5064
Variants._VarMod function 5064
Variants._VarMul 5065
Variants._VarMul function 5065
Variants._VarNeg 5065
Variants._VarNeg function 5065
Variants._VarNot 5065
Variants._VarNot function 5065
Variants._VarOr 5066
Variants._VarOr function 5066
Variants._VarRDiv 5066
Variants._VarRDiv function 5066
Variants._VarShl 5066
Variants._VarShl function 5066
Variants._VarShr 5067
Variants._VarShr function 5067
Variants._VarSub 5067
Variants._VarSub function 5067
Variants._VarToBool 5067
Variants._VarToBool function 5067
Variants._VarToCurrency 5068
Variants._VarToCurrency function 5068
Variants._VarToDisp 5068
Variants._VarToDisp function 5068
Variants._VarToDynArray 5068
Variants._VarToDynArray function 5068
Variants._VarToInt64 5069
Variants._VarToInt64 function 5069
Variants._VarToInteger 5069
Variants._VarToInteger function 5069
Variants._VarToIntf 5069
Variants._VarToIntf function 5069
Variants._VarToLStr 5070
Variants._VarToLStr function 5070
Variants._VarToPStr 5070
Variants._VarToPStr function 5070
Variants._VarToReal 5070
Variants._VarToReal function 5070
Variants._VarToUInt64 5071
Variants._VarToUInt64 function 5071
Variants._VarToUStr 5071
Variants._VarToUStr function 5071
Variants._VarToWStr 5071
Variants._VarToWStr function 5071
Variants._VarXor 5072
Variants._VarXor function 5072
Variants.BooleanToStringRule 5110
Variants.BooleanToStringRule variable 5110
Variants.ChangeAnyProc 5111
Variants.ChangeAnyProc variable 5111
Variants.ClearAnyProc 5111
Variants.ClearAnyProc variable 5111
Variants.DynArrayFromVariant 5072
Variants.DynArrayFromVariant function 5072
Variants.DynArrayToVariant 5072
Variants.DynArrayToVariant function 5072
Variants.EmptyParam 5111
Variants.EmptyParam variable 5111
Variants.EVariantArrayCreateError 5002
Variants.EVariantArrayLockedError 5003
Variants.EVariantBadIndexError 5004
Variants.EVariantBadVarTypeError 5004
Variants.EVariantDispatchError 5005
Variants.EVariantInvalidArgError 5005
Variants.EVariantInvalidNullOpError 5006
Variants.EVariantInvalidOpError 5007
Variants.EVariantNotAnArrayError 5007

Variants.EVariantNotImplError 5008
Variants.EVariantOutOfMemoryError 5008
Variants.EVariantOverflowError 5009
Variants.EVariantRangeCheckError 5010
Variants.EVariantTypeCastError 5010
Variants.EVariantUnexpectedError 5011
Variants.FindCustomVariantType 5073
Variants.FindCustomVariantType function 5073
Variants.FindVarData 5074
Variants.FindVarData function 5074
Variants.HandleConversionException 5074
Variants.HandleConversionException function 5074
Variants.IVarInstanceReference 5045
Variants.IVarInstanceReference.GetInstance 5046
Variants.IVarInvokeable 5046
Variants.IVarInvokeable.DoFunction 5047
Variants.IVarInvokeable.DoProcedure 5047
Variants.IVarInvokeable.GetProperty 5047
Variants.IVarInvokeable.SetProperty 5047
Variants.Null 5075
Variants.Null function 5075
Variants.NullAsStringValue 5112
Variants.NullAsStringValue variable 5112
Variants.NullEqualityRule 5112
Variants.NullEqualityRule variable 5112
Variants.NullMagnitudeRule 5113
Variants.NullMagnitudeRule variable 5113
Variants.NullStrictConvert 5114
Variants.NullStrictConvert variable 5114
Variants.OleVariantInt64AsDouble 5114
Variants.OleVariantInt64AsDouble variable 5114
Variants.PackVarCreation 5115
Variants.PackVarCreation variable 5115
Variants.RefAnyProc 5115
Variants.RefAnyProc variable 5115
Variants.TAnyProc 5108
Variants.TAnyProc type 5108
Variants.TBooleanToStringRule 5105
Variants.TBooleanToStringRule enumeration 5105
Variants.TCustomVariantType 5011
Variants.TCustomVariantType._AddRef 5017
Variants.TCustomVariantType._Release 5017
Variants.TCustomVariantType.BinaryOp 5018
Variants.TCustomVariantType.Cast 5018
Variants.TCustomVariantType.CastTo 5019
Variants.TCustomVariantType.CastToOle 5019
Variants.TCustomVariantType.Clear 5019
Variants.TCustomVariantType.Compare 5020
Variants.TCustomVariantType.CompareOp 5021
Variants.TCustomVariantType.Copy 5022
Variants.TCustomVariantType.Create 5015
Variants.TCustomVariantType.Destroy 5015
Variants.TCustomVariantType.DispInvoke 5022
Variants.TCustomVariantType.IsClear 5023
Variants.TCustomVariantType.LeftPromotion 5023
Variants.TCustomVariantType.OlePromotion 5024
Variants.TCustomVariantType.QueryInterface 5025
Variants.TCustomVariantType.RaiseCastError 5025
Variants.TCustomVariantType.RaiseDispError 5026
Variants.TCustomVariantType.RaiseInvalidOp 5026
Variants.TCustomVariantType.RightPromotion 5027
Variants.TCustomVariantType.SimplisticClear 5028
Variants.TCustomVariantType.SimplisticCopy 5028
Variants.TCustomVariantType.UnaryOp 5029
Variants.TCustomVariantType.VarDataCast 5029
Variants.TCustomVariantType.VarDataCastTo 5030
Variants.TCustomVariantType.VarDataCastToOleStr 5030
Variants.TCustomVariantType.VarDataClear 5031
Variants.TCustomVariantType.VarDataCopy 5031
Variants.TCustomVariantType.VarDataCopyNoInd 5032
Variants.TCustomVariantType.VarDataFromOleStr 5032
Variants.TCustomVariantType.VarDataFromStr 5033
Variants.TCustomVariantType.VarDataInit 5033
Variants.TCustomVariantType.VarDataIsArray 5033
Variants.TCustomVariantType.VarDataIsByRef 5034
Variants.TCustomVariantType.VarDataIsEmptyParam 5034
Variants.TCustomVariantType.VarDataIsFloat 5034
Variants.TCustomVariantType.VarDataIsNumeric 5035

Variants.TCustomVariantType.VarDataIsOrdinal 5035
Variants.TCustomVariantType.VarDataIsStr 5036
Variants.TCustomVariantType.VarDataToStr 5036
Variants.TCustomVariantType.VarType 5037
Variants.TCustomVariantTypeClass 5108
Variants.TCustomVariantTypeClass type 5108
Variants.TInvokeableVariantType 5037
Variants.TInvokeableVariantType.DispInvoke 5041
Variants.TInvokeableVariantType.DoFunction 5041
Variants.TInvokeableVariantType.DoProcedure 5042
Variants.TInvokeableVariantType.FixupIdent 5043
Variants.TInvokeableVariantType.GetProperty 5043
Variants.TInvokeableVariantType.SetProperty 5044
Variants.TNullCompareRule 5105
Variants.TNullCompareRule enumeration 5105
Variants.TVarCompareResult 5106
Variants.TVarCompareResult enumeration 5106
Variants.TVarDataArray 5109
Variants.TVarDataArray type 5109
Variants.TVarDispProc 5109
Variants.TVarDispProc type 5109
Variants.TVariantRelationship 5106
Variants.TVariantRelationship enumeration 5106
Variants.Unassigned 5076
Variants.Unassigned function 5076
Variants.VarArrayAsPSafeArray 5077
Variants.VarArrayAsPSafeArray function 5077
Variants.VarArrayCreate 5077
Variants.VarArrayCreate function 5077
Variants.VarArrayCreateError 5078
Variants.VarArrayCreateError function 5078
Variants.VarArrayDimCount 5078
Variants.VarArrayDimCount function 5078
Variants.VarArrayGet 5079
Variants.VarArrayGet function 5079
Variants.VarArrayHighBound 5079
Variants.VarArrayHighBound function 5079
Variants.VarArrayLock 5080
Variants.VarArrayLock function 5080
Variants.VarArrayLowBound 5080
Variants.VarArrayLowBound function 5080
Variants.VarArrayOf 5081
Variants.VarArrayOf function 5081
Variants.VarArrayPut 5081
Variants.VarArrayPut function 5081
Variants.VarArrayRef 5082
Variants.VarArrayRef function 5082
Variants.VarArrayUnlock 5083
Variants.VarArrayUnlock function 5083
Variants.VarAsError 5083
Variants.VarAsError function 5083
Variants.VarAsType 5083
Variants.VarAsType function 5083
Variants.VarCastError 5084
Variants.VarCastError function 5084
Variants.VarCheckEmpty 5084
Variants.VarCheckEmpty function 5084
Variants.VarCompareValue 5085
Variants.VarCompareValue function 5085
Variants.VarCopyNoInd 5086
Variants.VarCopyNoInd function 5086
Variants.VarDispProc 5115
Variants.VarDispProc variable 5115
Variants.VarEnsureRange 5086
Variants.VarEnsureRange function 5086
Variants.VarFromDateTime 5087
Variants.VarFromDateTime function 5087
Variants.VarInRange 5087
Variants.VarInRange function 5087
Variants.VarInvalidNullOp 5088
Variants.VarInvalidNullOp function 5088
Variants.VarInvalidOp 5088
Variants.VarInvalidOp function 5088
Variants.VarIsArray 5089
Variants.VarIsArray function 5089
Variants.VarIsByRef 5089
Variants.VarIsByRef function 5089
Variants.VarIsClear 5090

Variants.VarIsClear function 5090
Variants.VarIsCustom 5090
Variants.VarIsCustom function 5090
Variants.VarIsEmpty 5091
Variants.VarIsEmpty function 5091
Variants.VarIsEmptyParam 5092
Variants.VarIsEmptyParam function 5092
Variants.VarIsError 5092
Variants.VarIsError function 5092
Variants.VarIsFloat 5093
Variants.VarIsFloat function 5093
Variants.VarIsNull 5093
Variants.VarIsNull function 5093
Variants.VarIsNumeric 5094
Variants.VarIsNumeric function 5094
Variants.VarIsOrdinal 5094
Variants.VarIsOrdinal function 5094
Variants.VarIsStr 5095
Variants.VarIsStr function 5095
Variants.VarIsType 5095
Variants.VarIsType function 5095
Variants.VarOverflowError 5096
Variants.VarOverflowError function 5096
Variants.VarRangeCheckError 5096
Variants.VarRangeCheckError function 5096
Variants.VarResultCheck 5097
Variants.VarResultCheck function 5097
Variants.VarSameValue 5097
Variants.VarSameValue function 5097
Variants.VarSupports 5098
Variants.VarSupports function 5098
Variants.VarToDateTime 5098
Variants.VarToDateTime function 5098
Variants.VarToStr 5099
Variants.VarToStr function 5099
Variants.VarToStrDef 5099
Variants.VarToStrDef function 5099
Variants.VarToWideStr 5100
Variants.VarToWideStr function 5100
Variants.VarToWideStrDef 5100
Variants.VarToWideStrDef function 5100
Variants.VarType 5101
Variants.VarType function 5101
Variants.VarTypeAsText 5102
Variants.VarTypeAsText function 5102
Variants.VarTypeIsValidArrayType 5103
Variants.VarTypeIsValidArrayType function 5103
Variants.VarTypeIsValidElementType 5103
Variants.VarTypeIsValidElementType function 5103
VarUtils 5117
VarUtils namespace 5117
    Classes 5120
    Constants 5143
    Functions 5124
    Structs, Records, Enums 5134
    Variables 5135
VarUtils.ARR_DISPATCH 5143
VarUtils.ARR_DISPATCH constant 5143
VarUtils.ARR_FIXEDSIZE 5144
VarUtils.ARR_FIXEDSIZE constant 5144
VarUtils.ARR_NONE 5144
VarUtils.ARR_NONE constant 5144
VarUtils.ARR_OLESTR 5144
VarUtils.ARR_OLESTR constant 5144
VarUtils.ARR_UNKNOWN 5145
VarUtils.ARR_UNKNOWN constant 5145
VarUtils.ARR_VARIANT 5145
VarUtils.ARR_VARIANT constant 5145
VarUtils.CMaxArrayVarType 5145
VarUtils.CMaxArrayVarType constant 5145
VarUtils.CMinArrayVarType 5146
VarUtils.CMinArrayVarType constant 5146
VarUtils.CVarTypeToElementInfo 5146
VarUtils.CVarTypeToElementInfo constant 5146
VarUtils.ESafeArrayBoundsError 5120
VarUtils.ESafeArrayCreateError 5120
VarUtils.ESafeArrayError 5121
VarUtils.ESafeArrayError.CreateHResult 5122

VarUtils.ESafeArrayError.ErrorCode 5123
VarUtils.ESafeArrayLockedError 5123
VarUtils.GenericOperations 5147
VarUtils.GenericOperations constant 5147
VarUtils.GenericSafeArrays 5147
VarUtils.GenericSafeArrays constant 5147
VarUtils.GenericSafeArrayUsesLibC 5147
VarUtils.GenericSafeArrayUsesLibC constant 5147
VarUtils.GenericVariants 5148
VarUtils.GenericVariants constant 5148
VarUtils.GenericVarUtils 5148
VarUtils.GenericVarUtils constant 5148
VarUtils.SafeArrayAccessData 5124
VarUtils.SafeArrayAccessData function 5124
VarUtils.SafeArrayAllocData 5125
VarUtils.SafeArrayAllocData function 5125
VarUtils.SafeArrayAllocDescriptor 5125
VarUtils.SafeArrayAllocDescriptor function 5125
VarUtils.SafeArrayCheck 5125
VarUtils.SafeArrayCheck function 5125
VarUtils.SafeArrayCopy 5126
VarUtils.SafeArrayCopy function 5126
VarUtils.SafeArrayCopyData 5126
VarUtils.SafeArrayCopyData function 5126
VarUtils.SafeArrayCreate 5126
VarUtils.SafeArrayCreate function 5126
VarUtils.SafeArrayDestroy 5127
VarUtils.SafeArrayDestroy function 5127
VarUtils.SafeArrayDestroyData 5127
VarUtils.SafeArrayDestroyData function 5127
VarUtils.SafeArrayDestroyDescriptor 5127
VarUtils.SafeArrayDestroyDescriptor function 5127
VarUtils.SafeArrayError 5128
VarUtils.SafeArrayError function 5128
VarUtils.SafeArrayGetDim 5128
VarUtils.SafeArrayGetDim function 5128
VarUtils.SafeArrayGetElement 5128
VarUtils.SafeArrayGetElement function 5128
VarUtils.SafeArrayGetElemSize 5129
VarUtils.SafeArrayGetElemSize function 5129
VarUtils.SafeArrayGetLBound 5129
VarUtils.SafeArrayGetLBound function 5129
VarUtils.SafeArrayGetUBound 5129
VarUtils.SafeArrayGetUBound function 5129
VarUtils.SafeArrayLock 5130
VarUtils.SafeArrayLock function 5130
VarUtils.SafeArrayPtrOfIndex 5130
VarUtils.SafeArrayPtrOfIndex function 5130
VarUtils.SafeArrayPutElement 5130
VarUtils.SafeArrayPutElement function 5130
VarUtils.SafeArrayRedim 5131
VarUtils.SafeArrayRedim function 5131
VarUtils.SafeArrayUnaccessData 5131
VarUtils.SafeArrayUnaccessData function 5131
VarUtils.SafeArrayUnlock 5131
VarUtils.SafeArrayUnlock function 5131
VarUtils.TVarTypeToElementInfo 5134
VarUtils.TVarTypeToElementInfo record 5134
VarUtils.VAR_ARRAYISLOCKED 5148
VarUtils.VAR_ARRAYISLOCKED constant 5148
VarUtils.VAR_BADINDEX 5149
VarUtils.VAR_BADINDEX constant 5149
VarUtils.VAR_BADVARTYPE 5149
VarUtils.VAR_BADVARTYPE constant 5149
VarUtils.VAR_CMP_EQ 5149
VarUtils.VAR_CMP_EQ constant 5149
VarUtils.VAR_CMP_GT 5150
VarUtils.VAR_CMP_GT constant 5150
VarUtils.VAR_CMP_LT 5150
VarUtils.VAR_CMP_LT constant 5150
VarUtils.VAR_CMP_NULL 5150
VarUtils.VAR_CMP_NULL constant 5150
VarUtils.VAR_EXCEPTION 5151
VarUtils.VAR_EXCEPTION constant 5151
VarUtils.VAR_INVALIDARG 5151
VarUtils.VAR_INVALIDARG constant 5151
VarUtils.VAR_LOCALE_USER_DEFAULT 5151
VarUtils.VAR_LOCALE_USER_DEFAULT constant 5151

VarUtils.VAR_NOTIMPL 5152
VarUtils.VAR_NOTIMPL constant 5152
VarUtils.VAR_OK 5152
VarUtils.VAR_OK constant 5152
VarUtils.VAR_OUTOFMEMORY 5152
VarUtils.VAR_OUTOFMEMORY constant 5152
VarUtils.VAR_OVERFLOW 5153
VarUtils.VAR_OVERFLOW constant 5153
VarUtils.VAR_PARAMNOTFOUND 5153
VarUtils.VAR_PARAMNOTFOUND constant 5153
VarUtils.VAR_TYPEMISMATCH 5153
VarUtils.VAR_TYPEMISMATCH constant 5153
VarUtils.VAR_UNEXPECTED 5154
VarUtils.VAR_UNEXPECTED constant 5154
VarUtils.VarAdd 5135
VarUtils.VarAdd variable 5135
VarUtils.VarAnd 5136
VarUtils.VarAnd variable 5136
VarUtils.VarBoolFromStr 5136
VarUtils.VarBoolFromStr variable 5136
VarUtils.VarBstrFromBool 5136
VarUtils.VarBstrFromBool variable 5136
VarUtils.VarBstrFromCy 5137
VarUtils.VarBstrFromCy variable 5137
VarUtils.VarBstrFromDate 5137
VarUtils.VarBstrFromDate variable 5137
VarUtils.VarCmp 5137
VarUtils.VarCmp variable 5137
VarUtils.VarCyFromStr 5138
VarUtils.VarCyFromStr variable 5138
VarUtils.VarDateFromStr 5138
VarUtils.VarDateFromStr variable 5138
VarUtils.VarDiv 5138
VarUtils.VarDiv variable 5138
VarUtils.VarExceptionToResult 5132
VarUtils.VarExceptionToResult function 5132
VarUtils.VarI4FromStr 5139
VarUtils.VarI4FromStr variable 5139
VarUtils.VariantChangeType 5132
VarUtils.VariantChangeType function 5132
VarUtils.VariantChangeTypeEx 5139
VarUtils.VariantChangeTypeEx variable 5139
VarUtils.VariantClear 5132
VarUtils.VariantClear function 5132
VarUtils.VariantCopy 5133
VarUtils.VariantCopy function 5133
VarUtils.VariantCopyInd 5133
VarUtils.VariantCopyInd function 5133
VarUtils.VariantInit 5133
VarUtils.VariantInit function 5133
VarUtils.VarIDiv 5139
VarUtils.VarIDiv variable 5139
VarUtils.VarMod 5140
VarUtils.VarMod variable 5140
VarUtils.VarMul 5140
VarUtils.VarMul variable 5140
VarUtils.VarNeg 5140
VarUtils.VarNeg variable 5140
VarUtils.VarNot 5141
VarUtils.VarNot variable 5141
VarUtils.VarOr 5141
VarUtils.VarOr variable 5141
VarUtils.VarR4FromStr 5141
VarUtils.VarR4FromStr variable 5141
VarUtils.VarR8FromStr 5142
VarUtils.VarR8FromStr variable 5142
VarUtils.VarSub 5142
VarUtils.VarSub variable 5142
VarUtils.VarXor 5142
VarUtils.VarXor variable 5142
VCLCom 5155
VCLCom namespace 5155
Classes 5156
VCLCom.TComponentFactory 5156
VCLCom.TComponentFactory.Create 5157
VCLCom.TComponentFactory.CreateComObject 5157
VCLCom.TComponentFactory.CreateInstance 5158
VCLCom.TComponentFactory.UpdateRegistry 5158

vfByRef enumeration member 238
vfVariant enumeration member 238
vmDefault enumeration member 6491
vmDisplay enumeration member 6491
vmInput enumeration member 6491
vmToggleOnAccess enumeration member 6491
vrEqual enumeration member 5106
vrGreaterThan enumeration member 5106
vrLessThan enumeration member 5106
vrNotEqual enumeration member 5106
vuBlankPassword enumeration member 6764
vuBlankUserName enumeration member 6764
vuUnknownPassword enumeration member 6764
vuUnknownUserName enumeration member 6764

W

WebAdapt 5159
WebAdapt namespace 5159
- Classes 5164
- Constants 5907
- Functions 5883
- Interfaces 5860
- Structs, Records, Enums 5887
- Types 5889
- Variables 5906

WebAdapt.AddDefaultActions 5883
WebAdapt.AddDefaultActions function 5883
WebAdapt.AddDefaultFields 5883
WebAdapt.AddDefaultFields function 5883
WebAdapt.EAdapterException 5166
WebAdapt.EAdapterException.CreateID 5167
WebAdapt.EAdapterException.ID 5167
WebAdapt.EAdapterFieldException 5167
WebAdapt.EAdapterFieldException.Create 5168
WebAdapt.EAdapterFieldException.FieldName 5169
WebAdapt.FindAdapterAction 5884
WebAdapt.FindAdapterAction function 5884
WebAdapt.FindAdapterField 5884
WebAdapt.FindAdapterField function 5884
WebAdapt.FindAdapterInParent 5884
WebAdapt.FindAdapterInParent function 5884
WebAdapt.IActionFieldValue 5890
WebAdapt.IActionFieldValue type 5890
WebAdapt.IActionRequest 5890
WebAdapt.IActionRequest type 5890
WebAdapt.IAdapterEditor 5860
WebAdapt.IAdapterEditor.CanAddActionClass 5861
WebAdapt.IAdapterEditor.CanAddFieldClass 5861
WebAdapt.IAdapterNotifyAdapterChange 5861
WebAdapt.IAdapterNotifyAdapterChange.NotifyAdapterChange 5862
WebAdapt.ICheckOrigValue 5862
WebAdapt.ICheckOrigValue.CheckOrigValue 5863
WebAdapt.ICheckValueChange 5863
WebAdapt.ICheckValueChange.CheckValueChange 5864
WebAdapt.IClearAdapterValues 5864
WebAdapt.IClearAdapterValues.ClearAdapterValues 5865
WebAdapt.ICreateActionRequestContext 5865
WebAdapt.ICreateActionRequestContext.CreateActionRequestContext 5866
WebAdapt.IEchoActionFieldValue 5866
WebAdapt.IEchoActionFieldValue.EchoActionFieldValue 5868
WebAdapt.IEchoActionFieldValue.GetEchoActionFieldValue 5867
WebAdapt.IEchoActionFieldValue.SetEchoActionFieldValue 5867
WebAdapt.IEchoAdapterFieldValues 5868
WebAdapt.IEchoAdapterFieldValues.EchoActionFieldValues 5869
WebAdapt.IEchoAdapterFieldValues.GetEchoActionFieldValues 5869
WebAdapt.IEchoAdapterFieldValues.SetEchoActionFieldValues 5869
WebAdapt.IExecuteValue 5869
WebAdapt.IExecuteValue.ExecuteValue 5870
WebAdapt.IFullyQualifyInputNames 5871
WebAdapt.IFullyQualifyInputNames.FullyQualifyInputNames 5871
WebAdapt.IGetAdapterErrorsList 5872
WebAdapt.IGetAdapterErrorsList.GetAdapterErrorsList 5872

WebAdapt.IGetAdapterModeNames 5873
WebAdapt.IGetAdapterModeNames.GetAdapterModeNames 5873
WebAdapt.IGetRecordKeys 5874
WebAdapt.IGetRecordKeys.GetRecordKeys 5874
WebAdapt.IIdentifyAdapter 5875
WebAdapt.ISetSubComponents 5875
WebAdapt.ISetSubComponents.SetActions 5876
WebAdapt.ISetSubComponents.SetData 5876
WebAdapt.ISetWebFieldValue 5876
WebAdapt.ISetWebFieldValue.SetValue 5877
WebAdapt.IUpdateValue 5877
WebAdapt.IUpdateValue.UpdateValue 5878
WebAdapt.IValidateValue 5878
WebAdapt.IValidateValue.ValidateValue 5879
WebAdapt.IWebActionsSubcomponent 5879
WebAdapt.IWebFieldsSubcomponent 5879
WebAdapt.IWebGetFieldPrevValue 5880
WebAdapt.IWebGetFieldPrevValue.GetNeedPrevValue 5881
WebAdapt.IWebGetFieldPrevValue.GetPrevValue 5881
WebAdapt.IWebGetFieldPrevValue.NeedPrevValue 5881
WebAdapt.IWebGetFieldPrevValue.PrevValue 5881
WebAdapt.MakeAdapterRequestIdentifier 5885
WebAdapt.MakeAdapterRequestIdentifier function 5885
WebAdapt.MakeValidAdapterFieldName 5885
WebAdapt.MakeValidAdapterFieldName function 5885
WebAdapt.NotifyPossibleHTMLChange 5906
WebAdapt.NotifyPossibleHTMLChange variable 5906
WebAdapt.RaiseFileUploadNotSupported 5885
WebAdapt.RaiseFileUploadNotSupported function 5885
WebAdapt.RaiseMultipleFilesException 5886
WebAdapt.RaiseMultipleFilesException function 5886
WebAdapt.RaiseMultipleValuesException 5886
WebAdapt.RaiseMultipleValuesException function 5886
WebAdapt.sActionRequestLong 5907
WebAdapt.sActionRequestLong constant 5907
WebAdapt.sCurrentPage 5907
WebAdapt.sCurrentPage constant 5907
WebAdapt.sGotoPageAction 5908
WebAdapt.sGotoPageAction constant 5908
WebAdapt.sImageRequest 5908
WebAdapt.sImageRequest constant 5908
WebAdapt.sLoginPassword 5908
WebAdapt.sLoginPassword constant 5908
WebAdapt.sLoginUserName 5909
WebAdapt.sLoginUserName constant 5909
WebAdapt.sNextPageAction 5909
WebAdapt.sNextPageAction constant 5909
WebAdapt.sPrevPageAction 5909
WebAdapt.sPrevPageAction constant 5909
WebAdapt.TAbstractFieldValuesAdapter 5169
WebAdapt.TAbstractFieldValuesAdapter.GetField 5175
WebAdapt.TAbstractFieldValuesAdapter.GetValueFieldClass 5175
WebAdapt.TAbstractFieldValuesAdapter.GetValueIndex 5176
WebAdapt.TAbstractFieldValuesAdapter.ImplCanAddFieldClass 5176
WebAdapt.TAbstractFieldValuesAdapter.ImplEndIterator 5177
WebAdapt.TAbstractFieldValuesAdapter.ImplGetField 5177
WebAdapt.TAbstractFieldValuesAdapter.ImplGetFieldsList 5177
WebAdapt.TAbstractFieldValuesAdapter.ImplGetValueIndex 5178
WebAdapt.TAbstractFieldValuesAdapter.ImplNextIteration 5178
WebAdapt.TAbstractFieldValuesAdapter.ImplStartIterator 5179
WebAdapt.TAbstractWebRequestFile 5890
WebAdapt.TAbstractWebRequestFile type 5890
WebAdapt.TActionExecuteEvent 5891
WebAdapt.TActionExecuteEvent type 5891
WebAdapt.TActionExecuteHandledEvent 5891
WebAdapt.TActionExecuteHandledEvent type 5891
WebAdapt.TActionGetEnabledEvent 5891
WebAdapt.TActionGetEnabledEvent type 5891
WebAdapt.TActionGetParamsEvent 5892
WebAdapt.TActionGetParamsEvent type 5892
WebAdapt.TActionRequestFieldsEvent 5892
WebAdapt.TActionRequestFieldsEvent type 5892
WebAdapt.TActionRequestFieldsHandledEvent 5892
WebAdapt.TActionRequestFieldsHandledEvent type 5892
WebAdapt.TAdapter 5179
WebAdapt.TAdapter.Actions 5184

WebAdapt.TAdapter.Data 5185
WebAdapt.TAdapter.ModifyAccess 5185
WebAdapt.TAdapter.OnAfterExecuteAction 5186
WebAdapt.TAdapter.OnAfterGetActionResponse 5186
WebAdapt.TAdapter.OnAfterUpdateFields 5187
WebAdapt.TAdapter.OnAfterValidateFields 5187
WebAdapt.TAdapter.OnBeforeExecuteAction 5187
WebAdapt.TAdapter.OnBeforeGetActionResponse 5188
WebAdapt.TAdapter.OnBeforeUpdateFields 5188
WebAdapt.TAdapter.OnBeforeValidateFields 5189
WebAdapt.TAdapter.OnGetActionParams 5189
WebAdapt.TAdapter.OnIterateRecords 5189
WebAdapt.TAdapter.ViewAccess 5185
WebAdapt.TAdapterAction 5190
WebAdapt.TAdapterAction.ActionName 5194
WebAdapt.TAdapterAction.DisplayLabel 5194
WebAdapt.TAdapterAction.ExecuteAccess 5195
WebAdapt.TAdapterAction.OnAfterExecute 5196
WebAdapt.TAdapterAction.OnAfterGetResponse 5196
WebAdapt.TAdapterAction.OnBeforeExecute 5197
WebAdapt.TAdapterAction.OnBeforeGetResponse 5197
WebAdapt.TAdapterAction.OnExecute 5197
WebAdapt.TAdapterAction.OnGetEnabled 5198
WebAdapt.TAdapterAction.OnGetParams 5198
WebAdapt.TAdapterAction.RedirectOptions 5195
WebAdapt.TAdapterActionExecuteEvent 5893
WebAdapt.TAdapterActionExecuteEvent type 5893
WebAdapt.TAdapterActionExecuteHandledEvent 5893
WebAdapt.TAdapterActionExecuteHandledEvent type 5893
WebAdapt.TAdapterActionGetParamsEvent 5893
WebAdapt.TAdapterActionGetParamsEvent type 5893
WebAdapt.TAdapterActions 5198
WebAdapt.TAdapterActions.AdapterCanAddClass 5201
WebAdapt.TAdapterActions.ImplCanAddClass 5201
WebAdapt.TAdapterActions.ImplGetWebComponentNodeName 5202
WebAdapt.TAdapterActions.SetWebParent 5202
WebAdapt.TAdapterActions.WebActions 5203
WebAdapt.TAdapterActionsClass 5894
WebAdapt.TAdapterActionsClass type 5894
WebAdapt.TAdapterApplicationField 5203
WebAdapt.TAdapterApplicationField.Adapter 5207
WebAdapt.TAdapterApplicationTitleField 5207
WebAdapt.TAdapterApplicationTitleField.GetDefaultFieldName 5211
WebAdapt.TAdapterApplicationTitleField.ImplGetValue 5211
WebAdapt.TAdapterBooleanField 5212
WebAdapt.TAdapterBooleanField.DisplayLabel 5217
WebAdapt.TAdapterBooleanField.DisplayWidth 5217
WebAdapt.TAdapterBooleanField.FieldName 5217
WebAdapt.TAdapterBooleanField.ModifyAccess 5218
WebAdapt.TAdapterBooleanField.OnGetValue 5219
WebAdapt.TAdapterBooleanField.OnUpdateValue 5219
WebAdapt.TAdapterBooleanField.OnValidateValue 5220
WebAdapt.TAdapterBooleanField.ReadOnly 5218
WebAdapt.TAdapterBooleanField.ValuesList 5218
WebAdapt.TAdapterBooleanField.ViewAccess 5219
WebAdapt.TAdapterDefaultActions 5220
WebAdapt.TAdapterDefaultActions.ClearDefaultComponents 5223
WebAdapt.TAdapterDefaultActions.DefaultWebComponents 5226
WebAdapt.TAdapterDefaultActions.Destroy 5222
WebAdapt.TAdapterDefaultActions.GetAddAllFieldsItem 5224
WebAdapt.TAdapterDefaultActions.GetAddFieldsDlgCaption 5224
WebAdapt.TAdapterDefaultActions.GetAddFieldsItem 5224
WebAdapt.TAdapterDefaultActions.GetDefaultWebComponents 5224
WebAdapt.TAdapterDefaultActions.GetNewFieldPrefix 5225
WebAdapt.TAdapterDefaultActions.GetVariablesInUse 5225
WebAdapt.TAdapterDefaultActions.SetDefaultWebComponents 5225
WebAdapt.TAdapterDefaultFields 5226
WebAdapt.TAdapterDefaultFields.ClearDefaultComponents 5229
WebAdapt.TAdapterDefaultFields.DefaultWebComponents 5232
WebAdapt.TAdapterDefaultFields.Destroy 5229
WebAdapt.TAdapterDefaultFields.GetAddAllFieldsItem 5230
WebAdapt.TAdapterDefaultFields.GetAddFieldsDlgCaption 5230

WebAdapt.TAdapterDefaultFields.GetAddFieldsItem 5230
WebAdapt.TAdapterDefaultFields.GetDefaultWebComponents 5230
WebAdapt.TAdapterDefaultFields.GetNewFieldPrefix 5231
WebAdapt.TAdapterDefaultFields.GetVariablesInUse 5231
WebAdapt.TAdapterDefaultFields.SetDefaultWebComponents 5231
WebAdapt.TAdapterEndUserDisplayNameField 5233
WebAdapt.TAdapterEndUserDisplayNameField.GetDefaultFieldName 5236
WebAdapt.TAdapterEndUserDisplayNameField.ImplGetValue 5236
WebAdapt.TAdapterEndUserField 5237
WebAdapt.TAdapterEndUserField.Adapter 5240
WebAdapt.TAdapterEndUserLoggedInField 5241
WebAdapt.TAdapterEndUserLoggedInField.GetDefaultFieldName 5244
WebAdapt.TAdapterEndUserLoggedInField.ImplGetValue 5244
WebAdapt.TAdapterError 5245
WebAdapt.TAdapterError.Create 5247
WebAdapt.TAdapterError.GetErrorText 5247
WebAdapt.TAdapterError.GetField 5247
WebAdapt.TAdapterError.GetID 5248
WebAdapt.TAdapterErrorsList 5248
WebAdapt.TAdapterErrorsList.AddError 5252
WebAdapt.TAdapterErrorsList.Create 5251
WebAdapt.TAdapterErrorsList.Destroy 5251
WebAdapt.TAdapterErrorsList.FieldNames 5255
WebAdapt.TAdapterErrorsList.IDs 5255
WebAdapt.TAdapterErrorsList.ImplClearErrors 5253
WebAdapt.TAdapterErrorsList.ImplDefineLabel 5253
WebAdapt.TAdapterErrorsList.ImplGetError 5253
WebAdapt.TAdapterErrorsList.ImplGetErrorCount 5254
WebAdapt.TAdapterErrorsList.ImplGetObjectErrors 5254
WebAdapt.TAdapterErrorsList.ImplHasObjectErrors 5254
WebAdapt.TAdapterErrorsList.Messages 5255
WebAdapt.TAdapterField 5256
WebAdapt.TAdapterField.DisplayLabel 5261
WebAdapt.TAdapterField.DisplayWidth 5261
WebAdapt.TAdapterField.FieldName 5261
WebAdapt.TAdapterField.ModifyAccess 5262
WebAdapt.TAdapterField.OnGetDisplayText 5264
WebAdapt.TAdapterField.OnGetValue 5264
WebAdapt.TAdapterField.OnUpdateValue 5264
WebAdapt.TAdapterField.OnValidateValue 5265
WebAdapt.TAdapterField.ReadOnly 5262
WebAdapt.TAdapterField.Required 5262
WebAdapt.TAdapterField.ValuesList 5263
WebAdapt.TAdapterField.ViewAccess 5263
WebAdapt.TAdapterFieldBooleanValueEvent 5894
WebAdapt.TAdapterFieldBooleanValueEvent type 5894
WebAdapt.TAdapterFieldFileEvent 5894
WebAdapt.TAdapterFieldFileEvent type 5894
WebAdapt.TAdapterFieldFileListEvent 5895
WebAdapt.TAdapterFieldFileListEvent type 5895
WebAdapt.TAdapterFieldGetBooleanEvent 5895
WebAdapt.TAdapterFieldGetBooleanEvent type 5895
WebAdapt.TAdapterFieldGetMultiValueEvent 5895
WebAdapt.TAdapterFieldGetMultiValueEvent type 5895
WebAdapt.TAdapterFieldGetMultiValueStringEvent 5896
WebAdapt.TAdapterFieldGetMultiValueStringEvent type 5896
WebAdapt.TAdapterFieldGetStringEvent 5896
WebAdapt.TAdapterFieldGetStringEvent type 5896
WebAdapt.TAdapterFieldGetValueCountEvent 5896
WebAdapt.TAdapterFieldGetValueCountEvent type 5896
WebAdapt.TAdapterFieldGetValueEvent 5897
WebAdapt.TAdapterFieldGetValueEvent type 5897
WebAdapt.TAdapterFieldHasValueEvent 5897
WebAdapt.TAdapterFieldHasValueEvent type 5897
WebAdapt.TAdapterFields 5265
WebAdapt.TAdapterFields.AdapterCanAddClass 5267
WebAdapt.TAdapterFields.ImplCanAddClass 5268
WebAdapt.TAdapterFields.ImplGetWebComponentNodeName 5269
WebAdapt.TAdapterFields.SetWebParent 5269
WebAdapt.TAdapterFields.WebFields 5270
WebAdapt.TAdapterFieldsClass 5897
WebAdapt.TAdapterFieldsClass type 5897
WebAdapt.TAdapterFieldStringEvent 5898
WebAdapt.TAdapterFieldStringEvent type 5898

WebAdapt.TAdapterFieldStringsEvent 5898
WebAdapt.TAdapterFieldStringsEvent type 5898
WebAdapt.TAdapterFieldValidateValueEvent 5898
WebAdapt.TAdapterFieldValidateValueEvent type 5898
WebAdapt.TAdapterFieldValueEvent 5899
WebAdapt.TAdapterFieldValueEvent type 5899
WebAdapt.TAdapterFileField 5270
WebAdapt.TAdapterFileField.DisplayLabel 5276
WebAdapt.TAdapterFileField.DisplayWidth 5276
WebAdapt.TAdapterFileField.FieldName 5277
WebAdapt.TAdapterFileField.ModifyAccess 5277
WebAdapt.TAdapterFileField.OnGetFileCount 5278
WebAdapt.TAdapterFileField.OnGetFileNames 5279
WebAdapt.TAdapterFileField.OnUploadFiles 5279
WebAdapt.TAdapterFileField.OnValidateFiles 5279
WebAdapt.TAdapterFileField.ReadOnly 5277
WebAdapt.TAdapterFileField.Required 5278
WebAdapt.TAdapterFileField.ViewAccess 5278
WebAdapt.TAdapterGotoPageAction 5280
WebAdapt.TAdapterGotoPageAction.ExecuteAccess 5285
WebAdapt.TAdapterGotoPageAction.OnAfterExecute 5286
WebAdapt.TAdapterGotoPageAction.OnAfterGetResponse 5286
WebAdapt.TAdapterGotoPageAction.OnBeforeExecute 5286
WebAdapt.TAdapterGotoPageAction.OnBeforeGetResponse 5287
WebAdapt.TAdapterGotoPageAction.OnGetPageName 5287
WebAdapt.TAdapterGotoPageAction.OnGetParams 5288
WebAdapt.TAdapterGotoPageAction.RedirectOptions 5285
WebAdapt.TAdapterHiddenField 5288
WebAdapt.TAdapterHiddenField.Create 5290
WebAdapt.TAdapterHiddenField.GetName 5290
WebAdapt.TAdapterHiddenField.GetValue 5290
WebAdapt.TAdapterHiddenFieldOption 5887
WebAdapt.TAdapterHiddenFieldOption enumeration 5887
WebAdapt.TAdapterHiddenFieldOptions 5899
WebAdapt.TAdapterHiddenFieldOptions type 5899
WebAdapt.TAdapterHiddenFieldsList 5290
WebAdapt.TAdapterHiddenFieldsList.Add 5294
WebAdapt.TAdapterHiddenFieldsList.Clear 5294
WebAdapt.TAdapterHiddenFieldsList.Create 5293
WebAdapt.TAdapterHiddenFieldsList.Destroy 5293
WebAdapt.TAdapterHiddenFieldsList.EndIterator 5295
WebAdapt.TAdapterHiddenFieldsList.FieldCount 5296
WebAdapt.TAdapterHiddenFieldsList.Fields 5296
WebAdapt.TAdapterHiddenFieldsList.GetCurrentIntf 5295
WebAdapt.TAdapterHiddenFieldsList.GetField 5295
WebAdapt.TAdapterHiddenFieldsList.GetFieldCount 5295
WebAdapt.TAdapterHiddenFieldsList.NextIteration 5295
WebAdapt.TAdapterHiddenFieldsList.StartIterator 5296
WebAdapt.TAdapterImageField 5297
WebAdapt.TAdapterImageField.DisplayLabel 5302
WebAdapt.TAdapterImageField.DisplayWidth 5303
WebAdapt.TAdapterImageField.FieldName 5303
WebAdapt.TAdapterImageField.HREF 5303
WebAdapt.TAdapterImageField.ModifyAccess 5304
WebAdapt.TAdapterImageField.OnGetHREF 5306
WebAdapt.TAdapterImageField.OnGetImage 5306
WebAdapt.TAdapterImageField.OnGetImageName 5306
WebAdapt.TAdapterImageField.OnGetParams 5307
WebAdapt.TAdapterImageField.OnUploadFile 5307
WebAdapt.TAdapterImageField.OnValidateFile 5307
WebAdapt.TAdapterImageField.ReadOnly 5304
WebAdapt.TAdapterImageField.Required 5304
WebAdapt.TAdapterImageField.ValuesList 5305
WebAdapt.TAdapterImageField.ViewAccess 5305
WebAdapt.TAdapterLoginField 5308
WebAdapt.TAdapterLoginField.Adapter 5311
WebAdapt.TAdapterMemoField 5312
WebAdapt.TAdapterMemoField.DisplayLabel 5317
WebAdapt.TAdapterMemoField.DisplayWidth 5317
WebAdapt.TAdapterMemoField.FieldName 5317
WebAdapt.TAdapterMemoField.ModifyAccess 5318
WebAdapt.TAdapterMemoField.OnGetValue 5319
WebAdapt.TAdapterMemoField.OnUpdateValue 5319
WebAdapt.TAdapterMemoField.OnUploadFile 5320
WebAdapt.TAdapterMemoField.ReadOnly 5318
WebAdapt.TAdapterMemoField.Required 5318
WebAdapt.TAdapterMemoField.ViewAccess 5319

WebAdapt.TAdapterMultiValueField 5320
WebAdapt.TAdapterMultiValueField.DisplayLabel 5326
WebAdapt.TAdapterMultiValueField.DisplayWidth 5326
WebAdapt.TAdapterMultiValueField.FieldName 5327
WebAdapt.TAdapterMultiValueField.ModifyAccess 5327
WebAdapt.TAdapterMultiValueField.OnGetValueCount 5329
WebAdapt.TAdapterMultiValueField.OnGetValues 5329
WebAdapt.TAdapterMultiValueField.OnHasValue 5330
WebAdapt.TAdapterMultiValueField.OnUpdateValues 5330
WebAdapt.TAdapterMultiValueField.OnValidateValues 5330
WebAdapt.TAdapterMultiValueField.ReadOnly 5327
WebAdapt.TAdapterMultiValueField.Required 5328
WebAdapt.TAdapterMultiValueField.ValuesList 5328
WebAdapt.TAdapterMultiValueField.ViewAccess 5328
WebAdapt.TAdapterNamedDisplayField 5331
WebAdapt.TAdapterNamedDisplayField.Create 5334
WebAdapt.TAdapterNamedDisplayField.DisplayLabel 5338
WebAdapt.TAdapterNamedDisplayField.DisplayWidth 5338
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyle 5334
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyleType 5335
WebAdapt.TAdapterNamedDisplayField.GetInputStyle 5335
WebAdapt.TAdapterNamedDisplayField.GetInputStyleType 5335
WebAdapt.TAdapterNamedDisplayField.GetViewMode 5336
WebAdapt.TAdapterNamedDisplayField.GetViewModeType 5336
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayLabel 5337
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayWidth 5337
WebAdapt.TAdapterNamedDisplayField.ImplGetRequired 5337
WebAdapt.TAdapterNamedDisplayField.ReadOnly 5339
WebAdapt.TAdapterNamedDisplayField.Required 5339
WebAdapt.TAdapterNamedDisplayField.SetDisplayLabel 5337
WebAdapt.TAdapterNamedField 5339
WebAdapt.TAdapterNamedField.ActionOrigValue 5355
WebAdapt.TAdapterNamedField.ActionValue 5356
WebAdapt.TAdapterNamedField.Adapter 5356
WebAdapt.TAdapterNamedField.AsFormatted 5343
WebAdapt.TAdapterNamedField.CheckModifyAccess 5343
WebAdapt.TAdapterNamedField.CheckViewAccess 5344
WebAdapt.TAdapterNamedField.EchoActionFieldValue 5356
WebAdapt.TAdapterNamedField.FieldName 5357
WebAdapt.TAdapterNamedField.FullyQualifyInputName 5344
WebAdapt.TAdapterNamedField.GetAdapter 5344
WebAdapt.TAdapterNamedField.GetDefaultDisplayLabel 5345
WebAdapt.TAdapterNamedField.GetDefaultFieldName 5345
WebAdapt.TAdapterNamedField.GetDisplayLabel 5345
WebAdapt.TAdapterNamedField.GetDisplayWidth 5346
WebAdapt.TAdapterNamedField.GetEchoActionFieldValue 5346
WebAdapt.TAdapterNamedField.GetInputName 5346
WebAdapt.TAdapterNamedField.GetMaxLength 5347
WebAdapt.TAdapterNamedField.GetRequired 5347
WebAdapt.TAdapterNamedField.GetValue 5347
WebAdapt.TAdapterNamedField.GetVariableName 5348
WebAdapt.TAdapterNamedField.GetVisible 5348
WebAdapt.TAdapterNamedField.GetWebSnapObjClass 5348
WebAdapt.TAdapterNamedField.HasModifyAccess 5349
WebAdapt.TAdapterNamedField.HasViewAccess 5349
WebAdapt.TAdapterNamedField.ImplAsFormatted 5349
WebAdapt.TAdapterNamedField.ImplGetDisplayLabel 5349
WebAdapt.TAdapterNamedField.ImplGetDisplayWidth 5350
WebAdapt.TAdapterNamedField.ImplGetEchoActionFieldValue 5350
WebAdapt.TAdapterNamedField.ImplGetInputName 5350
WebAdapt.TAdapterNamedField.ImplGetMaxLength 5351
WebAdapt.TAdapterNamedField.ImplGetRequired 5351
WebAdapt.TAdapterNamedField.ImplGetValue 5351
WebAdapt.TAdapterNamedField.ImplGetVariableName 5352
WebAdapt.TAdapterNamedField.ImplGetVisible 5352
WebAdapt.TAdapterNamedField.ImplHasModifyAccess 5353
WebAdapt.TAdapterNamedField.ImplHasViewAccess 5353
WebAdapt.TAdapterNamedField.ImplSetEchoActionFieldValue 5353
WebAdapt.TAdapterNamedField.ImplSetFieldName 5354
WebAdapt.TAdapterNamedField.ModifyAccess 5357
WebAdapt.TAdapterNamedField.ParentChanged 5354
WebAdapt.TAdapterNamedField.SetEchoActionFieldValue 5354
WebAdapt.TAdapterNamedField.SetFieldName 5355
WebAdapt.TAdapterNamedField.Value 5357

WebAdapt.TAdapterNamedField.ViewAccess 5358
WebAdapt.TAdapterNextPageAction 5358
WebAdapt.TAdapterNextPageAction.DisplayLabel 5364
WebAdapt.TAdapterNextPageAction.ExecuteAccess 5364
WebAdapt.TAdapterNextPageAction.GetDefaultActionName 5363
WebAdapt.TAdapterNextPageAction.GetDefaultDisplayLabel 5363
WebAdapt.TAdapterNextPageAction.GetNewPage 5363
WebAdapt.TAdapterNextPageAction.OnAfterExecute 5365
WebAdapt.TAdapterNextPageAction.OnAfterGetResponse 5366
WebAdapt.TAdapterNextPageAction.OnBeforeExecute 5366
WebAdapt.TAdapterNextPageAction.OnBeforeGetResponse 5366
WebAdapt.TAdapterNextPageAction.OnGetParams 5367
WebAdapt.TAdapterNextPageAction.RedirectOptions 5365
WebAdapt.TAdapterNextPageField 5367
WebAdapt.TAdapterNextPageField.Destroy 5371
WebAdapt.TAdapterNextPageField.DisplayLabel 5374
WebAdapt.TAdapterNextPageField.DisplayWidth 5374
WebAdapt.TAdapterNextPageField.FieldName 5375
WebAdapt.TAdapterNextPageField.GetAdapterValuesList 5372
WebAdapt.TAdapterNextPageField.GetInputStyleType 5372
WebAdapt.TAdapterNextPageField.ImplGetEchoActionFieldValue 5372
WebAdapt.TAdapterNextPageField.ImplGetValue 5373
WebAdapt.TAdapterNextPageField.ImplSetEchoActionFieldValue 5373
WebAdapt.TAdapterNextPageField.UpdateValue 5374
WebAdapt.TAdapterObjectErrorsList 5375
WebAdapt.TAdapterObjectErrorsList.Create 5377
WebAdapt.TAdapterObjectErrorsList.ImplDefineLabel 5378
WebAdapt.TAdapterObjectErrorsList.ImplGetError 5378
WebAdapt.TAdapterObjectErrorsList.ImplGetErrorCount 5378
WebAdapt.TAdapterPasswordField 5379
WebAdapt.TAdapterPasswordField.DisplayLabel 5383
WebAdapt.TAdapterPasswordField.DisplayWidth 5383
WebAdapt.TAdapterPasswordField.FieldName 5384
WebAdapt.TAdapterPasswordField.GetInputStyleType 5382
WebAdapt.TAdapterPasswordField.ImplGetValue 5383
WebAdapt.TAdapterPrevPageAction 5384
WebAdapt.TAdapterPrevPageAction.DisplayLabel 5390
WebAdapt.TAdapterPrevPageAction.ExecuteAccess 5391
WebAdapt.TAdapterPrevPageAction.GetDefaultActionName 5389
WebAdapt.TAdapterPrevPageAction.GetDefaultDisplayLabel 5389
WebAdapt.TAdapterPrevPageAction.GetNewPage 5390
WebAdapt.TAdapterPrevPageAction.OnAfterExecute 5392
WebAdapt.TAdapterPrevPageAction.OnAfterGetResponse 5392
WebAdapt.TAdapterPrevPageAction.OnBeforeExecute 5392
WebAdapt.TAdapterPrevPageAction.OnBeforeGetResponse 5393
WebAdapt.TAdapterPrevPageAction.OnGetParams 5393
WebAdapt.TAdapterPrevPageAction.RedirectOptions 5391
WebAdapt.TAdapterSubComponent 5394
WebAdapt.TAdapterSubComponent.Adapter 5406
WebAdapt.TAdapterSubComponent.CanAddClass 5396
WebAdapt.TAdapterSubComponent.EndIterator 5397
WebAdapt.TAdapterSubComponent.FindNamedVariable 5397
WebAdapt.TAdapterSubComponent.FindVariable 5398
WebAdapt.TAdapterSubComponent.FindVariableContainer 5398
WebAdapt.TAdapterSubComponent.GetCurrentObject 5399
WebAdapt.TAdapterSubComponent.GetDefaultWebComponents 5399
WebAdapt.TAdapterSubComponent.GetParentComponent 5400
WebAdapt.TAdapterSubComponent.GetVariable 5400
WebAdapt.TAdapterSubComponent.GetVariableCount 5401
WebAdapt.TAdapterSubComponent.GetVariablesInUse 5401
WebAdapt.TAdapterSubComponent.GetWebComponentNodeName 5401
WebAdapt.TAdapterSubComponent.HasParent 5402
WebAdapt.TAdapterSubComponent.ImplCanAddClass 5402
WebAdapt.TAdapterSubComponent.ImplGetWebComponentNodeName 5403
WebAdapt.TAdapterSubComponent.NextIteration 5403
WebAdapt.TAdapterSubComponent.NotifyAdapterChange 5404
WebAdapt.TAdapterSubComponent.ReadState 5404
WebAdapt.TAdapterSubComponent.SetDefaultWebComponent

s
5404
WebAdapt.TAdapterSubComponent.SetParentComponent 5405
WebAdapt.TAdapterSubComponent.SetWebParent 5405
WebAdapt.TAdapterSubComponent.StartIterator 5406
WebAdapt.TAdapterSubComponent.VariablesInUse 5407
WebAdapt.TAdapterUpdateField 5407
WebAdapt.TAdapterUpdateField.CheckValueChange 5412
WebAdapt.TAdapterUpdateField.ConvertValue 5413
WebAdapt.TAdapterUpdateField.ConvertValueCount 5413
WebAdapt.TAdapterUpdateField.ConvertValues 5413
WebAdapt.TAdapterUpdateField.Create 5411
WebAdapt.TAdapterUpdateField.DoOnValidateValue 5414
WebAdapt.TAdapterUpdateField.GetAdapterValuesList 5414
WebAdapt.TAdapterUpdateField.GetInputStyleType 5414
WebAdapt.TAdapterUpdateField.GetValueCount 5415
WebAdapt.TAdapterUpdateField.GetValues 5415
WebAdapt.TAdapterUpdateField.ImplAsFormatted 5415
WebAdapt.TAdapterUpdateField.ImplCheckValueChange 5416
WebAdapt.TAdapterUpdateField.ImplGetDisplayLabel 5416
WebAdapt.TAdapterUpdateField.ImplGetDisplayWidth 5416
WebAdapt.TAdapterUpdateField.ImplGetEchoActionFieldValue 5417
WebAdapt.TAdapterUpdateField.ImplGetVariableName 5417
WebAdapt.TAdapterUpdateField.ImplSetEchoActionFieldValue 5417
WebAdapt.TAdapterUpdateField.ImplSetFieldName 5417
WebAdapt.TAdapterUpdateField.ImplUpdateValue 5418
WebAdapt.TAdapterUpdateField.ImplValidateValue 5418
WebAdapt.TAdapterUpdateField.IsMultiValueField 5418
WebAdapt.TAdapterUpdateField.Notification 5419
WebAdapt.TAdapterUpdateField.OnGetDisplayText 5421
WebAdapt.TAdapterUpdateField.OnValidateValue 5421
WebAdapt.TAdapterUpdateField.UpdateValue 5419
WebAdapt.TAdapterUpdateField.ValidateValue 5419
WebAdapt.TAdapterUpdateField.Value 5420
WebAdapt.TAdapterUpdateField.ValuesList 5420
WebAdapt.TAdapterUserNameField 5422
WebAdapt.TAdapterUserNameField.DisplayLabel 5426
WebAdapt.TAdapterUserNameField.DisplayWidth 5426
WebAdapt.TAdapterUserNameField.FieldName 5426
WebAdapt.TAdapterUserNameField.ImplGetValue 5425
WebAdapt.TApplicationAdapter 5427
WebAdapt.TApplicationAdapter.Actions 5432
WebAdapt.TApplicationAdapter.ApplicationTitle 5432
WebAdapt.TApplicationAdapter.Data 5433
WebAdapt.TApplicationAdapter.OnAfterExecuteAction 5433
WebAdapt.TApplicationAdapter.OnAfterGetActionResponse 5433
WebAdapt.TApplicationAdapter.OnBeforeExecuteAction 5434
WebAdapt.TApplicationAdapter.OnBeforeGetActionResponse 5434
WebAdapt.TApplicationAdapter.OnGetActionParams 5435
WebAdapt.TBaseAdapterAction 5435
WebAdapt.TBaseAdapterAction.ActionName 5462
WebAdapt.TBaseAdapterAction.Adapter 5462
WebAdapt.TBaseAdapterAction.CheckExecuteAccess 5440
WebAdapt.TBaseAdapterAction.CreateRequestContext 5440
WebAdapt.TBaseAdapterAction.DisplayLabel 5462
WebAdapt.TBaseAdapterAction.DoAfterExecuteActionRequest 5441
WebAdapt.TBaseAdapterAction.DoAfterGetExecuteActionResponse 5441
WebAdapt.TBaseAdapterAction.DoBeforeExecuteActionRequest 5442
WebAdapt.TBaseAdapterAction.DoBeforeGetExecuteActionResponse 5442
WebAdapt.TBaseAdapterAction.EndIterator 5443
WebAdapt.TBaseAdapterAction.Errors 5463
WebAdapt.TBaseAdapterAction.ExecuteAccess 5463
WebAdapt.TBaseAdapterAction.ExecuteActionRequest 5443
WebAdapt.TBaseAdapterAction.GetActionName 5444
WebAdapt.TBaseAdapterAction.GetAdapter 5444
WebAdapt.TBaseAdapterAction.GetAdapterItemRequestParams 5444
WebAdapt.TBaseAdapterAction.GetDefaultActionName 5445
WebAdapt.TBaseAdapterAction.GetDefaultDisplayLabel 5445
WebAdapt.TBaseAdapterAction.GetDisplayLabel 5445
WebAdapt.TBaseAdapterAction.GetDisplayStyle 5445

WebAdapt.TBaseAdapterAction.GetDisplayStyleType 5446
WebAdapt.TBaseAdapterAction.GetErrors 5446
WebAdapt.TBaseAdapterAction.GetExecuteActionResponse 5446
WebAdapt.TBaseAdapterAction.GetFailureOptions 5447
WebAdapt.TBaseAdapterAction.GetInputStyle 5447
WebAdapt.TBaseAdapterAction.GetIsAdapterActionList 5448
WebAdapt.TBaseAdapterAction.GetItemCount 5448
WebAdapt.TBaseAdapterAction.GetIteratorIndex 5448
WebAdapt.TBaseAdapterAction.GetSuccessOptions 5449
WebAdapt.TBaseAdapterAction.GetVariableName 5449
WebAdapt.TBaseAdapterAction.GetViewMode 5450
WebAdapt.TBaseAdapterAction.GetViewModeType 5450
WebAdapt.TBaseAdapterAction.GetVisible 5450
WebAdapt.TBaseAdapterAction.GetWebSnapObjClass 5451
WebAdapt.TBaseAdapterAction.HandleRequest 5451
WebAdapt.TBaseAdapterAction.HasExecuteAccess 5451
WebAdapt.TBaseAdapterAction.ImplCreateRequestContext 5452
WebAdapt.TBaseAdapterAction.ImplEndIterator 5452
WebAdapt.TBaseAdapterAction.ImplExecuteActionRequest 5452
WebAdapt.TBaseAdapterAction.ImplGetActionName 5453
WebAdapt.TBaseAdapterAction.ImplGetAdapterItemRequestParams 5453
WebAdapt.TBaseAdapterAction.ImplGetDisplayLabel 5453
WebAdapt.TBaseAdapterAction.ImplGetExecuteActionResponse 5454
WebAdapt.TBaseAdapterAction.ImplGetFailureOptions 5454
WebAdapt.TBaseAdapterAction.ImplGetIsAdapterActionList 5454
WebAdapt.TBaseAdapterAction.ImplGetIteratorIndex 5455
WebAdapt.TBaseAdapterAction.ImplGetSuccessOptions 5455
WebAdapt.TBaseAdapterAction.ImplGetVariableName 5455
WebAdapt.TBaseAdapterAction.ImplGetVisible 5456
WebAdapt.TBaseAdapterAction.ImplHandleRequest 5456
WebAdapt.TBaseAdapterAction.ImplHasExecuteAccess 5456
WebAdapt.TBaseAdapterAction.ImplInIterator 5457
WebAdapt.TBaseAdapterAction.ImplIsDefaultAction 5457
WebAdapt.TBaseAdapterAction.ImplNextIteration 5458
WebAdapt.TBaseAdapterAction.ImplStartIterator 5458
WebAdapt.TBaseAdapterAction.ImplWebEnabled 5458
WebAdapt.TBaseAdapterAction.InIterator 5459
WebAdapt.TBaseAdapterAction.IsDefaultAction 5459
WebAdapt.TBaseAdapterAction.ItemCount 5463
WebAdapt.TBaseAdapterAction.ItemIndex 5464
WebAdapt.TBaseAdapterAction.NextIteration 5460
WebAdapt.TBaseAdapterAction.OnAfterExecute 5465
WebAdapt.TBaseAdapterAction.OnAfterGetResponse 5465
WebAdapt.TBaseAdapterAction.OnBeforeExecute 5465
WebAdapt.TBaseAdapterAction.OnBeforeGetResponse 5466
WebAdapt.TBaseAdapterAction.OnGetParams 5466
WebAdapt.TBaseAdapterAction.ParentChanged 5460
WebAdapt.TBaseAdapterAction.RedirectOptions 5464
WebAdapt.TBaseAdapterAction.SetDisplayLabel 5460
WebAdapt.TBaseAdapterAction.StartIterator 5461
WebAdapt.TBaseAdapterAction.WebEnabled 5461
WebAdapt.TBaseAdapterErrorsList 5467
WebAdapt.TBaseAdapterErrorsList.AddError 5470
WebAdapt.TBaseAdapterErrorsList.ClearErrors 5470
WebAdapt.TBaseAdapterErrorsList.DefineLabel 5470
WebAdapt.TBaseAdapterErrorsList.EndIterator 5471
WebAdapt.TBaseAdapterErrorsList.ErrorCount 5474
WebAdapt.TBaseAdapterErrorsList.Errors 5475
WebAdapt.TBaseAdapterErrorsList.GetCurrentIntf 5471
WebAdapt.TBaseAdapterErrorsList.GetError 5471
WebAdapt.TBaseAdapterErrorsList.GetErrorCount 5471
WebAdapt.TBaseAdapterErrorsList.GetObjectErrors 5472
WebAdapt.TBaseAdapterErrorsList.HasObjectErrors 5472
WebAdapt.TBaseAdapterErrorsList.ImplClearErrors 5472
WebAdapt.TBaseAdapterErrorsList.ImplDefineLabel 5472
WebAdapt.TBaseAdapterErrorsList.ImplGetError 5473
WebAdapt.TBaseAdapterErrorsList.ImplGetErrorCount 5473
WebAdapt.TBaseAdapterErrorsList.ImplGetObjectErrors 5473
WebAdapt.TBaseAdapterErrorsList.ImplHasObjectErrors 5473
WebAdapt.TBaseAdapterErrorsList.NextIteration 5474
WebAdapt.TBaseAdapterErrorsList.StartIterator 5474
WebAdapt.TBaseAdapterMultiValueField 5475
WebAdapt.TBaseAdapterMultiValueField.Create 5480

WebAdapt.TBaseAdapterMultiValueField.Destroy 5480
WebAdapt.TBaseAdapterMultiValueField.EndValueIterator 5481
WebAdapt.TBaseAdapterMultiValueField.GetCurrentValue 5481
WebAdapt.TBaseAdapterMultiValueField.GetFieldValuesAdapte
r
5482
WebAdapt.TBaseAdapterMultiValueField.GetValueCount 5482
WebAdapt.TBaseAdapterMultiValueField.GetValues 5482
WebAdapt.TBaseAdapterMultiValueField.HasValue 5483
WebAdapt.TBaseAdapterMultiValueField.ImplGetCurrentValue
5483
WebAdapt.TBaseAdapterMultiValueField.ImplGetFieldValuesAd
apter
5483
WebAdapt.TBaseAdapterMultiValueField.ImplHasValue 5484
WebAdapt.TBaseAdapterMultiValueField.NextValueIteration
5484
WebAdapt.TBaseAdapterMultiValueField.StartValueIterator
5484
WebAdapt.TBaseEndUserAdapter 5485
WebAdapt.TBaseEndUserAdapter.CanExecuteLoginForm 5491
WebAdapt.TBaseEndUserAdapter.CanExecuteLogout 5491
WebAdapt.TBaseEndUserAdapter.CanProduceLoginPage 5492
WebAdapt.TBaseEndUserAdapter.DoOnHasRights 5492
WebAdapt.TBaseEndUserAdapter.ExecuteLogin 5492
WebAdapt.TBaseEndUserAdapter.ExecuteLoginForm 5493
WebAdapt.TBaseEndUserAdapter.ExecuteLogout 5493
WebAdapt.TBaseEndUserAdapter.GetDisplayName 5493
WebAdapt.TBaseEndUserAdapter.GetDisplayNameValue 5494
WebAdapt.TBaseEndUserAdapter.GetEndUserAdapter 5494
WebAdapt.TBaseEndUserAdapter.GetLoggedIn 5495
WebAdapt.TBaseEndUserAdapter.GetLoggedInValue 5495
WebAdapt.TBaseEndUserAdapter.GetLoginFormAction 5495
WebAdapt.TBaseEndUserAdapter.GetLogoutAction 5496
WebAdapt.TBaseEndUserAdapter.GetUserID 5496
WebAdapt.TBaseEndUserAdapter.HasExecuteRights 5496
WebAdapt.TBaseEndUserAdapter.HasModifyRights 5497
WebAdapt.TBaseEndUserAdapter.HasViewRights 5497
WebAdapt.TBaseEndUserAdapter.ImplCanAddActionClass
5497
WebAdapt.TBaseEndUserAdapter.ImplCanAddFieldClass 5498
WebAdapt.TBaseEndUserAdapter.ImplGetActionsList 5498
WebAdapt.TBaseEndUserAdapter.ImplGetDisplayName 5498
WebAdapt.TBaseEndUserAdapter.ImplGetEndUserAdapter
5499
WebAdapt.TBaseEndUserAdapter.ImplGetFieldsList 5499
WebAdapt.TBaseEndUserAdapter.ImplGetLoggedIn 5499
WebAdapt.TBaseEndUserAdapter.ImplGetLoginFormAction
5499
WebAdapt.TBaseEndUserAdapter.ImplGetLogoutAction 5500
WebAdapt.TBaseEndUserAdapter.ImplHasExecuteRights 5500
WebAdapt.TBaseEndUserAdapter.ImplHasModifyRights 5500
WebAdapt.TBaseEndUserAdapter.ImplHasViewRights 5501
WebAdapt.TBaseEndUserAdapter.ImplLogIn 5501
WebAdapt.TBaseEndUserAdapter.ImplProduceLoginPage 5501
WebAdapt.TBaseEndUserAdapter.LogIn 5502
WebAdapt.TBaseEndUserAdapter.LoginPage 5503
WebAdapt.TBaseEndUserAdapter.OnHasRights 5503
WebAdapt.TBaseEndUserAdapter.OnProduceLoginPage 5504
WebAdapt.TBaseEndUserAdapter.ProduceLoginPage 5502
WebAdapt.TBaseEndUserAdapter.UserList 5503
WebAdapt.TBaseValuesListAdapter 5504
WebAdapt.TBaseValuesListAdapter.GetListImage 5510
WebAdapt.TBaseValuesListAdapter.GetListImageOfValue 5510
WebAdapt.TBaseValuesListAdapter.GetListName 5511
WebAdapt.TBaseValuesListAdapter.GetListNameOfValue 5511
WebAdapt.TBaseValuesListAdapter.GetListValue 5511
WebAdapt.TBaseValuesListAdapter.GetNameFieldClass 5512
WebAdapt.TBaseValuesListAdapter.GetValueFieldClass 5512
WebAdapt.TBaseValuesListAdapter.GetWebSnapObjClass
5512
WebAdapt.TBaseValuesListAdapter.ImplCanAddFieldClass
5512
WebAdapt.TBaseValuesListAdapter.ImplGetFieldsList 5513
WebAdapt.TBaseValuesListAdapter.ImplGetListImage 5513
WebAdapt.TBaseValuesListAdapter.ImplGetListImageOfValue
5513
WebAdapt.TBaseValuesListAdapter.ImplGetListName 5514
WebAdapt.TBaseValuesListAdapter.ImplGetListNameOfValue
5514
WebAdapt.TBaseValuesListAdapter.ImplGetListValue 5514
WebAdapt.TBaseValuesListField 5515
WebAdapt.TBaseValuesListField.Adapter 5518

WebAdapt.TBaseValuesListField.GetAdapter 5518
WebAdapt.TCustomAdapter 5519
WebAdapt.TCustomAdapter.Actions 5558
WebAdapt.TCustomAdapter.AddAdapterModeHiddenField 5526
WebAdapt.TCustomAdapter.AddOrigValueHiddenFields 5526
WebAdapt.TCustomAdapter.AddRecordKeysHiddenFields 5526
WebAdapt.TCustomAdapter.CanAddActionClass 5527
WebAdapt.TCustomAdapter.CanAddFieldClass 5527
WebAdapt.TCustomAdapter.CheckModifyAccess 5527
WebAdapt.TCustomAdapter.ClearAdapterValues 5528
WebAdapt.TCustomAdapter.ClearErrors 5528
WebAdapt.TCustomAdapter.Create 5523
WebAdapt.TCustomAdapter.CreateActionRequestContext 5528
WebAdapt.TCustomAdapter.CreateErrorsList 5529
WebAdapt.TCustomAdapter.Data 5558
WebAdapt.TCustomAdapter.Destroy 5523
WebAdapt.TCustomAdapter.EchoActionFieldValues 5559
WebAdapt.TCustomAdapter.EncodeActionParams 5529
WebAdapt.TCustomAdapter.EndIterator 5529
WebAdapt.TCustomAdapter.Errors 5559
WebAdapt.TCustomAdapter.ExecuteAccess 5560
WebAdapt.TCustomAdapter.ExtractRequestParams 5530
WebAdapt.TCustomAdapter.FindAction 5530
WebAdapt.TCustomAdapter.FindField 5530
WebAdapt.TCustomAdapter.FindNamedAction 5531
WebAdapt.TCustomAdapter.FindNamedField 5531
WebAdapt.TCustomAdapter.FindNamedVariable 5531
WebAdapt.TCustomAdapter.FindVariable 5532
WebAdapt.TCustomAdapter.FindVariableContainer 5532
WebAdapt.TCustomAdapter.GetActionsClass 5533
WebAdapt.TCustomAdapter.GetActionsList 5533
WebAdapt.TCustomAdapter.GetAdapterActions 5533
WebAdapt.TCustomAdapter.GetAdapterErrors 5534
WebAdapt.TCustomAdapter.GetAdapterErrorsList 5534
WebAdapt.TCustomAdapter.GetAdapterFields 5534
WebAdapt.TCustomAdapter.GetAdapterHiddenFields 5534
WebAdapt.TCustomAdapter.GetAdapterHiddenRecordFields 5535
WebAdapt.TCustomAdapter.GetChildren 5535
WebAdapt.TCustomAdapter.GetDataClass 5535
WebAdapt.TCustomAdapter.GetEchoActionFieldValues 5536
WebAdapt.TCustomAdapter.GetFieldsList 5536
WebAdapt.TCustomAdapter.GetHiddenFieldOptions 5536
WebAdapt.TCustomAdapter.GetIteratorIndex 5537
WebAdapt.TCustomAdapter.GetVariable 5537
WebAdapt.TCustomAdapter.GetVariableCount 5538
WebAdapt.TCustomAdapter.GetVariableName 5538
WebAdapt.TCustomAdapter.GetVisibleActions 5538
WebAdapt.TCustomAdapter.GetVisibleFields 5538
WebAdapt.TCustomAdapter.GetWebSnapObjClass 5539
WebAdapt.TCustomAdapter.HasExecuteAccess 5539
WebAdapt.TCustomAdapter.HasModifyAccess 5539
WebAdapt.TCustomAdapter.HasViewAccess 5539
WebAdapt.TCustomAdapter.ImplCanAddActionClass 5540
WebAdapt.TCustomAdapter.ImplCanAddFieldClass 5540
WebAdapt.TCustomAdapter.ImplClearAdapterValues 5540
WebAdapt.TCustomAdapter.ImplEndIterator 5541
WebAdapt.TCustomAdapter.ImplFindVariable 5541
WebAdapt.TCustomAdapter.ImplFindVariableContainer 5541
WebAdapt.TCustomAdapter.ImplGetActionsList 5542
WebAdapt.TCustomAdapter.ImplGetEchoActionFieldValues 5542
WebAdapt.TCustomAdapter.ImplGetFieldsList 5542
WebAdapt.TCustomAdapter.ImplGetIteratorIndex 5543
WebAdapt.TCustomAdapter.ImplGetVariable 5543
WebAdapt.TCustomAdapter.ImplGetVariableCount 5544
WebAdapt.TCustomAdapter.ImplGetVariableName 5544
WebAdapt.TCustomAdapter.ImplGetVisibleActions 5544
WebAdapt.TCustomAdapter.ImplGetVisibleFields 5544
WebAdapt.TCustomAdapter.ImplHasExecuteAccess 5545
WebAdapt.TCustomAdapter.ImplHasModifyAccess 5545
WebAdapt.TCustomAdapter.ImplHasViewAccess 5546
WebAdapt.TCustomAdapter.ImplInIterator 5546
WebAdapt.TCustomAdapter.ImplNextIteration 5546
WebAdapt.TCustomAdapter.ImplNotifyActivate 5547
WebAdapt.TCustomAdapter.ImplNotifyDeactivate 5547
WebAdapt.TCustomAdapter.ImplSetActions 5547
WebAdapt.TCustomAdapter.ImplSetData 5548
WebAdapt.TCustomAdapter.ImplSetEchoActionFieldValues 5548

WebAdapt.TCustomAdapter.ImplStartIterator 5548
WebAdapt.TCustomAdapter.InIterator 5549
WebAdapt.TCustomAdapter.IsActionInUse 5549
WebAdapt.TCustomAdapter.IsFieldInUse 5549
WebAdapt.TCustomAdapter.ListHiddenFields 5550
WebAdapt.TCustomAdapter.ListHiddenRecordFields 5550
WebAdapt.TCustomAdapter.ModifyAccess 5560
WebAdapt.TCustomAdapter.NextIteration 5551
WebAdapt.TCustomAdapter.Notification 5551
WebAdapt.TCustomAdapter.NotifyActivate 5551
WebAdapt.TCustomAdapter.NotifyAdapterChange 5551
WebAdapt.TCustomAdapter.NotifyDeactivate 5552
WebAdapt.TCustomAdapter.NotifyList 5560
WebAdapt.TCustomAdapter.OnAfterExecuteAction 5561
WebAdapt.TCustomAdapter.OnAfterGetActionResponse 5562
WebAdapt.TCustomAdapter.OnAfterUpdateFields 5562
WebAdapt.TCustomAdapter.OnAfterValidateFields 5563
WebAdapt.TCustomAdapter.OnBeforeExecuteAction 5563
WebAdapt.TCustomAdapter.OnBeforeGetActionResponse 5563
WebAdapt.TCustomAdapter.OnBeforeUpdateFields 5564
WebAdapt.TCustomAdapter.OnBeforeValidateFields 5564
WebAdapt.TCustomAdapter.OnGetActionParams 5564
WebAdapt.TCustomAdapter.OnIterateRecords 5565
WebAdapt.TCustomAdapter.SetActions 5552
WebAdapt.TCustomAdapter.SetData 5553
WebAdapt.TCustomAdapter.SetEchoActionFieldValues 5553
WebAdapt.TCustomAdapter.StartIterator 5553
WebAdapt.TCustomAdapter.UpdateFieldsExecuteValues 5554
WebAdapt.TCustomAdapter.UpdateFieldsGetAnyChanges 5554
WebAdapt.TCustomAdapter.UpdateFieldsUpdateValues 5555
WebAdapt.TCustomAdapter.UpdateFieldsValidateValues 5555
WebAdapt.TCustomAdapter.UpdateRecords 5556
WebAdapt.TCustomAdapter.UpdateRecordsGetAnyChanges 5556
WebAdapt.TCustomAdapter.UpdateRecordsUpdateValues 5557
WebAdapt.TCustomAdapter.UpdateRecordsValidateValues 5557
WebAdapt.TCustomAdapter.ViewAccess 5560
WebAdapt.TCustomAdapter.VisibleActions 5561
WebAdapt.TCustomAdapter.VisibleFields 5561
WebAdapt.TCustomAdapter.WebComponentsChange 5557
WebAdapt.TCustomAdapterAction 5565
WebAdapt.TCustomAdapterAction.ActionName 5572
WebAdapt.TCustomAdapterAction.ImplExecuteActionRequest 5570
WebAdapt.TCustomAdapterAction.ImplGetActionName 5570
WebAdapt.TCustomAdapterAction.ImplGetAdapterItemRequest Params
5570
WebAdapt.TCustomAdapterAction.ImplGetSuccessOptions 5571
WebAdapt.TCustomAdapterAction.ImplSetActionName 5571
WebAdapt.TCustomAdapterAction.ImplWebEnabled 5571
WebAdapt.TCustomAdapterAction.OnAfterExecute 5573
WebAdapt.TCustomAdapterAction.OnAfterGetResponse 5573
WebAdapt.TCustomAdapterAction.OnBeforeExecute 5574
WebAdapt.TCustomAdapterAction.OnBeforeGetResponse 5574
WebAdapt.TCustomAdapterAction.OnExecute 5574
WebAdapt.TCustomAdapterAction.OnGetEnabled 5575
WebAdapt.TCustomAdapterAction.SetActionName 5572
WebAdapt.TCustomAdapterBooleanField 5575
WebAdapt.TCustomAdapterBooleanField.ConvertValue 5580
WebAdapt.TCustomAdapterBooleanField.GetDisplayStyleType 5580
WebAdapt.TCustomAdapterBooleanField.GetInputStyleType 5581
WebAdapt.TCustomAdapterBooleanField.ImplGetValue 5581
WebAdapt.TCustomAdapterBooleanField.ImplIsEqual 5582
WebAdapt.TCustomAdapterBooleanField.ImplUpdateValue 5582
WebAdapt.TCustomAdapterBooleanField.IsEqual 5582
WebAdapt.TCustomAdapterBooleanField.OnGetValue 5583
WebAdapt.TCustomAdapterBooleanField.OnUpdateValue 5583
WebAdapt.TCustomAdapterField 5583
WebAdapt.TCustomAdapterField.DoOnValidateValue 5588
WebAdapt.TCustomAdapterField.ImplGetValue 5588
WebAdapt.TCustomAdapterField.ImplUpdateValue 5589
WebAdapt.TCustomAdapterField.OnGetValue 5589
WebAdapt.TCustomAdapterField.OnUpdateValue 5590
WebAdapt.TCustomAdapterFileField 5590
WebAdapt.TCustomAdapterFileField.DoOnValidateValue 5595
WebAdapt.TCustomAdapterFileField.FileCount 5598

WebAdapt.TCustomAdapterFileField.FileNames 5598
WebAdapt.TCustomAdapterFileField.GetDisplayStyleType 5595
WebAdapt.TCustomAdapterFileField.GetFileNames 5596
WebAdapt.TCustomAdapterFileField.GetInputStyleType 5596
WebAdapt.TCustomAdapterFileField.GetValueCount 5596
WebAdapt.TCustomAdapterFileField.GetValues 5597
WebAdapt.TCustomAdapterFileField.ImplCheckValueChange 5597
WebAdapt.TCustomAdapterFileField.ImplUpdateValue 5597
WebAdapt.TCustomAdapterFileField.OnGetFileCount 5599
WebAdapt.TCustomAdapterFileField.OnGetFileNames 5599
WebAdapt.TCustomAdapterFileField.OnUploadFiles 5599
WebAdapt.TCustomAdapterFileField.OnValidateFiles 5600
WebAdapt.TCustomAdapterGotoPageAction 5600
WebAdapt.TCustomAdapterGotoPageAction.GetDefaultActionName 5605
WebAdapt.TCustomAdapterGotoPageAction.GetDisplayStyleType 5605
WebAdapt.TCustomAdapterGotoPageAction.GetItemCount 5606
WebAdapt.TCustomAdapterGotoPageAction.ImplExecuteActionRequest 5606
WebAdapt.TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams 5606
WebAdapt.TCustomAdapterGotoPageAction.ImplGetDisplayLabel 5607
WebAdapt.TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList 5607
WebAdapt.TCustomAdapterGotoPageAction.ImplIsDefaultAction 5608
WebAdapt.TCustomAdapterGotoPageAction.ImplWebEnabled 5608
WebAdapt.TCustomAdapterGotoPageAction.OnGetPageName 5608
WebAdapt.TCustomAdapterImageField 5609
WebAdapt.TCustomAdapterImageField.CreateRequestContext 5614
WebAdapt.TCustomAdapterImageField.DoOnValidateValue 5614
WebAdapt.TCustomAdapterImageField.GetAdapterImage 5615
WebAdapt.TCustomAdapterImageField.GetAdapterItemRequestParams 5615
WebAdapt.TCustomAdapterImageField.GetDisplayStyleType 5616
WebAdapt.TCustomAdapterImageField.GetInputStyleType 5616
WebAdapt.TCustomAdapterImageField.HandleRequest 5616
WebAdapt.TCustomAdapterImageField.HREF 5620
WebAdapt.TCustomAdapterImageField.ImplCheckValueChange 5617
WebAdapt.TCustomAdapterImageField.ImplCreateRequestContext 5617
WebAdapt.TCustomAdapterImageField.ImplGetValue 5618
WebAdapt.TCustomAdapterImageField.ImplHandleRequest 5618
WebAdapt.TCustomAdapterImageField.ImplUpdateValue 5619
WebAdapt.TCustomAdapterImageField.ImplWebImageHREF 5619
WebAdapt.TCustomAdapterImageField.OnGetHREF 5621
WebAdapt.TCustomAdapterImageField.OnGetImage 5622
WebAdapt.TCustomAdapterImageField.OnGetImageName 5622
WebAdapt.TCustomAdapterImageField.OnGetParams 5622
WebAdapt.TCustomAdapterImageField.OnUploadFile 5623
WebAdapt.TCustomAdapterImageField.OnValidateFile 5623
WebAdapt.TCustomAdapterImageField.ReadOnly 5621
WebAdapt.TCustomAdapterImageField.RenderAdapterImage 5619
WebAdapt.TCustomAdapterImageField.WebImageHREF 5620
WebAdapt.TCustomAdapterMemoField 5624
WebAdapt.TCustomAdapterMemoField.DoOnValidateValue 5628
WebAdapt.TCustomAdapterMemoField.GetDisplayStyleType 5628
WebAdapt.TCustomAdapterMemoField.GetInputStyleType 5629
WebAdapt.TCustomAdapterMemoField.ImplCheckValueChange 5629
WebAdapt.TCustomAdapterMemoField.ImplGetValue 5630
WebAdapt.TCustomAdapterMemoField.ImplUpdateValue 5630
WebAdapt.TCustomAdapterMemoField.OnGetValue 5631
WebAdapt.TCustomAdapterMemoField.OnUpdateValue 5631

WebAdapt.TCustomAdapterMemoField.OnUploadFile 5631
WebAdapt.TCustomAdapterMultiValueField 5632
WebAdapt.TCustomAdapterMultiValueField.DoOnValidateValue 5637
WebAdapt.TCustomAdapterMultiValueField.GetDisplayStyleType 5637
WebAdapt.TCustomAdapterMultiValueField.GetInputStyleType 5637
WebAdapt.TCustomAdapterMultiValueField.GetValueCount 5638
WebAdapt.TCustomAdapterMultiValueField.GetValues 5638
WebAdapt.TCustomAdapterMultiValueField.ImplHasValue 5639
WebAdapt.TCustomAdapterMultiValueField.ImplUpdateValue 5639
WebAdapt.TCustomAdapterMultiValueField.OnGetValueCount 5640
WebAdapt.TCustomAdapterMultiValueField.OnGetValues 5641
WebAdapt.TCustomAdapterMultiValueField.OnHasValue 5641
WebAdapt.TCustomAdapterMultiValueField.OnUpdateValues 5642
WebAdapt.TCustomAdapterMultiValueField.OnValidateValues 5642
WebAdapt.TCustomAdapterMultiValueField.ValueCount 5639
WebAdapt.TCustomAdapterMultiValueField.Values 5640
WebAdapt.TCustomAdapterPageAction 5642
WebAdapt.TCustomAdapterPageAction.GetDisplayStyleType 5647
WebAdapt.TCustomAdapterPageAction.GetNewPage 5647
WebAdapt.TCustomAdapterPageAction.ImplExecuteActionRequest 5648
WebAdapt.TCustomAdapterPageAction.ImplGetAdapterItemRequestParams 5648
WebAdapt.TCustomAdapterPageAction.ImplIsDefaultAction 5649
WebAdapt.TCustomAdapterPageAction.ImplWebEnabled 5649
WebAdapt.TCustomApplicationAdapter 5649
WebAdapt.TCustomApplicationAdapter.ApplicationTitle 5657
WebAdapt.TCustomApplicationAdapter.GetApplicationAdapter 5655
WebAdapt.TCustomApplicationAdapter.GetApplicationTitle 5655
WebAdapt.TCustomApplicationAdapter.GetWebSnapObjClass 5655
WebAdapt.TCustomApplicationAdapter.ImplCanAddActionClass 5655
WebAdapt.TCustomApplicationAdapter.ImplCanAddFieldClass 5656
WebAdapt.TCustomApplicationAdapter.ImplGetActionsList 5656
WebAdapt.TCustomApplicationAdapter.ImplGetFieldsList 5656
WebAdapt.TCustomApplicationAdapterAction 5657
WebAdapt.TCustomApplicationAdapterAction.Adapter 5662
WebAdapt.TCustomEndUserAdapter 5662
WebAdapt.TCustomEndUserAdapter.ExecuteLogin 5669
WebAdapt.TCustomEndUserAdapter.ExecuteLogout 5669
WebAdapt.TCustomEndUserAdapter.GetDisplayNameValue 5669
WebAdapt.TCustomEndUserAdapter.GetLoggedInValue 5670
WebAdapt.TCustomEndUserAdapter.GetUserID 5670
WebAdapt.TCustomEndUserAdapter.OnGetDisplayName 5671
WebAdapt.TCustomEndUserAdapter.OnGetUserID 5671
WebAdapt.TCustomEndUserAdapter.OnIsLoggedIn 5672
WebAdapt.TCustomEndUserAdapter.OnLogin 5672
WebAdapt.TCustomEndUserAdapter.OnLogout 5673
WebAdapt.TCustomEndUserAdapterAction 5673
WebAdapt.TCustomEndUserAdapterAction.Adapter 5678
WebAdapt.TCustomEndUserSessionAdapter 5678
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogin 5685
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogout 5685
WebAdapt.TCustomEndUserSessionAdapter.GetDisplayNameValue 5685
WebAdapt.TCustomEndUserSessionAdapter.GetLoggedInValue 5686
WebAdapt.TCustomEndUserSessionAdapter.GetUserID 5686
WebAdapt.TCustomEndUserSessionAdapter.UserID 5687
WebAdapt.TCustomLoginFormAdapter 5687
WebAdapt.TCustomLoginFormAdapter.EndUser 5693
WebAdapt.TCustomLoginFormAdapter.ExecuteLogin 5693
WebAdapt.TCustomLoginFormAdapter.GetNextPage 5694
WebAdapt.TCustomLoginFormAdapter.ImplCanAddActionClass 5694
WebAdapt.TCustomLoginFormAdapter.ImplCanAddFieldClass 5695

WebAdapt.TCustomLoginFormAdapter.ImplGetActionsList 5695
WebAdapt.TCustomLoginFormAdapter.ImplGetFieldsList 5696
WebAdapt.TCustomLoginFormAdapter.ImplNotifyDeactivate 5696
WebAdapt.TCustomLoginFormAdapter.NextPage 5699
WebAdapt.TCustomLoginFormAdapter.OnLogin 5701
WebAdapt.TCustomLoginFormAdapter.OnValidatePassword 5702
WebAdapt.TCustomLoginFormAdapter.OnValidateUserName 5702
WebAdapt.TCustomLoginFormAdapter.Password 5700
WebAdapt.TCustomLoginFormAdapter.PasswordRequired 5700
WebAdapt.TCustomLoginFormAdapter.RaiseBlankPassword 5697
WebAdapt.TCustomLoginFormAdapter.RaiseBlankUserName 5697
WebAdapt.TCustomLoginFormAdapter.UserList 5697
WebAdapt.TCustomLoginFormAdapter.UserName 5700
WebAdapt.TCustomLoginFormAdapter.ValidatePassword 5698
WebAdapt.TCustomLoginFormAdapter.ValidateUserName 5698
WebAdapt.TCustomLoginFormAdapterAction 5703
WebAdapt.TCustomLoginFormAdapterAction.Adapter 5707
WebAdapt.TCustomPagedAdapter 5707
WebAdapt.TCustomPagedAdapterAction 5708
WebAdapt.TCustomPagedAdapterAction.Adapter 5713
WebAdapt.TCustomPagedAdapterAction.GetAdapter 5713
WebAdapt.TCustomStringsValuesList 5714
WebAdapt.TCustomStringsValuesList.Create 5719
WebAdapt.TCustomStringsValuesList.Destroy 5719
WebAdapt.TCustomStringsValuesList.GetNameFieldClass 5720
WebAdapt.TCustomStringsValuesList.GetValueFieldClass 5720
WebAdapt.TCustomStringsValuesList.ImplEndIterator 5720
WebAdapt.TCustomStringsValuesList.ImplGetListName 5721
WebAdapt.TCustomStringsValuesList.ImplGetListNameOfValue 5721
WebAdapt.TCustomStringsValuesList.ImplGetListValue 5721
WebAdapt.TCustomStringsValuesList.ImplNextIteration 5722
WebAdapt.TCustomStringsValuesList.ImplNotifyDeactivate 5722
WebAdapt.TCustomStringsValuesList.ImplStartIterator 5722
WebAdapt.TCustomStringsValuesList.OnPrepareStrings 5724
WebAdapt.TCustomStringsValuesList.PrepareStrings 5723
WebAdapt.TCustomStringsValuesList.Strings 5723
WebAdapt.TCustomStringsValuesList.StringsPrepared 5723
WebAdapt.TDefaultFieldsAdapter 5724
WebAdapt.TDefaultFieldsAdapter.Actions 5732
WebAdapt.TDefaultFieldsAdapter.AddDefaultActions 5729
WebAdapt.TDefaultFieldsAdapter.AddDefaultFields 5730
WebAdapt.TDefaultFieldsAdapter.Data 5733
WebAdapt.TDefaultFieldsAdapter.DefaultWebComponentsChange 5730
WebAdapt.TDefaultFieldsAdapter.GetActionsClass 5730
WebAdapt.TDefaultFieldsAdapter.GetDataClass 5731
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleActions 5731
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleFields 5731
WebAdapt.TDefaultFieldsAdapter.ImplSetActions 5732
WebAdapt.TDefaultFieldsAdapter.ImplSetData 5732
WebAdapt.TDefaultFieldsPagedAdapter 5733
WebAdapt.TDefaultFieldsPagedAdapter.AddDefaultActions 5739
WebAdapt.TDefaultFieldsPagedAdapter.CurrentPage 5746
WebAdapt.TDefaultFieldsPagedAdapter.EncodeActionParams 5740
WebAdapt.TDefaultFieldsPagedAdapter.EndOfPage 5740
WebAdapt.TDefaultFieldsPagedAdapter.ExtractRequestParams 5740
WebAdapt.TDefaultFieldsPagedAdapter.GetEOF 5741
WebAdapt.TDefaultFieldsPagedAdapter.GetFirstRecord 5741
WebAdapt.TDefaultFieldsPagedAdapter.GetNextRecord 5741
WebAdapt.TDefaultFieldsPagedAdapter.GetPageCount 5742
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordCount 5742
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordIndex 5742
WebAdapt.TDefaultFieldsPagedAdapter.ImplCanAddActionClass 5743
WebAdapt.TDefaultFieldsPagedAdapter.ImplEndIterator 5743
WebAdapt.TDefaultFieldsPagedAdapter.ImplGetActionsList 5743
WebAdapt.TDefaultFieldsPagedAdapter.ImplNextIteration 5744
WebAdapt.TDefaultFieldsPagedAdapter.ImplNotifyDeactivate 5744
WebAdapt.TDefaultFieldsPagedAdapter.ImplStartIterator 5745
WebAdapt.TDefaultFieldsPagedAdapter.OnGetEOF 5748

WebAdapt.TDefaultFieldsPagedAdapter.OnGetFirstRecord 5749
WebAdapt.TDefaultFieldsPagedAdapter.OnGetNextRecord 5749
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordCount 5749
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordIndex 5749
WebAdapt.TDefaultFieldsPagedAdapter.PageCount 5747
WebAdapt.TDefaultFieldsPagedAdapter.PageSize 5747
WebAdapt.TDefaultFieldsPagedAdapter.RecordCount 5747
WebAdapt.TDefaultFieldsPagedAdapter.RecordIndex 5748
WebAdapt.TDefaultFieldsPagedAdapter.SetPageSize 5745
WebAdapt.TDefaultFieldsPagedAdapter.SkipToPage 5746
WebAdapt.TEndUserAdapter 5750
WebAdapt.TEndUserAdapter.Actions 5756
WebAdapt.TEndUserAdapter.Data 5756
WebAdapt.TEndUserAdapter.LoginPage 5757
WebAdapt.TEndUserAdapter.OnAfterExecuteAction 5757
WebAdapt.TEndUserAdapter.OnAfterGetActionResponse 5758
WebAdapt.TEndUserAdapter.OnBeforeExecuteAction 5758
WebAdapt.TEndUserAdapter.OnBeforeGetActionResponse 5758
WebAdapt.TEndUserAdapter.OnGetActionParams 5759
WebAdapt.TEndUserAdapter.OnGetDisplayName 5759
WebAdapt.TEndUserAdapter.OnHasRights 5760
WebAdapt.TEndUserAdapter.OnIsLoggedIn 5760
WebAdapt.TEndUserAdapter.OnLogin 5761
WebAdapt.TEndUserAdapter.OnLogout 5761
WebAdapt.TEndUserAdapter.OnProduceLoginPage 5762
WebAdapt.TEndUserAdapterLoginFormAction 5762
WebAdapt.TEndUserAdapterLoginFormAction.DisplayLabel 5768
WebAdapt.TEndUserAdapterLoginFormAction.GetDefaultActionName 5766
WebAdapt.TEndUserAdapterLoginFormAction.ImplExecuteActionRequest 5767
WebAdapt.TEndUserAdapterLoginFormAction.ImplWebEnabled 5767
WebAdapt.TEndUserAdapterLoginFormAction.OnAfterExecute 5768
WebAdapt.TEndUserAdapterLoginFormAction.OnAfterGetResponse 5769
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeExecute 5769
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeGetResponse 5769
WebAdapt.TEndUserAdapterLoginFormAction.OnGetParams 5770
WebAdapt.TEndUserAdapterLogoutAction 5770
WebAdapt.TEndUserAdapterLogoutAction.DisplayLabel 5776
WebAdapt.TEndUserAdapterLogoutAction.GetDefaultActionName 5775
WebAdapt.TEndUserAdapterLogoutAction.ImplExecuteActionRequest 5775
WebAdapt.TEndUserAdapterLogoutAction.ImplWebEnabled 5775
WebAdapt.TEndUserAdapterLogoutAction.OnAfterExecute 5777
WebAdapt.TEndUserAdapterLogoutAction.OnAfterGetResponse 5777
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeExecute 5777
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeGetResponse 5778
WebAdapt.TEndUserAdapterLogoutAction.OnGetParams 5778
WebAdapt.TEndUserGetNameEvent 5899
WebAdapt.TEndUserGetNameEvent type 5899
WebAdapt.TEndUserGetUserIDEvent 5900
WebAdapt.TEndUserGetUserIDEvent type 5900
WebAdapt.TEndUserHandledEvent 5900
WebAdapt.TEndUserHandledEvent type 5900
WebAdapt.TEndUserIsLoggedInEvent 5900
WebAdapt.TEndUserIsLoggedInEvent type 5900
WebAdapt.TEndUserRightsEvent 5901
WebAdapt.TEndUserRightsEvent type 5901
WebAdapt.TEndUserSessionAdapter 5779
WebAdapt.TEndUserSessionAdapter.Actions 5785
WebAdapt.TEndUserSessionAdapter.Data 5786
WebAdapt.TEndUserSessionAdapter.LoginPage 5786

WebAdapt.TEndUserSessionAdapter.OnAfterExecuteAction 5787
WebAdapt.TEndUserSessionAdapter.OnAfterGetActionResponse 5787
WebAdapt.TEndUserSessionAdapter.OnBeforeExecuteAction 5787
WebAdapt.TEndUserSessionAdapter.OnBeforeGetActionResponse 5788
WebAdapt.TEndUserSessionAdapter.OnGetActionParams 5788
WebAdapt.TEndUserSessionAdapter.OnHasRights 5788
WebAdapt.TEndUserSessionAdapter.OnProduceLoginPage 5789
WebAdapt.TEndUserUserIDEvent 5901
WebAdapt.TEndUserUserIDEvent type 5901
WebAdapt.TFieldValuesAdapter 5789
WebAdapt.TFieldValuesAdapter.Create 5795
WebAdapt.TFieldValuesAdapter.Field 5798
WebAdapt.TFieldValuesAdapter.GetValueFieldClass 5795
WebAdapt.TFieldValuesAdapter.ImplEndIterator 5796
WebAdapt.TFieldValuesAdapter.ImplGetField 5796
WebAdapt.TFieldValuesAdapter.ImplGetValueIndex 5796
WebAdapt.TFieldValuesAdapter.ImplNextIteration 5797
WebAdapt.TFieldValuesAdapter.ImplStartIterator 5797
WebAdapt.TFieldValuesValueField 5799
WebAdapt.TFieldValuesValueField.Field 5803
WebAdapt.TFieldValuesValueField.GetDefaultFieldName 5802
WebAdapt.TFieldValuesValueField.ImplAsFormatted 5802
WebAdapt.TFieldValuesValueField.ImplGetValue 5803
WebAdapt.TGetEOFEvent 5901
WebAdapt.TGetEOFEvent type 5901
WebAdapt.TGetRecordCountEvent 5902
WebAdapt.TGetRecordCountEvent type 5902
WebAdapt.TGetRecordEvent 5902
WebAdapt.TGetRecordEvent type 5902
WebAdapt.TGetRecordIndexEvent 5902
WebAdapt.TGetRecordIndexEvent type 5902
WebAdapt.THandledEvent 5903
WebAdapt.THandledEvent type 5903
WebAdapt.TImageFieldGetHREFEvent 5903
WebAdapt.TImageFieldGetHREFEvent type 5903
WebAdapt.TImageFieldGetImageEvent 5903
WebAdapt.TImageFieldGetImageEvent type 5903
WebAdapt.TImageFieldGetParamsEvent 5904
WebAdapt.TImageFieldGetParamsEvent type 5904
WebAdapt.TImplementedAdapterAction 5803
WebAdapt.TIterateRecordsEvent 5904
WebAdapt.TIterateRecordsEvent type 5904
WebAdapt.TIteratorMethod 5887
WebAdapt.TIteratorMethod enumeration 5887
WebAdapt.TLogicalPageList 5805
WebAdapt.TLogicalPageList.Create 5810
WebAdapt.TLoginFormAdapter 5810
WebAdapt.TLoginFormAdapter.Actions 5816
WebAdapt.TLoginFormAdapter.Data 5817
WebAdapt.TLoginFormAdapter.OnAfterExecuteAction 5818
WebAdapt.TLoginFormAdapter.OnAfterGetActionResponse 5818
WebAdapt.TLoginFormAdapter.OnBeforeExecuteAction 5819
WebAdapt.TLoginFormAdapter.OnBeforeGetActionResponse 5819
WebAdapt.TLoginFormAdapter.OnGetActionParams 5819
WebAdapt.TLoginFormAdapter.OnLogin 5820
WebAdapt.TLoginFormAdapter.OnValidatePassword 5820
WebAdapt.TLoginFormAdapter.OnValidateUserName 5821
WebAdapt.TLoginFormAdapter.PasswordRequired 5817
WebAdapt.TLoginFormAdapterLoginAction 5821
WebAdapt.TLoginFormAdapterLoginAction.DisplayLabel 5828
WebAdapt.TLoginFormAdapterLoginAction.GetDefaultActionName 5826
WebAdapt.TLoginFormAdapterLoginAction.ImplExecuteActionRequest 5826
WebAdapt.TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams 5827
WebAdapt.TLoginFormAdapterLoginAction.NextPage 5827
WebAdapt.TLoginFormAdapterLoginAction.OnAfterExecute 5828
WebAdapt.TLoginFormAdapterLoginAction.OnAfterGetResponse 5829
WebAdapt.TLoginFormAdapterLoginAction.OnBeforeExecute 5829

WebAdapt.TLoginFormAdapterLoginAction.OnBeforeGetResponse 5830
WebAdapt.TLoginFormAdapterLoginAction.OnGetParams 5830
WebAdapt.TLoginUserIDEvent 5904
WebAdapt.TLoginUserIDEvent type 5904
WebAdapt.TObjectList 5905
WebAdapt.TObjectList type 5905
WebAdapt.TPagedAdapter 5830
WebAdapt.TPagedAdapter.Actions 5836
WebAdapt.TPagedAdapter.Data 5836
WebAdapt.TPagedAdapter.ModifyAccess 5837
WebAdapt.TPagedAdapter.OnAfterExecuteAction 5838
WebAdapt.TPagedAdapter.OnAfterGetActionResponse 5838
WebAdapt.TPagedAdapter.OnAfterUpdateFields 5839
WebAdapt.TPagedAdapter.OnAfterValidateFields 5839
WebAdapt.TPagedAdapter.OnBeforeExecuteAction 5839
WebAdapt.TPagedAdapter.OnBeforeGetActionResponse 5840
WebAdapt.TPagedAdapter.OnBeforeUpdateFields 5840
WebAdapt.TPagedAdapter.OnBeforeValidateFields 5841
WebAdapt.TPagedAdapter.OnGetActionParams 5841
WebAdapt.TPagedAdapter.OnGetEOF 5841
WebAdapt.TPagedAdapter.OnGetFirstRecord 5842
WebAdapt.TPagedAdapter.OnGetNextRecord 5842
WebAdapt.TPagedAdapter.OnGetRecordCount 5842
WebAdapt.TPagedAdapter.OnGetRecordIndex 5842
WebAdapt.TPagedAdapter.PageSize 5837
WebAdapt.TPagedAdapter.ViewAccess 5837
WebAdapt.TStringsValuesList 5843
WebAdapt.TStringsValuesList.Actions 5848
WebAdapt.TStringsValuesList.Data 5849
WebAdapt.TStringsValuesList.OnPrepareStrings 5849
WebAdapt.TStringsValuesList.Strings 5849
WebAdapt.TUpdateFileList 5850
WebAdapt.TUpdateFileList.Count 5851
WebAdapt.TUpdateFileList.Create 5850
WebAdapt.TUpdateFileList.Files 5851
WebAdapt.TValidateAdapterFieldEvent 5905
WebAdapt.TValidateAdapterFieldEvent type 5905
WebAdapt.TValuesListNameField 5852
WebAdapt.TValuesListNameField.GetDefaultFieldName 5855
WebAdapt.TValuesListNameField.ImplGetValue 5855
WebAdapt.TValuesListValueField 5855
WebAdapt.TValuesListValueField.GetDefaultFieldName 5859
WebAdapt.TValuesListValueField.ImplGetValue 5859
WebAdapt.TWebFormGetPageNameEvent 5905
WebAdapt.TWebFormGetPageNameEvent type 5905
WebBroker 5911
WebBroker namespace 5911
  Classes 5912
  Types 5917
  Variables 5918
WebBroker.Application 5918
WebBroker.Application variable 5918
WebBroker.HandleShutdownException 5918
WebBroker.HandleShutdownException variable 5918
WebBroker.THandleShutdownException 5917
WebBroker.THandleShutdownException type 5917
WebBroker.TServerExceptionEvent 5917
WebBroker.TServerExceptionEvent type 5917
WebBroker.TWebApplication 5912
WebBroker.TWebApplication.Create 5913
WebBroker.TWebApplication.CreateForm 5914
WebBroker.TWebApplication.Destroy 5914
WebBroker.TWebApplication.Initialize 5915
WebBroker.TWebApplication.Run 5915
WebBroker.TWebApplication.Title 5916
WebBrokerSOAP 5919
WebBrokerSOAP namespace 5919
  Classes 5920
  Functions 5925
WebBrokerSOAP.GetSOAPWebModule 5925
WebBrokerSOAP.GetSOAPWebModule function 5925
WebBrokerSOAP.THTTPSoapDispatcher 5920
WebBrokerSOAP.THTTPSoapDispatcher.Create 5921
WebBrokerSOAP.THTTPSoapDispatcher.Destroy 5922
WebBrokerSOAP.THTTPSoapDispatcher.DispatchEnabled 5922
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMask 5923
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMethodType

5923
WebBrokerSOAP.THTTPSoapDispatcher.DispatchRequest 5923
WebBrokerSOAP.THTTPSoapDispatcher.WebDispatch 5924
WebCntxt 5927
WebCntxt namespace 5927
Classes 5929
Functions 5977
Interfaces 5962
Structs, Records, Enums 5978
Types 5979
Variables 5981
WebCntxt.GetModuleFileNameProc 5981
WebCntxt.GetModuleFileNameProc variable 5981
WebCntxt.GetWebContextProc 5981
WebCntxt.GetWebContextProc variable 5981
WebCntxt.GetWebEndUserProc 5982
WebCntxt.GetWebEndUserProc variable 5982
WebCntxt.IWebEndUser 5962
WebCntxt.IWebEndUser.DisplayName 5968
WebCntxt.IWebEndUser.EndUserAdapter 5968
WebCntxt.IWebEndUser.GetDisplayName 5963
WebCntxt.IWebEndUser.GetEndUserAdapter 5964
WebCntxt.IWebEndUser.GetLoggedIn 5964
WebCntxt.IWebEndUser.GetLoginFormAction 5964
WebCntxt.IWebEndUser.GetLogoutAction 5965
WebCntxt.IWebEndUser.HasExecuteRights 5965
WebCntxt.IWebEndUser.HasModifyRights 5966
WebCntxt.IWebEndUser.HasViewRights 5966
WebCntxt.IWebEndUser.LoggedIn 5968
WebCntxt.IWebEndUser.LogIn 5967
WebCntxt.IWebEndUser.LoginFormAction 5969
WebCntxt.IWebEndUser.LogoutAction 5970
WebCntxt.IWebEndUser.ProduceLoginPage 5967
WebCntxt.IWebPageInfo 5970
WebCntxt.IWebPageInfo.GetHasViewAccess 5971
WebCntxt.IWebPageInfo.GetLoginRequired 5972
WebCntxt.IWebPageInfo.GetPageDescription 5972
WebCntxt.IWebPageInfo.GetPageHREF 5972
WebCntxt.IWebPageInfo.GetPageIsPublished 5973
WebCntxt.IWebPageInfo.GetPageName 5973
WebCntxt.IWebPageInfo.GetPageTitle 5973
WebCntxt.IWebPageInfo.HasViewAccess 5974
WebCntxt.IWebPageInfo.LoginRequired 5974
WebCntxt.IWebPageInfo.PageDescription 5974
WebCntxt.IWebPageInfo.PageHREF 5975
WebCntxt.IWebPageInfo.PageIsPublished 5975
WebCntxt.IWebPageInfo.PageName 5976
WebCntxt.IWebPageInfo.PageTitle 5976
WebCntxt.SetWebContextProc 5982
WebCntxt.SetWebContextProc variable 5982
WebCntxt.TAbstractAdapterRequest 5929
WebCntxt.TAbstractAdapterRequest.Create 5931
WebCntxt.TAbstractAdapterRequest.Request 5932
WebCntxt.TAbstractAdapterResponse 5932
WebCntxt.TAbstractAdapterResponse.Create 5934
WebCntxt.TAbstractAdapterResponse.Response 5934
WebCntxt.TAbstractWebContext 5935
WebCntxt.TAbstractWebContext.AdapterRequest 5946
WebCntxt.TAbstractWebContext.AdapterResponse 5946
WebCntxt.TAbstractWebContext.Create 5937
WebCntxt.TAbstractWebContext.Destroy 5937
WebCntxt.TAbstractWebContext.DispatchedPageName 5947
WebCntxt.TAbstractWebContext.EndUser 5947
WebCntxt.TAbstractWebContext.FindApplicationModule 5938
WebCntxt.TAbstractWebContext.FindDefaultPageInfo 5939
WebCntxt.TAbstractWebContext.FindModuleClass 5939
WebCntxt.TAbstractWebContext.FindModuleName 5940
WebCntxt.TAbstractWebContext.FindPage 5940
WebCntxt.TAbstractWebContext.FindPageInfo 5941
WebCntxt.TAbstractWebContext.FindVariableContainer 5941
WebCntxt.TAbstractWebContext.GetAdapterRequest 5942
WebCntxt.TAbstractWebContext.GetAdapterResponse 5942
WebCntxt.TAbstractWebContext.GetDispatchedPageName 5942
WebCntxt.TAbstractWebContext.GetEndUser 5942
WebCntxt.TAbstractWebContext.GetPageCount 5943
WebCntxt.TAbstractWebContext.GetPageInfo 5943
WebCntxt.TAbstractWebContext.GetRequestedPageName 5943

WebCntxt.TAbstractWebContext.GetSession 5943
WebCntxt.TAbstractWebContext.GetWebModules 5944
WebCntxt.TAbstractWebContext.GetWebRequest 5944
WebCntxt.TAbstractWebContext.GetWebResponse 5944
WebCntxt.TAbstractWebContext.PageCount 5947
WebCntxt.TAbstractWebContext.Pages 5948
WebCntxt.TAbstractWebContext.Request 5948
WebCntxt.TAbstractWebContext.RequestedPageName 5949
WebCntxt.TAbstractWebContext.Response 5949
WebCntxt.TAbstractWebContext.Session 5949
WebCntxt.TAbstractWebContext.SetAdapterRequest 5944
WebCntxt.TAbstractWebContext.SetAdapterResponse 5945
WebCntxt.TAbstractWebContext.SetDispatchedPageName 5945
WebCntxt.TAbstractWebContext.SetRequestedPageName 5945
WebCntxt.TAbstractWebContext.WebModules 5950
WebCntxt.TFindPageOption 5978
WebCntxt.TFindPageOption enumeration 5978
WebCntxt.TFindPageOptions 5979
WebCntxt.TFindPageOptions type 5979
WebCntxt.TGetWebContextProc 5979
WebCntxt.TGetWebContextProc type 5979
WebCntxt.TGetWebEndUserProc 5980
WebCntxt.TGetWebEndUserProc type 5980
WebCntxt.TSetWebContextProc 5980
WebCntxt.TSetWebContextProc type 5980
WebCntxt.TWebContext 5950
WebCntxt.TWebContext.Create 5952
WebCntxt.TWebContext.Destroy 5953
WebCntxt.TWebContext.FindApplicationModule 5954
WebCntxt.TWebContext.FindDefaultPageInfo 5955
WebCntxt.TWebContext.FindModuleClass 5955
WebCntxt.TWebContext.FindModuleName 5956
WebCntxt.TWebContext.FindPage 5956
WebCntxt.TWebContext.FindPageInfo 5957
WebCntxt.TWebContext.FindVariableContainer 5957
WebCntxt.TWebContext.GetAdapterRequest 5958
WebCntxt.TWebContext.GetAdapterResponse 5958
WebCntxt.TWebContext.GetDispatchedPageName 5958
WebCntxt.TWebContext.GetEndUser 5958
WebCntxt.TWebContext.GetPageCount 5959
WebCntxt.TWebContext.GetPageInfo 5959
WebCntxt.TWebContext.GetRequestedPageName 5959
WebCntxt.TWebContext.GetSession 5959
WebCntxt.TWebContext.GetWebModules 5960
WebCntxt.TWebContext.GetWebRequest 5960
WebCntxt.TWebContext.GetWebResponse 5960
WebCntxt.TWebContext.SetAdapterRequest 5960
WebCntxt.TWebContext.SetAdapterResponse 5961
WebCntxt.TWebContext.SetDispatchedPageName 5961
WebCntxt.TWebContext.SetRequestedPageName 5961
WebCntxt.WebContext 5977
WebCntxt.WebContext function 5977
WebComp 5983
WebComp namespace 5983
  Classes 5985
  Constants 6087
  Functions 6079
  Interfaces 6052
  Structs, Records, Enums 6082
  Types 6084
WebComp.DesigningComponent 6079
WebComp.DesigningComponent function 6079
WebComp.EnumRegisteredWebComponents 6079
WebComp.EnumRegisteredWebComponents function 6079
WebComp.EnumRegisteredWebComponentsProc 6087
WebComp.EnumRegisteredWebComponentsProc constant 6087
WebComp.FindWebComponentEditorHelper 6080
WebComp.FindWebComponentEditorHelper function 6080
WebComp.FindWebComponentEditorHelperProc 6087
WebComp.FindWebComponentEditorHelperProc constant 6087
WebComp.IAddFieldsEditor 6052
WebComp.IAddFieldsEditor.GetAddAllFieldsItem 6053
WebComp.IAddFieldsEditor.GetAddFieldsDlgCaption 6053
WebComp.IAddFieldsEditor.GetAddFieldsItem 6053
WebComp.IAddFieldsEditor.GetNewFieldPrefix 6054
WebComp.IGetWebComponentList 6054
WebComp.IGetWebComponentList.GetComponentList 6055
WebComp.IGetWebComponentList.GetDefaultComponentList

6055
WebComp.IGetWebComponentNodeName 6055
WebComp.IGetWebComponentNodeName.GetWebComponentNodeName 6056
WebComp.IGetXMLStream 6056
WebComp.IGetXMLStream.GetXMLStream 6057
WebComp.ILayoutWebContent 6057
WebComp.ILayoutWebContent.LayoutButton 6058
WebComp.ILayoutWebContent.LayoutField 6058
WebComp.ILayoutWebContent.LayoutLabelAndField 6058
WebComp.ILayoutWebContent.LayoutTable 6059
WebComp.INotifyActivateModule 6059
WebComp.INotifyActivateModule.NotifyActivateModule 6060
WebComp.INotifyActivateModule.NotifyDeactivateModule 6060
WebComp.INotifyWebActivate 6060
WebComp.INotifyWebActivate.NotifyActivate 6061
WebComp.INotifyWebActivate.NotifyDeactivate 6061
WebComp.IProducerEditorViewSupport 6062
WebComp.IProducerEditorViewSupport.GetTemplateFileType 6063
WebComp.IProducerEditorViewSupport.GetXMLData 6063
WebComp.IProducerEditorViewSupport.GetXSLData 6063
WebComp.IProducerEditorViewSupport.HasHTMLBrowserView 6063
WebComp.IProducerEditorViewSupport.HasScriptView 6063
WebComp.IProducerEditorViewSupport.HasXMLBrowserView 6064
WebComp.IProducerEditorViewSupport.HasXSLBrowserView 6064
WebComp.ISetWebContentOptions 6064
WebComp.ISetWebContentOptions.SetWebContentOptions 6065
WebComp.ITopLevelWebComponent 6065
WebComp.IWebComponent 6065
WebComp.IWebComponent.ComponentList 6068
WebComp.IWebComponent.Container 6069
WebComp.IWebComponent.GetContainer 6067
WebComp.IWebComponent.GetIndex 6067
WebComp.IWebComponent.Index 6069
WebComp.IWebComponent.SetComponentList 6067
WebComp.IWebComponent.SetContainer 6067
WebComp.IWebComponent.SetIndex 6068
WebComp.IWebComponentCommands 6069
WebComp.IWebComponentCommands.ExecuteVerb 6070
WebComp.IWebComponentCommands.GetVerb 6070
WebComp.IWebComponentCommands.GetVerbCount 6071
WebComp.IWebComponentContainer 6071
WebComp.IWebComponentContainer.ComponentCount 6072
WebComp.IWebComponentContainer.Components 6073
WebComp.IWebComponentContainer.GetComponent 6072
WebComp.IWebComponentContainer.GetComponentCount 6072
WebComp.IWebComponentEditor 6073
WebComp.IWebComponentEditor.CanAddClass 6074
WebComp.IWebComponentEditorHelper 6074
WebComp.IWebComponentEditorHelper.CanAddClassHelper 6075
WebComp.IWebContent 6075
WebComp.IWebContent.Content 6076
WebComp.IWebVariablesContainerName 6077
WebComp.IWebVariablesContainerName.ContainerName 6078
WebComp.IWebVariablesContainerName.GetContainerName 6077
WebComp.QualifyFileName 6080
WebComp.QualifyFileName function 6080
WebComp.RegisterWebComponents 6080
WebComp.RegisterWebComponents function 6080
WebComp.RegisterWebComponentsProc 6088
WebComp.RegisterWebComponentsProc constant 6088
WebComp.TEnumWebComponentsProc 6084
WebComp.TEnumWebComponentsProc type 6084
WebComp.TGetComponentListEvent 6084
WebComp.TGetComponentListEvent type 6084
WebComp.TLabelPosition 6082
WebComp.TLabelPosition enumeration 6082
WebComp.TLayout 6085
WebComp.TLayout type 6085
WebComp.TLayoutAttributes 5985
WebComp.TLayoutAttributes.ControlAttributes 5987
WebComp.TLayoutAttributes.Create 5987
WebComp.TLayoutAttributes.LabelAttributes 5987

WebComp.TLayoutAttributes.LabelPosition 5988
WebComp.TLayoutWebContent 5988
WebComp.TLayoutWebContent._AddRef 5991
WebComp.TLayoutWebContent._Release 5991
WebComp.TLayoutWebContent.Create 5990
WebComp.TLayoutWebContent.ImplLayoutButton 5991
WebComp.TLayoutWebContent.ImplLayoutField 5992
WebComp.TLayoutWebContent.ImplLayoutLabelAndField 5992
WebComp.TLayoutWebContent.ImplLayoutTable 5993
WebComp.TLayoutWebContent.LayoutButton 5994
WebComp.TLayoutWebContent.LayoutField 5994
WebComp.TLayoutWebContent.LayoutLabelAndField 5994
WebComp.TLayoutWebContent.LayoutTable 5995
WebComp.TLayoutWebContent.ParentLayout 5996
WebComp.TLayoutWebContent.QueryInterface 5995
WebComp.TSetComponentListEvent 6085
WebComp.TSetComponentListEvent type 6085
WebComp.TWebComponentClassDescription 5996
WebComp.TWebComponentClassDescription.ComponentClass 5998
WebComp.TWebComponentClassDescription.ComponentClassName 5998
WebComp.TWebComponentClassDescription.Editor 5998
WebComp.TWebComponentClassDescription.FrameClass 5999
WebComp.TWebComponentContainer 6085
WebComp.TWebComponentContainer type 6085
WebComp.TWebComponentList 5999
WebComp.TWebComponentList._AddRef 6002
WebComp.TWebComponentList._Release 6002
WebComp.TWebComponentList.Add 6002
WebComp.TWebComponentList.Changed 6003
WebComp.TWebComponentList.Clear 6003
WebComp.TWebComponentList.Count 6007
WebComp.TWebComponentList.Create 6001
WebComp.TWebComponentList.Destroy 6001
WebComp.TWebComponentList.GetComponent 6003
WebComp.TWebComponentList.GetCount 6004
WebComp.TWebComponentList.GetOwner 6004
WebComp.TWebComponentList.IndexOf 6004
WebComp.TWebComponentList.LockNotify 6005
WebComp.TWebComponentList.NewComponent 6005
WebComp.TWebComponentList.NotifyLocked 6005
WebComp.TWebComponentList.OnChange 6009
WebComp.TWebComponentList.ParentComponent 6008
WebComp.TWebComponentList.QueryInterface 6005
WebComp.TWebComponentList.Remove 6005
WebComp.TWebComponentList.SetComponentIndex 6006
WebComp.TWebComponentList.UnlockNotify 6006
WebComp.TWebComponentList.WebComponentContainer_GetComponent 6007
WebComp.TWebComponentList.WebComponentContainer_GetComponentCount 6007
WebComp.TWebComponentList.WebComponents 6008
WebComp.TWebComponentsEditorHelper 6009
WebComp.TWebComponentsEditorHelper._AddRef 6011
WebComp.TWebComponentsEditorHelper._Release 6011
WebComp.TWebComponentsEditorHelper.CanAddClassHelper 6011
WebComp.TWebComponentsEditorHelper.ImplCanAddClassHelper 6011
WebComp.TWebComponentsEditorHelper.QueryInterface 6011
WebComp.TWebContainedComponent 6012
WebComp.TWebContainedComponent.Create 6013
WebComp.TWebContainedComponent.Destroy 6014
WebComp.TWebContainedComponent.GetContainerList 6014
WebComp.TWebContainedComponent.GetParentComponent 6015
WebComp.TWebContainedComponent.HasParent 6015
WebComp.TWebContainedComponent.IsDefaultField 6015
WebComp.TWebContainedComponent.ParentChanged 6016
WebComp.TWebContainedComponent.ReadState 6016
WebComp.TWebContainedComponent.SetParentComponent 6017
WebComp.TWebContainedComponent.SetWebParent 6017
WebComp.TWebContainedComponent.WebContainedComponentHelper 6018
WebComp.TWebContainedComponent.WebParent 6018
WebComp.TWebContainedComponentHelper 6018

WebComp.TWebContainedComponentHelper.ContainerList 6025
WebComp.TWebContainedComponentHelper.Create 6020
WebComp.TWebContainedComponentHelper.Destroy 6021
WebComp.TWebContainedComponentHelper.GetContainer 6021
WebComp.TWebContainedComponentHelper.GetIndex 6022
WebComp.TWebContainedComponentHelper.IsDefaultField 6022
WebComp.TWebContainedComponentHelper.OnParentChanged 6025
WebComp.TWebContainedComponentHelper.ParentChanged 6023
WebComp.TWebContainedComponentHelper.SetComponentList 6023
WebComp.TWebContainedComponentHelper.SetContainer 6023
WebComp.TWebContainedComponentHelper.SetIndex 6024
WebComp.TWebContainedComponentHelper.SetWebParent 6024
WebComp.TWebContainedComponentHelper.WebParent 6025
WebComp.TWebContainedContainerComponent 6026
WebComp.TWebContainedContainerComponent.Create 6028
WebComp.TWebContainedContainerComponent.Destroy 6028
WebComp.TWebContainedContainerComponent.GetChildren 6029
WebComp.TWebContainedContainerComponent.GetContainerList 6029
WebComp.TWebContainedContainerComponent.GetDefaultWebComponents 6030
WebComp.TWebContainedContainerComponent.GetParentComponent 6030
WebComp.TWebContainedContainerComponent.GetWebComponents 6031
WebComp.TWebContainedContainerComponent.HasParent 6031
WebComp.TWebContainedContainerComponent.ImplGetDefaultComponentList 6031
WebComp.TWebContainedContainerComponent.IsDefaultField 6032
WebComp.TWebContainedContainerComponent.ParentChanged 6032
WebComp.TWebContainedContainerComponent.ReadState 6033
WebComp.TWebContainedContainerComponent.SetChildOrder 6033
WebComp.TWebContainedContainerComponent.SetDefaultWebComponents 6033
WebComp.TWebContainedContainerComponent.SetParentComponent 6034
WebComp.TWebContainedContainerComponent.SetWebParent 6034
WebComp.TWebContainedContainerComponent.WebContainedComponentHelper 6035
WebComp.TWebContainedContainerComponent.WebContainerComponentHelper 6035
WebComp.TWebContainedContainerComponent.WebParent 6036
WebComp.TWebContainerComponent 6036
WebComp.TWebContainerComponent.Create 6037
WebComp.TWebContainerComponent.Destroy 6038
WebComp.TWebContainerComponent.GetChildren 6038
WebComp.TWebContainerComponent.GetDefaultWebComponents 6039
WebComp.TWebContainerComponent.GetWebComponents 6039
WebComp.TWebContainerComponent.ImplGetDefaultComponentList 6039
WebComp.TWebContainerComponent.SetChildOrder 6040
WebComp.TWebContainerComponent.SetDefaultWebComponents 6040
WebComp.TWebContainerComponent.WebContainerComponentHelper 6041
WebComp.TWebContainerComponentHelper 6041
WebComp.TWebContainerComponentHelper.Create 6043
WebComp.TWebContainerComponentHelper.DefaultWebComponents 6047

WebComp.TWebContainerComponentHelper.Destroy 6044
WebComp.TWebContainerComponentHelper.GetChildren 6044
WebComp.TWebContainerComponentHelper.GetComponentLis
t
6045
WebComp.TWebContainerComponentHelper.GetDefaultCompo
nentList
6045
WebComp.TWebContainerComponentHelper.GetDefaultWebCo
mponents
6046
WebComp.TWebContainerComponentHelper.OnGetDefaultCom
ponentList
6048
WebComp.TWebContainerComponentHelper.OnSetDefaultCom
ponentList
6049
WebComp.TWebContainerComponentHelper.SetChildOrder
6046
WebComp.TWebContainerComponentHelper.SetDefaultWebCo
mponents
6046
WebComp.TWebContainerComponentHelper.Webcomponents
6048
WebComp.TWebContentFlag 6082
WebComp.TWebContentFlag enumeration 6082
WebComp.TWebContentFlags 6086
WebComp.TWebContentFlags type 6086
WebComp.TWebContentOptions 6049
WebComp.TWebContentOptions.Create 6050
WebComp.TWebContentOptions.Flags 6051
WebComp.UnRegisterWebComponents 6081
WebComp.UnRegisterWebComponents function 6081
WebComp.UnRegisterWebComponentsProc 6088
WebComp.UnRegisterWebComponentsProc constant 6088
WebComp.ValidateWebParent 6081
WebComp.ValidateWebParent function 6081
WebContnrs 6089
WebContnrs namespace 6089
Classes 6090
Structs, Records, Enums 6113
Types 6114
WebContnrs.ENamedVariantsError 6090
WebContnrs.PNamedVariantItem 6114
WebContnrs.PNamedVariantItem type 6114
WebContnrs.PNamedVariantList 6114
WebContnrs.PNamedVariantList type 6114
WebContnrs.TAbstractNamedVariants 6090
WebContnrs.TAbstractNamedVariants.Add 6093
WebContnrs.TAbstractNamedVariants.Append 6094
WebContnrs.TAbstractNamedVariants.Assign 6094
WebContnrs.TAbstractNamedVariants.BeginUpdate 6094
WebContnrs.TAbstractNamedVariants.Capacity 6103
WebContnrs.TAbstractNamedVariants.Clear 6095
WebContnrs.TAbstractNamedVariants.CompareStrings 6095
WebContnrs.TAbstractNamedVariants.Count 6103
WebContnrs.TAbstractNamedVariants.DefineProperties 6096
WebContnrs.TAbstractNamedVariants.Delete 6096
WebContnrs.TAbstractNamedVariants.Destroy 6092
WebContnrs.TAbstractNamedVariants.EndUpdate 6097
WebContnrs.TAbstractNamedVariants.Equals 6097
WebContnrs.TAbstractNamedVariants.Error 6098
WebContnrs.TAbstractNamedVariants.Exchange 6098
WebContnrs.TAbstractNamedVariants.Get 6098
WebContnrs.TAbstractNamedVariants.GetCapacity 6099
WebContnrs.TAbstractNamedVariants.GetCount 6099
WebContnrs.TAbstractNamedVariants.IndexOfName 6100
WebContnrs.TAbstractNamedVariants.Insert 6100
WebContnrs.TAbstractNamedVariants.Move 6101
WebContnrs.TAbstractNamedVariants.Names 6104
WebContnrs.TAbstractNamedVariants.Put 6101
WebContnrs.TAbstractNamedVariants.SetCapacity 6102
WebContnrs.TAbstractNamedVariants.SetUpdateState 6102
WebContnrs.TAbstractNamedVariants.UpdateCount 6104
WebContnrs.TAbstractNamedVariants.Values 6105
WebContnrs.TAbstractNamedVariants.Variants 6105
WebContnrs.TNamedVariantItem 6113
WebContnrs.TNamedVariantItem record 6113
WebContnrs.TNamedVariantList 6114
WebContnrs.TNamedVariantList type 6114
WebContnrs.TNamedVariantsList 6106
WebContnrs.TNamedVariantsList.Clear 6108
WebContnrs.TNamedVariantsList.Delete 6108

WebContnrs.TNamedVariantsList.Destroy 6107
WebContnrs.TNamedVariantsList.Get 6109
WebContnrs.TNamedVariantsList.GetCapacity 6109
WebContnrs.TNamedVariantsList.GetCount 6110
WebContnrs.TNamedVariantsList.Insert 6110
WebContnrs.TNamedVariantsList.Put 6111
WebContnrs.TNamedVariantsList.SetCapacity 6111
WebDisp 6117
WebDisp namespace 6117
Classes 6120
Constants 6247
Functions 6233
Interfaces 6231
Structs, Records, Enums 6239
Types 6240
WebDisp.AdapterRequestParamsAsStrings 6233
WebDisp.AdapterRequestParamsAsStrings function 6233
WebDisp.DispatchPageName 6233
WebDisp.DispatchPageName function 6233
WebDisp.EAdapterDispatcherException 6120
WebDisp.EncodeParamNameValues 6234
WebDisp.EncodeParamNameValues function 6234
WebDisp.EncodeParamsAsHTTPQuery 6234
WebDisp.EncodeParamsAsHTTPQuery function 6234
WebDisp.EPageDispatcherException 6121
WebDisp.ESessionException 6122
WebDisp.ExpandScriptException 6234
WebDisp.ExpandScriptException function 6234
WebDisp.ExtractParamNameValues 6235
WebDisp.ExtractParamNameValues function 6235
WebDisp.FindDefaultAppPageName 6235
WebDisp.FindDefaultAppPageName function 6235
WebDisp.IWebSessionAccess 6231
WebDisp.IWebSessionAccess.GetSessionsService 6232
WebDisp.IWebSessionAccess.GetWebSession 6232
WebDisp.PageNameOfRequest 6235
WebDisp.PageNameOfRequest function 6235
WebDisp.RaiseDispatchBlankPageNameException 6236
WebDisp.RaiseDispatchBlankPageNameException function 6236
WebDisp.RaisePageDoesNotProvideContent 6236
WebDisp.RaisePageDoesNotProvideContent function 6236
WebDisp.RaisePageNotFound 6236
WebDisp.RaisePageNotFound function 6236
WebDisp.RedirectToPageName 6237
WebDisp.RedirectToPageName function 6237
WebDisp.RegisterAdapterRequestIdentifier 6237
WebDisp.RegisterAdapterRequestIdentifier function 6237
WebDisp.SafeRedirect 6237
WebDisp.SafeRedirect function 6237
WebDisp.sFailurePageParamName 6247
WebDisp.sFailurePageParamName constant 6247
WebDisp.sPageParamName 6247
WebDisp.sPageParamName constant 6247
WebDisp.sRequestIDParamName 6248
WebDisp.sRequestIDParamName constant 6248
WebDisp.sSessionCookieName 6248
WebDisp.sSessionCookieName constant 6248
WebDisp.sSuccessPageParamName 6248
WebDisp.sSuccessPageParamName constant 6248
WebDisp.TActivateWebModuleEvent 6240
WebDisp.TActivateWebModuleEvent type 6240
WebDisp.TAdapterDispatcher 6122
WebDisp.TAdapterDispatcher.OnActionRequestNotHandled 6124
WebDisp.TAdapterDispatcher.OnAfterDispatchAction 6125
WebDisp.TAdapterDispatcher.OnAfterDispatchImage 6125
WebDisp.TAdapterDispatcher.OnBeforeDispatchAction 6125
WebDisp.TAdapterDispatcher.OnBeforeDispatchImage 6126
WebDisp.TAdapterDispatcher.OnGetAdapterRequestParams 6126
WebDisp.TAdapterDispatcher.OnImageRequestNotHandled 6127
WebDisp.TAdapterDispatchParams 6128
WebDisp.TAdapterDispatchParams.Create 6130
WebDisp.TAdapterDispatchParams.Destroy 6130
WebDisp.TAdapterDispatchParams.GetFailurePage 6131
WebDisp.TAdapterDispatchParams.GetHandler 6131
WebDisp.TAdapterDispatchParams.GetObjectIdentifier 6132
WebDisp.TAdapterDispatchParams.GetParams 6132

WebDisp.TAdapterDispatchParams.GetRequestIdentifier 6132
WebDisp.TAdapterDispatchParams.GetSuccessPage 6132
WebDisp.TAdapterDispatchParams.Handler 6133
WebDisp.TAdapterDispatchParams.ObjectIdentifier 6133
WebDisp.TAdapterDispatchParams.RequestIdentifier 6134
WebDisp.TAdapterItemRequestParams 6134
WebDisp.TAdapterItemRequestParams.Create 6136
WebDisp.TAdapterItemRequestParams.Destroy 6137
WebDisp.TAdapterItemRequestParams.FailurePage 6139
WebDisp.TAdapterItemRequestParams.GetFailurePage 6137
WebDisp.TAdapterItemRequestParams.GetParamValues 6137
WebDisp.TAdapterItemRequestParams.GetSuccessPage 6138
WebDisp.TAdapterItemRequestParams.ParamValues 6139
WebDisp.TAdapterItemRequestParams.SetFailurePage 6138
WebDisp.TAdapterItemRequestParams.SetSuccessPage 6138
WebDisp.TAdapterItemRequestParams.SuccessPage 6140
WebDisp.TBaseRequestHandler 6140
WebDisp.TBaseRequestHandler.AppDispatcher 6142
WebDisp.TBaseRequestHandler.HandleRequest 6141
WebDisp.TBaseRequestHandler.ImplHandleRequest 6141
WebDisp.TBaseRequestHandler.SetAppDispatcher 6142
WebDisp.TCanViewPageEvent 6240
WebDisp.TCanViewPageEvent type 6240
WebDisp.TCustomAdapterDispatcher 6142
WebDisp.TCustomAdapterDispatcher.AddSpecialParams 6144
WebDisp.TCustomAdapterDispatcher.DoAdapterRequestNotHandled
6145
WebDisp.TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest
6146
WebDisp.TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest
6146
WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue
6147
WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF
6147
WebDisp.TCustomAdapterDispatcher.GetAdapterItemRequestParams
6148
WebDisp.TCustomAdapterDispatcher.GetDispatchParams 6148
WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue
6149
WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF
6149
WebDisp.TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams
6150
WebDisp.TCustomAdapterDispatcher.ImplHandleRequest 6150
WebDisp.TCustomAdapterDispatcher.OnActionRequestNotHandled
6151
WebDisp.TCustomAdapterDispatcher.OnAfterDispatchAction
6152
WebDisp.TCustomAdapterDispatcher.OnAfterDispatchImage
6152
WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchAction
6153
WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchImage
6153
WebDisp.TCustomAdapterDispatcher.OnGetAdapterRequestParams
6154
WebDisp.TCustomAdapterDispatcher.OnImageRequestNotHandled
6154
WebDisp.TCustomLocateFileService 6155
WebDisp.TCustomLocateFileService.FindIncludeFile 6156
WebDisp.TCustomLocateFileService.FindStream 6157
WebDisp.TCustomLocateFileService.FindTemplateFile 6157
WebDisp.TCustomLocateFileService.GetFileStream 6158
WebDisp.TCustomLocateFileService.GetTemplateStream 6158
WebDisp.TCustomLocateFileService.ImplFindIncludeFile 6159
WebDisp.TCustomLocateFileService.ImplGetTemplateStream
6159
WebDisp.TCustomLocateFileService.OnFindIncludeFile 6160
WebDisp.TCustomLocateFileService.OnFindStream 6161
WebDisp.TCustomLocateFileService.OnFindTemplateFile 6161
WebDisp.TCustomPageDispatcher 6162
WebDisp.TCustomPageDispatcher.DefaultPage 6169
WebDisp.TCustomPageDispatcher.DispatchPageName 6164
WebDisp.TCustomPageDispatcher.EndUser 6164
WebDisp.TCustomPageDispatcher.GetCanView 6165
WebDisp.TCustomPageDispatcher.GetDefaultPageName 6165

WebDisp.TCustomPageDispatcher.GetLoginRequired 6165
WebDisp.TCustomPageDispatcher.GetRequestedPage 6166
WebDisp.TCustomPageDispatcher.ImplDispatchPageName 6166
WebDisp.TCustomPageDispatcher.ImplGetCanView 6166
WebDisp.TCustomPageDispatcher.ImplGetDefaultPageName 6166
WebDisp.TCustomPageDispatcher.ImplGetLoginRequired 6167
WebDisp.TCustomPageDispatcher.ImplHandleRequest 6167
WebDisp.TCustomPageDispatcher.ImplPageNameOfRequest 6167
WebDisp.TCustomPageDispatcher.ImplRedirectToPageName 6167
WebDisp.TCustomPageDispatcher.IsLoggedIn 6168
WebDisp.TCustomPageDispatcher.OnAfterDispatchPage 6170
WebDisp.TCustomPageDispatcher.OnAfterRedirectToPage 6171
WebDisp.TCustomPageDispatcher.OnBeforeDispatchPage 6171
WebDisp.TCustomPageDispatcher.OnBeforeRedirectToPage 6172
WebDisp.TCustomPageDispatcher.OnCanViewPage 6172
WebDisp.TCustomPageDispatcher.OnIsLoginRequired 6173
WebDisp.TCustomPageDispatcher.OnPageAccessDenied 6174
WebDisp.TCustomPageDispatcher.OnPageNotFound 6174
WebDisp.TCustomPageDispatcher.PageNameOfRequest 6168
WebDisp.TCustomPageDispatcher.ProduceLoginPage 6168
WebDisp.TCustomPageDispatcher.RedirectToPageName 6168
WebDisp.TCustomPageDispatcher.ValidatePage 6169
WebDisp.TCustomWebAppComponents 6175
WebDisp.TCustomWebAppComponents.AdapterDispatcher 6185
WebDisp.TCustomWebAppComponents.ApplicationAdapter 6186
WebDisp.TCustomWebAppComponents.CreateContext 6178
WebDisp.TCustomWebAppComponents.DispatchActions 6186
WebDisp.TCustomWebAppComponents.EndUserAdapter 6187
WebDisp.TCustomWebAppComponents.GetAdapterDispatcher 6179
WebDisp.TCustomWebAppComponents.GetLocateFileService 6179
WebDisp.TCustomWebAppComponents.GetPageDispatcher 6179
WebDisp.TCustomWebAppComponents.GetWebApplicationInfo 6180
WebDisp.TCustomWebAppComponents.GetWebEndUser 6180
WebDisp.TCustomWebAppComponents.GetWebUserList 6180
WebDisp.TCustomWebAppComponents.ImplFinishContext 6181
WebDisp.TCustomWebAppComponents.ImplHandleException 6181
WebDisp.TCustomWebAppComponents.ImplHandleRequest 6182
WebDisp.TCustomWebAppComponents.ImplInitContext 6182
WebDisp.TCustomWebAppComponents.InitModule 6183
WebDisp.TCustomWebAppComponents.InitModules 6184
WebDisp.TCustomWebAppComponents.ListRequestHandlers 6184
WebDisp.TCustomWebAppComponents.LocateFileService 6187
WebDisp.TCustomWebAppComponents.Notification 6185
WebDisp.TCustomWebAppComponents.OnException 6189
WebDisp.TCustomWebAppComponents.PageDispatcher 6188
WebDisp.TCustomWebAppComponents.Sessions 6188
WebDisp.TCustomWebAppComponents.UserListService 6189
WebDisp.TDispatchActionEvent 6241
WebDisp.TDispatchActionEvent type 6241
WebDisp.TDispatchActionHandledEvent 6241
WebDisp.TDispatchActionHandledEvent type 6241
WebDisp.TDispatchImageEvent 6241
WebDisp.TDispatchImageEvent type 6241
WebDisp.TDispatchImageHandledEvent 6242
WebDisp.TDispatchImageHandledEvent type 6242
WebDisp.TDispatchPageEvent 6242
WebDisp.TDispatchPageEvent type 6242
WebDisp.TDispatchPageHandledEvent 6242
WebDisp.TDispatchPageHandledEvent type 6242
WebDisp.TDispatchPageParamsHandledEvent 6243
WebDisp.TDispatchPageParamsHandledEvent type 6243
WebDisp.TGetAdapterRequestParamsEvent 6243
WebDisp.TGetAdapterRequestParamsEvent type 6243
WebDisp.TInvalidWebSession 6190
WebDisp.TInvalidWebSession.Create 6191
WebDisp.TInvalidWebSession.GetSessionID 6191
WebDisp.TInvalidWebSession.GetTimeoutMinutes 6192

WebDisp.TInvalidWebSession.GetValue 6192
WebDisp.TInvalidWebSession.SetTimeoutMinutes 6193
WebDisp.TInvalidWebSession.SetValue 6193
WebDisp.TInvalidWebSession.Terminate 6193
WebDisp.TInvalidWebSession.UpdateResponse 6194
WebDisp.TLocateFileService 6194
WebDisp.TLocateFileService.OnFindIncludeFile 6195
WebDisp.TLocateFileService.OnFindStream 6196
WebDisp.TLocateFileService.OnFindTemplateFile 6196
WebDisp.TLocateFileServiceFindFileEvent 6243
WebDisp.TLocateFileServiceFindFileEvent type 6243
WebDisp.TLocateFileServiceFindStreamEvent 6244
WebDisp.TLocateFileServiceFindStreamEvent type 6244
WebDisp.TLoginPagedHandledEvent 6244
WebDisp.TLoginPagedHandledEvent type 6244
WebDisp.TLoginRequiredEvent 6244
WebDisp.TLoginRequiredEvent type 6244
WebDisp.TMultiModuleWebAppServices 6197
WebDisp.TMultiModuleWebAppServices.Access_Request 6199
WebDisp.TMultiModuleWebAppServices.Access_Response 6199
WebDisp.TMultiModuleWebAppServices.CreateContext 6200
WebDisp.TMultiModuleWebAppServices.DoAfterDispatch 6200
WebDisp.TMultiModuleWebAppServices.DoBeforeDispatch 6200
WebDisp.TMultiModuleWebAppServices.DoneModule 6201
WebDisp.TMultiModuleWebAppServices.FinishContext 6201
WebDisp.TMultiModuleWebAppServices.GetExceptionHandler 6202
WebDisp.TMultiModuleWebAppServices.HandleException 6202
WebDisp.TMultiModuleWebAppServices.HandleRequest 6203
WebDisp.TMultiModuleWebAppServices.ImplFinishContext 6203
WebDisp.TMultiModuleWebAppServices.ImplHandleException 6204
WebDisp.TMultiModuleWebAppServices.ImplHandleRequest 6204
WebDisp.TMultiModuleWebAppServices.ImplInitContext 6205
WebDisp.TMultiModuleWebAppServices.InitContext 6205
WebDisp.TMultiModuleWebAppServices.InitModule 6206
WebDisp.TMultiModuleWebAppServices.InitModules 6207
WebDisp.TMultiModuleWebAppServices.ModuleAdded 6207
WebDisp.TMultiModuleWebAppServices.OnAfterDispatch 6209
WebDisp.TMultiModuleWebAppServices.OnBeforeDispatch 6209
WebDisp.TMultiModuleWebAppServices.Request 6208
WebDisp.TMultiModuleWebAppServices.Response 6208
WebDisp.TMultiModuleWebAppServices.WebModules 6208
WebDisp.TPageAccessDenied 6239
WebDisp.TPageAccessDenied enumeration 6239
WebDisp.TPageAccessDeniedEvent 6245
WebDisp.TPageAccessDeniedEvent type 6245
WebDisp.TPageDispatcher 6210
WebDisp.TPageDispatcher.DefaultPage 6212
WebDisp.TPageDispatcher.OnAfterDispatchPage 6213
WebDisp.TPageDispatcher.OnBeforeDispatchPage 6213
WebDisp.TPageDispatcher.OnCanViewPage 6214
WebDisp.TPageDispatcher.OnIsLoginRequired 6215
WebDisp.TPageDispatcher.OnPageAccessDenied 6215
WebDisp.TPageDispatcher.OnPageNotFound 6216
WebDisp.TWebAppComponents 6217
WebDisp.TWebAppComponents.AdapterDispatcher 6219
WebDisp.TWebAppComponents.ApplicationAdapter 6220
WebDisp.TWebAppComponents.DispatchActions 6220
WebDisp.TWebAppComponents.EndUserAdapter 6221
WebDisp.TWebAppComponents.LocateFileService 6221
WebDisp.TWebAppComponents.OnAfterDispatch 6223
WebDisp.TWebAppComponents.OnBeforeDispatch 6224
WebDisp.TWebAppComponents.OnException 6224
WebDisp.TWebAppComponents.PageDispatcher 6222
WebDisp.TWebAppComponents.Sessions 6222
WebDisp.TWebAppComponents.UserListService 6223
WebDisp.TWebRequest 6245
WebDisp.TWebRequest type 6245
WebDisp.TWebResponse 6245
WebDisp.TWebResponse type 6245
WebDisp.TWebSession 6225
WebDisp.TWebSession.Create 6226
WebDisp.TWebSession.GetSessionID 6227
WebDisp.TWebSession.GetTimeoutMinutes 6227
WebDisp.TWebSession.GetValue 6228

WebDisp.TWebSession.SetTimeoutMinutes 6228
WebDisp.TWebSession.SetValue 6228
WebDisp.TWebSession.Terminate 6229
WebDisp.TWebSession.UpdateResponse 6229
WebFact 6249
WebFact namespace 6249
Classes 6250
Structs, Records, Enums 6275
Types 6276
WebFact.TBaseWebModuleFactory 6250
WebFact.TBaseWebModuleFactory.Create 6251
WebFact.TBaseWebModuleFactory.GetCacheMode 6252
WebFact.TBaseWebModuleFactory.GetComponentClass 6252
WebFact.TBaseWebModuleFactory.GetCreateMode 6253
WebFact.TBaseWebModuleFactory.GetIsAppModule 6253
WebFact.TBaseWebModuleFactory.GetModule 6253
WebFact.TBaseWebModuleFactory.GetModuleName 6254
WebFact.TBaseWebModuleFactory.PreventDestruction 6254
WebFact.TBaseWebPageInfo 6255
WebFact.TBaseWebPageInfo.Create 6256
WebFact.TBaseWebPageInfo.GetIsLoginRequired 6257
WebFact.TBaseWebPageInfo.GetIsPublished 6257
WebFact.TBaseWebPageInfo.GetPageDescription 6257
WebFact.TBaseWebPageInfo.GetPageFile 6258
WebFact.TBaseWebPageInfo.GetPageHREF 6258
WebFact.TBaseWebPageInfo.GetPageName 6258
WebFact.TBaseWebPageInfo.GetPageTitle 6259
WebFact.TBaseWebPageInfo.GetViewAccess 6259
WebFact.TBaseWebPageInfo.SetFactory 6260
WebFact.TBaseWebPageModuleFactory 6260
WebFact.TBaseWebPageModuleFactory.Create 6261
WebFact.TBaseWebPageModuleFactory.GetCacheMode 6262
WebFact.TBaseWebPageModuleFactory.GetComponentClass 6262
WebFact.TBaseWebPageModuleFactory.GetCreateMode 6263
WebFact.TBaseWebPageModuleFactory.GetIsAppModule 6263
WebFact.TBaseWebPageModuleFactory.GetModule 6263
WebFact.TBaseWebPageModuleFactory.GetModuleName 6264
WebFact.TBaseWebPageModuleFactory.PreventDestruction 6264
WebFact.TWebAppDataModuleFactory 6265
WebFact.TWebAppDataModuleFactory.Create 6266
WebFact.TWebAppDataModuleFactory.GetIsAppModule 6266
WebFact.TWebAppPageModuleFactory 6267
WebFact.TWebAppPageModuleFactory.Create 6268
WebFact.TWebAppPageModuleFactory.GetIsAppModule 6269
WebFact.TWebDataModuleFactory 6269
WebFact.TWebDataModuleFactory.Create 6270
WebFact.TWebPageAccess 6276
WebFact.TWebPageAccess type 6276
WebFact.TWebPageAccessFlags 6275
WebFact.TWebPageAccessFlags enumeration 6275
WebFact.TWebPageInfo 6271
WebFact.TWebPageInfo.Create 6272
WebFact.TWebPageModuleFactory 6273
WebFact.TWebPageModuleFactory.Create 6274
WebForm 6277
WebForm namespace 6277
Classes 6280
Constants 6496
Functions 6485
Interfaces 6476
Structs, Records, Enums 6488
Types 6494
WebForm.CalcDisplayFieldHTMLElementType 6485
WebForm.CalcDisplayFieldHTMLElementType function 6485
WebForm.CalcDisplayFieldViewMode 6485
WebForm.CalcDisplayFieldViewMode function 6485
WebForm.FindVariablesContainer 6486
WebForm.FindVariablesContainer function 6486
WebForm.FindVariablesContainerInParent 6486
WebForm.FindVariablesContainerInParent function 6486
WebForm.GetAdapterModeOfAdapterInParent 6486
WebForm.GetAdapterModeOfAdapterInParent function 6486
WebForm.IAdapterDisplay 6476
WebForm.IGetAdapterFormLayout 6476
WebForm.IGetAdapterFormLayout.GetAdapterFormLayout 6477
WebForm.IGetAdapterModeOfDisplay 6477
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter

6478
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay 6478
WebForm.IGetAdapterOfDisplay 6478
WebForm.IGetAdapterOfDisplay.GetAdapter 6479
WebForm.IGetAdapterVariableName 6479
WebForm.IGetAdapterVariableName.AdapterVariableName 6481
WebForm.IGetAdapterVariableName.GetAdapterVariableName 6480
WebForm.IGetControlRequirements 6481
WebForm.IGetControlRequirements.GetControlRequirements 6482
WebForm.IGetFormRequirements 6482
WebForm.IGetFormRequirements.GetFormRequirements 6483
WebForm.IGetVariablesContainerOfDisplay 6483
WebForm.IGetVariablesContainerOfDisplay.GetVariablesContainer 6484
WebForm.MakeAdapterHTMLName 6487
WebForm.MakeAdapterHTMLName function 6487
WebForm.MakeAdapterVariableName 6487
WebForm.MakeAdapterVariableName function 6487
WebForm.sDisplayText 6496
WebForm.sDisplayText constant 6496
WebForm.sFieldText 6496
WebForm.sFieldText constant 6496
WebForm.sFormGet 6497
WebForm.sFormGet constant 6497
WebForm.sFormPost 6497
WebForm.sFormPost constant 6497
WebForm.sMultipartForm 6497
WebForm.sMultipartForm constant 6497
WebForm.TAbstractControlRequirements 6281
WebForm.TAbstractControlRequirements.DefineFunction 6282
WebForm.TAbstractControlRequirements.DefineVariable 6282
WebForm.TAbstractControlRequirements.IsFunctionDefined 6283
WebForm.TAbstractControlRequirements.IsVariableDefined 6283
WebForm.TAbstractControlRequirements.RequiresMultipartForm 6284
WebForm.TAbstractFormRequirements 6284
WebForm.TAbstractFormRequirements.DefineFunction 6285
WebForm.TAbstractFormRequirements.DefineVariable 6285
WebForm.TAbstractFormRequirements.IsFunctionDefined 6285
WebForm.TAbstractFormRequirements.IsVariableDefined 6286
WebForm.TActionButtonHideOption 6488
WebForm.TActionButtonHideOption enumeration 6488
WebForm.TActionButtonHideOptions 6494
WebForm.TActionButtonHideOptions type 6494
WebForm.TAdapterActionButton 6286
WebForm.TAdapterActionButton.ActionName 6288
WebForm.TAdapterActionButton.Align 6288
WebForm.TAdapterActionButton.Caption 6289
WebForm.TAdapterActionButton.Custom 6289
WebForm.TAdapterActionButton.DisplayColumns 6290
WebForm.TAdapterActionButton.DisplayType 6290
WebForm.TAdapterActionButton.ErrorPageName 6291
WebForm.TAdapterActionButton.HideOptions 6291
WebForm.TAdapterActionButton.ImageHeight 6291
WebForm.TAdapterActionButton.ImageProducer 6292
WebForm.TAdapterActionButton.ImageWidth 6292
WebForm.TAdapterActionButton.PageName 6293
WebForm.TAdapterActionButton.Style 6293
WebForm.TAdapterActionButton.StyleRule 6293
WebForm.TAdapterActionButton.VAlign 6294
WebForm.TAdapterActionsListHelper 6294
WebForm.TAdapterActionsListHelper.AddDefaultActions 6301
WebForm.TAdapterActionsListHelper.Create 6296
WebForm.TAdapterActionsListHelper.FindAction 6297
WebForm.TAdapterActionsListHelper.GetActionsList 6297
WebForm.TAdapterActionsListHelper.GetAddAllFieldsItem 6298
WebForm.TAdapterActionsListHelper.GetAddFieldsDlgCaption 6298
WebForm.TAdapterActionsListHelper.GetAddFieldsItem 6299
WebForm.TAdapterActionsListHelper.GetNewFieldPrefix 6299
WebForm.TAdapterActionsListHelper.GetVisibleActions 6300
WebForm.TAdapterActionsListHelper.IsActionInUse 6300
WebForm.TAdapterCommandColumn 6301
WebForm.TAdapterCommandColumn.AddDefaultCommands

6304
WebForm.TAdapterCommandColumn.Align 6304
WebForm.TAdapterCommandColumn.Caption 6304
WebForm.TAdapterCommandColumn.CaptionAttributes 6305
WebForm.TAdapterCommandColumn.Custom 6305
WebForm.TAdapterCommandColumn.DisplayColumns 6306
WebForm.TAdapterCommandColumn.Style 6306
WebForm.TAdapterCommandColumn.StyleRule 6306
WebForm.TAdapterCommandColumn.VAlign 6307
WebForm.TAdapterCommandGroup 6307
WebForm.TAdapterCommandGroup.AddDefaultCommands 6310
WebForm.TAdapterCommandGroup.Custom 6310
WebForm.TAdapterCommandGroup.DisplayComponent 6311
WebForm.TAdapterCommandGroup.Style 6311
WebForm.TAdapterCommandGroup.StyleRule 6311
WebForm.TAdapterControlGroup 6312
WebForm.TAdapterControlGroup.Adapter 6322
WebForm.TAdapterControlGroup.AdapterChange 6315
WebForm.TAdapterControlGroup.AdapterMode 6323
WebForm.TAdapterControlGroup.AddDefaultFields 6323
WebForm.TAdapterControlGroup.AddNotify 6315
WebForm.TAdapterControlGroup.Create 6313
WebForm.TAdapterControlGroup.CreateDefaultFields 6316
WebForm.TAdapterControlGroup.Destroy 6314
WebForm.TAdapterControlGroup.FindField 6316
WebForm.TAdapterControlGroup.GetAdapter 6317
WebForm.TAdapterControlGroup.GetAdapterModeOfAdapter 6317
WebForm.TAdapterControlGroup.GetAdapterModeOfDisplay 6317
WebForm.TAdapterControlGroup.GetAdapterVariableName 6318
WebForm.TAdapterControlGroup.GetControlRequirements 6318
WebForm.TAdapterControlGroup.GetDesigntimeWarnings 6318
WebForm.TAdapterControlGroup.GetFieldsList 6319
WebForm.TAdapterControlGroup.GetVisibleFields 6319
WebForm.TAdapterControlGroup.ImplGetControlRequirements 6320
WebForm.TAdapterControlGroup.IsFieldInUse 6320
WebForm.TAdapterControlGroup.Notification 6320
WebForm.TAdapterControlGroup.NotifyAdapterChange 6321
WebForm.TAdapterControlGroup.RemoveNotify 6321
WebForm.TAdapterControlGroup.SetAdapter 6322
WebForm.TAdapterControlGroup.VisibleFields 6324
WebForm.TAdapterDataDisplay 6324
WebForm.TAdapterDataDisplay.Align 6332
WebForm.TAdapterDataDisplay.Create 6326
WebForm.TAdapterDataDisplay.FieldConditions 6327
WebForm.TAdapterDataDisplay.FieldName 6332
WebForm.TAdapterDataDisplay.FindAssociatedField 6327
WebForm.TAdapterDataDisplay.FormatCaption 6327
WebForm.TAdapterDataDisplay.FormatColumnData 6327
WebForm.TAdapterDataDisplay.FormatColumnHeading 6327
WebForm.TAdapterDataDisplay.GetAdapterModeOfAdapter 6328
WebForm.TAdapterDataDisplay.GetAdapterVariableName 6328
WebForm.TAdapterDataDisplay.GetCaption 6328
WebForm.TAdapterDataDisplay.GetControlRequirements 6328
WebForm.TAdapterDataDisplay.GetDesigntimeWarnings 6329
WebForm.TAdapterDataDisplay.GetFieldName 6329
WebForm.TAdapterDataDisplay.GetVariablesContainer 6329
WebForm.TAdapterDataDisplay.HideOptions 6332
WebForm.TAdapterDataDisplay.ImplContent 6329
WebForm.TAdapterDataDisplay.ImplFormatColumnData 6330
WebForm.TAdapterDataDisplay.ImplFormatColumnHeading 6330
WebForm.TAdapterDataDisplay.ImplGetControlRequirements 6330
WebForm.TAdapterDataDisplay.ImplRestoreDefaults 6330
WebForm.TAdapterDataDisplay.RestoreDefaults 6331
WebForm.TAdapterDataDisplay.SetCaption 6331
WebForm.TAdapterDataDisplay.SetFieldName 6331
WebForm.TAdapterDataDisplay.VAlign 6332
WebForm.TAdapterDataDisplayClass 6494
WebForm.TAdapterDataDisplayClass type 6494
WebForm.TAdapterDisplayColumn 6333
WebForm.TAdapterDisplayColumn.Align 6335
WebForm.TAdapterDisplayColumn.Caption 6336
WebForm.TAdapterDisplayColumn.CaptionAttributes 6336
WebForm.TAdapterDisplayColumn.Custom 6336

WebForm.TAdapterDisplayColumn.DisplayType 6337
WebForm.TAdapterDisplayColumn.FieldName 6337
WebForm.TAdapterDisplayColumn.HideOptions 6338
WebForm.TAdapterDisplayColumn.ImageHeight 6338
WebForm.TAdapterDisplayColumn.ImageWidth 6338
WebForm.TAdapterDisplayColumn.Style 6338
WebForm.TAdapterDisplayColumn.StyleRule 6339
WebForm.TAdapterDisplayColumn.VAlign 6339
WebForm.TAdapterDisplayField 6339
WebForm.TAdapterDisplayField.Align 6343
WebForm.TAdapterDisplayField.Caption 6343
WebForm.TAdapterDisplayField.CaptionAttributes 6343
WebForm.TAdapterDisplayField.CaptionPosition 6344
WebForm.TAdapterDisplayField.Custom 6344
WebForm.TAdapterDisplayField.DisplayColumns 6344
WebForm.TAdapterDisplayField.DisplayRows 6345
WebForm.TAdapterDisplayField.DisplayType 6345
WebForm.TAdapterDisplayField.DisplayWidth 6345
WebForm.TAdapterDisplayField.FieldName 6346
WebForm.TAdapterDisplayField.HideOptions 6346
WebForm.TAdapterDisplayField.ImageHeight 6346
WebForm.TAdapterDisplayField.ImageWidth 6347
WebForm.TAdapterDisplayField.InputType 6347
WebForm.TAdapterDisplayField.MaxLength 6347
WebForm.TAdapterDisplayField.SelectRows 6348
WebForm.TAdapterDisplayField.Style 6348
WebForm.TAdapterDisplayField.StyleRule 6348
WebForm.TAdapterDisplayField.TextAreaWrap 6349
WebForm.TAdapterDisplayField.VAlign 6349
WebForm.TAdapterDisplayField.ViewMode 6349
WebForm.TAdapterDisplayReferenceGroup 6349
WebForm.TAdapterDisplayReferenceGroup.AdapterChange 6352
WebForm.TAdapterDisplayReferenceGroup.Create 6351
WebForm.TAdapterDisplayReferenceGroup.Destroy 6351
WebForm.TAdapterDisplayReferenceGroup.DisplayComponent 6355
WebForm.TAdapterDisplayReferenceGroup.DisplayComponentChange 6352
WebForm.TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics 6353
WebForm.TAdapterDisplayReferenceGroup.GetDesigntimeWarnings 6353
WebForm.TAdapterDisplayReferenceGroup.NotifyAdapterChange 6353
WebForm.TAdapterDisplayReferenceGroup.OnDisplayComponentChange 6354
WebForm.TAdapterDisplayReferenceGroup.SetWebDisplayComponent 6354
WebForm.TAdapterEditColumn 6355
WebForm.TAdapterEditColumn.Align 6358
WebForm.TAdapterEditColumn.Caption 6358
WebForm.TAdapterEditColumn.CaptionAttributes 6359
WebForm.TAdapterEditColumn.CaptionPosition 6359
WebForm.TAdapterEditColumn.Custom 6360
WebForm.TAdapterEditColumn.DisplayColumns 6360
WebForm.TAdapterEditColumn.DisplayRows 6360
WebForm.TAdapterEditColumn.DisplayType 6361
WebForm.TAdapterEditColumn.DisplayWidth 6361
WebForm.TAdapterEditColumn.FieldName 6361
WebForm.TAdapterEditColumn.HideOptions 6362
WebForm.TAdapterEditColumn.ImageHeight 6362
WebForm.TAdapterEditColumn.ImageWidth 6362
WebForm.TAdapterEditColumn.InputType 6362
WebForm.TAdapterEditColumn.MaxLength 6363
WebForm.TAdapterEditColumn.SelectRows 6363
WebForm.TAdapterEditColumn.Style 6363
WebForm.TAdapterEditColumn.StyleRule 6364
WebForm.TAdapterEditColumn.TextAreaWrap 6364
WebForm.TAdapterEditColumn.VAlign 6364
WebForm.TAdapterEditColumn.ViewMode 6365
WebForm.TAdapterErrorList 6365
WebForm.TAdapterErrorList.Adapter 6367
WebForm.TAdapterFieldGroup 6367
WebForm.TAdapterFieldGroup.Adapter 6370
WebForm.TAdapterFieldGroup.AdapterMode 6371
WebForm.TAdapterFieldGroup.AddDefaultFields 6371

WebForm.TAdapterFieldGroup.Custom 6372
WebForm.TAdapterFieldGroup.Style 6372
WebForm.TAdapterFieldGroup.StyleRule 6373
WebForm.TAdapterForm 6373
WebForm.TAdapterForm.Custom 6375
WebForm.TAdapterForm.FormMethod 6375
WebForm.TAdapterForm.Style 6376
WebForm.TAdapterForm.StyleRule 6376
WebForm.TAdapterFormLayout 6376
WebForm.TAdapterFormLayout.BreakButtons 6385
WebForm.TAdapterFormLayout.ChildTableIndent 6385
WebForm.TAdapterFormLayout.ColumnCount 6385
WebForm.TAdapterFormLayout.Create 6378
WebForm.TAdapterFormLayout.EndButtons 6379
WebForm.TAdapterFormLayout.EndField 6380
WebForm.TAdapterFormLayout.EndFields 6380
WebForm.TAdapterFormLayout.EndLayout 6380
WebForm.TAdapterFormLayout.EndRow 6380
WebForm.TAdapterFormLayout.EndTable 6380
WebForm.TAdapterFormLayout.FieldIndent 6386
WebForm.TAdapterFormLayout.GetAdapterFormLayout 6381
WebForm.TAdapterFormLayout.GetChildTableIndent 6381
WebForm.TAdapterFormLayout.GetFieldIndent 6381
WebForm.TAdapterFormLayout.GetTableIndent 6381
WebForm.TAdapterFormLayout.ImplLayoutButton 6381
WebForm.TAdapterFormLayout.ImplLayoutField 6382
WebForm.TAdapterFormLayout.ImplLayoutLabelAndField 6382
WebForm.TAdapterFormLayout.ImplLayoutTable 6383
WebForm.TAdapterFormLayout.NextColumn 6384
WebForm.TAdapterFormLayout.StartButtons 6384
WebForm.TAdapterFormLayout.StartFields 6384
WebForm.TAdapterFormLayout.StartRow 6384
WebForm.TAdapterFormLayout.StartTable 6384
WebForm.TAdapterFormLayout.TableHeader 6386
WebForm.TAdapterFormLayout.TableIndent 6386
WebForm.TAdapterGrid 6386
WebForm.TAdapterGrid.Adapter 6389
WebForm.TAdapterGrid.AdapterMode 6390
WebForm.TAdapterGrid.AddDefaultFields 6390
WebForm.TAdapterGrid.HeadingAttributes 6391
WebForm.TAdapterGrid.RowAttributes 6391
WebForm.TAdapterGrid.TableAttributes 6392
WebForm.TColumnHTMLElementType 6489
WebForm.TColumnHTMLElementType enumeration 6489
WebForm.TCommandHTMLElementType 6489
WebForm.TCommandHTMLElementType enumeration 6489
WebForm.TControlRequirements 6392
WebForm.TControlRequirements.Create 6393
WebForm.TControlRequirements.DefineFunction 6394
WebForm.TControlRequirements.DefineVariable 6395
WebForm.TControlRequirements.Destroy 6394
WebForm.TControlRequirements.IsFunctionDefined 6395
WebForm.TControlRequirements.IsVariableDefined 6396
WebForm.TControlRequirements.NeedMultipartForm 6396
WebForm.TControlRequirements.RequiresMultipartForm 6396
WebForm.TCustomAdapterActionButton 6397
WebForm.TCustomAdapterActionButton.ActionName 6405
WebForm.TCustomAdapterActionButton.Align 6405
WebForm.TCustomAdapterActionButton.Create 6398
WebForm.TCustomAdapterActionButton.Destroy 6399
WebForm.TCustomAdapterActionButton.DisplayColumns 6405
WebForm.TCustomAdapterActionButton.DisplayType 6406
WebForm.TCustomAdapterActionButton.ErrorPageName 6406
WebForm.TCustomAdapterActionButton.FindAssociatedAction 6400
WebForm.TCustomAdapterActionButton.GetActionName 6400
WebForm.TCustomAdapterActionButton.GetAdapterModeOfAdapter 6400
WebForm.TCustomAdapterActionButton.GetAdapterVariableName 6400
WebForm.TCustomAdapterActionButton.GetCaption 6401
WebForm.TCustomAdapterActionButton.GetControlRequirements 6401
WebForm.TCustomAdapterActionButton.GetDesigntimeWarnings 6401
WebForm.TCustomAdapterActionButton.GetHTMLFormVarName 6401

WebForm.TCustomAdapterActionButton.GetLayoutAttributes 6402
WebForm.TCustomAdapterActionButton.HideOptions 6407
WebForm.TCustomAdapterActionButton.ImageHeight 6407
WebForm.TCustomAdapterActionButton.ImageProducer 6408
WebForm.TCustomAdapterActionButton.ImageWidth 6408
WebForm.TCustomAdapterActionButton.ImplContent 6402
WebForm.TCustomAdapterActionButton.ImplGetControlRequirements 6402
WebForm.TCustomAdapterActionButton.ImplRestoreDefaults 6403
WebForm.TCustomAdapterActionButton.Notification 6403
WebForm.TCustomAdapterActionButton.PageName 6408
WebForm.TCustomAdapterActionButton.RestoreDefaults 6403
WebForm.TCustomAdapterActionButton.SetActionName 6404
WebForm.TCustomAdapterActionButton.SetCaption 6404
WebForm.TCustomAdapterActionButton.VAlign 6409
WebForm.TCustomAdapterCommandColumn 6409
WebForm.TCustomAdapterCommandColumn.AdapterActionsListHelper 6416
WebForm.TCustomAdapterCommandColumn.AddDefaultCommands 6416
WebForm.TCustomAdapterCommandColumn.Align 6417
WebForm.TCustomAdapterCommandColumn.Caption 6417
WebForm.TCustomAdapterCommandColumn.CaptionAttributes 6418
WebForm.TCustomAdapterCommandColumn.Create 6411
WebForm.TCustomAdapterCommandColumn.Custom 6418
WebForm.TCustomAdapterCommandColumn.Destroy 6412
WebForm.TCustomAdapterCommandColumn.DisplayColumns 6418
WebForm.TCustomAdapterCommandColumn.FormatColumnData 6412
WebForm.TCustomAdapterCommandColumn.FormatColumnHeading 6413
WebForm.TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics 6413
WebForm.TCustomAdapterCommandColumn.GetLayoutAttributes 6413
WebForm.TCustomAdapterCommandColumn.ImplCanAddClass 6414
WebForm.TCustomAdapterCommandColumn.ImplContent 6414
WebForm.TCustomAdapterCommandColumn.NotifyAdapterChange 6415
WebForm.TCustomAdapterCommandColumn.Style 6419
WebForm.TCustomAdapterCommandColumn.StyleRule 6419
WebForm.TCustomAdapterCommandColumn.VAlign 6420
WebForm.TCustomAdapterCommandColumn.VisibleCommands 6420
WebForm.TCustomAdapterCommandGroup 6421
WebForm.TCustomAdapterCommandGroup.Adapter 6426
WebForm.TCustomAdapterCommandGroup.AdapterActionsListHelper 6426
WebForm.TCustomAdapterCommandGroup.AddDefaultCommands 6426
WebForm.TCustomAdapterCommandGroup.Create 6423
WebForm.TCustomAdapterCommandGroup.Custom 6426
WebForm.TCustomAdapterCommandGroup.Destroy 6423
WebForm.TCustomAdapterCommandGroup.GetAdapter 6424
WebForm.TCustomAdapterCommandGroup.GetAdapterFormLayout 6424
WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfAdapter 6424
WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfDisplay 6424
WebForm.TCustomAdapterCommandGroup.ImplCanAddClass 6425
WebForm.TCustomAdapterCommandGroup.ImplContent 6425
WebForm.TCustomAdapterCommandGroup.Style 6427
WebForm.TCustomAdapterCommandGroup.StyleRule 6427
WebForm.TCustomAdapterCommandGroup.VisibleCommands 6427
WebForm.TCustomAdapterDisplayColumn 6428
WebForm.TCustomAdapterDisplayColumn.CanAddClass 6430
WebForm.TCustomAdapterDisplayColumn.ControlContent 6431
WebForm.TCustomAdapterDisplayColumn.Create 6430
WebForm.TCustomAdapterDisplayColumn.DisplayType 6433

WebForm.TCustomAdapterDisplayColumn.FormatImage 6431
WebForm.TCustomAdapterDisplayColumn.FormatListDisplay 6431
WebForm.TCustomAdapterDisplayColumn.FormatTextDisplay 6431
WebForm.TCustomAdapterDisplayColumn.ImageHeight 6433
WebForm.TCustomAdapterDisplayColumn.ImageWidth 6434
WebForm.TCustomAdapterDisplayColumn.ImplCanAddClass 6432
WebForm.TCustomAdapterDisplayColumn.ImplFormatColumnHeading 6432
WebForm.TCustomAdapterDisplayColumn.ImplGetControlRequirements 6432
WebForm.TCustomAdapterDisplayColumn.ImplRestoreDefaults 6432
WebForm.TCustomAdapterDisplayColumn.IsColumn 6432
WebForm.TCustomAdapterDisplayField 6434
WebForm.TCustomAdapterDisplayField.AddAttributes 6437
WebForm.TCustomAdapterDisplayField.ControlContent 6437
WebForm.TCustomAdapterDisplayField.Create 6436
WebForm.TCustomAdapterDisplayField.DisplayColumns 6439
WebForm.TCustomAdapterDisplayField.DisplayRows 6439
WebForm.TCustomAdapterDisplayField.DisplayType 6439
WebForm.TCustomAdapterDisplayField.DisplayWidth 6440
WebForm.TCustomAdapterDisplayField.GetLayoutAttributes 6437
WebForm.TCustomAdapterDisplayField.ImageHeight 6440
WebForm.TCustomAdapterDisplayField.ImageWidth 6440
WebForm.TCustomAdapterDisplayField.ImplContent 6437
WebForm.TCustomAdapterDisplayField.ImplGetControlRequirements 6438
WebForm.TCustomAdapterDisplayField.ImplRestoreDefaults 6438
WebForm.TCustomAdapterDisplayField.InputType 6441
WebForm.TCustomAdapterDisplayField.MaxLength 6441
WebForm.TCustomAdapterDisplayField.SelectRows 6441
WebForm.TCustomAdapterDisplayField.TextAreaWrap 6441
WebForm.TCustomAdapterDisplayField.ViewMode 6442
WebForm.TCustomAdapterEditColumn 6442
WebForm.TCustomAdapterEditColumn.ImplContent 6444
WebForm.TCustomAdapterErrorList 6445
WebForm.TCustomAdapterErrorList.Adapter 6448
WebForm.TCustomAdapterErrorList.Content 6447
WebForm.TCustomAdapterErrorList.Create 6446
WebForm.TCustomAdapterErrorList.Destroy 6446
WebForm.TCustomAdapterErrorList.GetDesigntimeWarnings 6447
WebForm.TCustomAdapterErrorList.ImplContent 6448
WebForm.TCustomAdapterErrorList.Notification 6448
WebForm.TCustomAdapterFieldGroup 6449
WebForm.TCustomAdapterFieldGroup.Create 6451
WebForm.TCustomAdapterFieldGroup.Custom 6455
WebForm.TCustomAdapterFieldGroup.Destroy 6451
WebForm.TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics 6452
WebForm.TCustomAdapterFieldGroup.GetAdapterFormLayout 6452
WebForm.TCustomAdapterFieldGroup.GetAddAllFieldsItem 6453
WebForm.TCustomAdapterFieldGroup.GetAddFieldsDlgCaption 6453
WebForm.TCustomAdapterFieldGroup.GetAddFieldsItem 6453
WebForm.TCustomAdapterFieldGroup.GetLayoutAttributes 6453
WebForm.TCustomAdapterFieldGroup.GetNewFieldPrefix 6454
WebForm.TCustomAdapterFieldGroup.ImplCanAddClass 6454
WebForm.TCustomAdapterFieldGroup.ImplContent 6454
WebForm.TCustomAdapterFieldGroup.Style 6455
WebForm.TCustomAdapterFieldGroup.StyleRule 6456
WebForm.TCustomAdapterForm 6456
WebForm.TCustomAdapterForm.Create 6457
WebForm.TCustomAdapterForm.Custom 6461
WebForm.TCustomAdapterForm.Destroy 6458
WebForm.TCustomAdapterForm.FormMethod 6461
WebForm.TCustomAdapterForm.GetAdapterFormLayout 6458
WebForm.TCustomAdapterForm.GetFormRequirements 6458
WebForm.TCustomAdapterForm.GetHTMLFormName 6459
WebForm.TCustomAdapterForm.GetHTMLFormVarName 6459
WebForm.TCustomAdapterForm.GetLayoutAttributes 6459
WebForm.TCustomAdapterForm.ImplCanAddClass 6460
WebForm.TCustomAdapterForm.ImplContent 6460

WebForm.TCustomAdapterForm.Style 6462
WebForm.TCustomAdapterForm.StyleRule 6462
WebForm.TCustomAdapterGrid 6462
WebForm.TCustomAdapterGrid.Create 6465
WebForm.TCustomAdapterGrid.Destroy 6465
WebForm.TCustomAdapterGrid.GetAdapterDisplayCharacteristics 6466
WebForm.TCustomAdapterGrid.GetAdapterFormLayout 6466
WebForm.TCustomAdapterGrid.GetAddAllFieldsItem 6466
WebForm.TCustomAdapterGrid.GetAddFieldsDlgCaption 6467
WebForm.TCustomAdapterGrid.GetAddFieldsItem 6467
WebForm.TCustomAdapterGrid.GetNewFieldPrefix 6467
WebForm.TCustomAdapterGrid.HeadingAttributes 6468
WebForm.TCustomAdapterGrid.ImplCanAddClass 6467
WebForm.TCustomAdapterGrid.ImplContent 6468
WebForm.TCustomAdapterGrid.RowAttributes 6469
WebForm.TCustomAdapterGrid.TableAttributes 6469
WebForm.TDesigntimeWarnings 6470
WebForm.TDesigntimeWarnings.AddObject 6471
WebForm.TDesigntimeWarnings.AddString 6471
WebForm.TDesigntimeWarnings.Create 6470
WebForm.TDesigntimeWarnings.Destroy 6470
WebForm.TDesigntimeWarnings.Strings 6472
WebForm.TDisplayFieldHideOption 6490
WebForm.TDisplayFieldHideOption enumeration 6490
WebForm.TDisplayFieldHideOptions 6494
WebForm.TDisplayFieldHideOptions type 6494
WebForm.TDisplayFieldHTMLElementType 6491
WebForm.TDisplayFieldHTMLElementType enumeration 6491
WebForm.TDisplayFieldViewMode 6491
WebForm.TDisplayFieldViewMode enumeration 6491
WebForm.TGeneratedFunctions 6472
WebForm.TGeneratedFunctions.Add 6473
WebForm.TGeneratedFunctions.Create 6473
WebForm.TGeneratedFunctions.Destroy 6473
WebForm.TGeneratedFunctions.Exists 6474
WebForm.TGeneratedFunctions.ItemCount 6474
WebForm.TGeneratedFunctions.Names 6474
WebForm.TGeneratedFunctions.Script 6474
WebForm.TInputFieldHTMLElementType 6492
WebForm.TInputFieldHTMLElementType enumeration 6492
WebForm.TLayoutState 6493
WebForm.TLayoutState enumeration 6493
weBinding enumeration member 7137
WebModu 6499
WebModu namespace 6499
  Classes 6501
  Functions 6578
  Types 6580
WebModu.EWebModuleException 6501
WebModu.FindNamedModuleVariable 6578
WebModu.FindNamedModuleVariable function 6578
WebModu.GetModuleVariable 6578
WebModu.GetModuleVariable function 6578
WebModu.GetModuleVariableCount 6578
WebModu.GetModuleVariableCount function 6578
WebModu.TAdapterActionName 6502
WebModu.TAdapterActionName.Action 6504
WebModu.TAdapterActionName.ActionName 6505
WebModu.TAdapterActionName.Adapter 6505
WebModu.TAdapterActionName.AssignTo 6503
WebModu.TAdapterActionName.Create 6503
WebModu.TCustomWebAppDataModule 6505
WebModu.TCustomWebAppDataModule.AppServices 6510
WebModu.TCustomWebAppDataModule.Create 6507
WebModu.TCustomWebAppDataModule.CreateNew 6507
WebModu.TCustomWebAppDataModule.Destroy 6508
WebModu.TCustomWebAppDataModule.GetWebAppServices 6508
WebModu.TCustomWebAppDataModule.Notification 6509
WebModu.TCustomWebAppDataModule.QueryInterface 6509
WebModu.TCustomWebAppPageModule 6510
WebModu.TCustomWebAppPageModule.AppServices 6515
WebModu.TCustomWebAppPageModule.Create 6512
WebModu.TCustomWebAppPageModule.CreateNew 6513
WebModu.TCustomWebAppPageModule.Destroy 6513
WebModu.TCustomWebAppPageModule.GetWebAppServices 6514
WebModu.TCustomWebAppPageModule.Notification 6514

WebModu.TCustomWebAppPageModule.QueryInterface 6514
WebModu.TCustomWebAppPageModule.SetAppServices 6515
WebModu.TCustomWebDataModule 6516
WebModu.TCustomWebDataModule.Create 6517
WebModu.TCustomWebDataModule.CreateNew 6518
WebModu.TCustomWebDataModule.Destroy 6518
WebModu.TCustomWebDataModule.GetSiteModuleHelper 6519
WebModu.TCustomWebDataModule.GetWebSnapObjClass 6519
WebModu.TCustomWebDataModule.Notification 6519
WebModu.TCustomWebDataModule.OnActivate 6522
WebModu.TCustomWebDataModule.OnDeactivate 6522
WebModu.TCustomWebDataModule.Request 6520
WebModu.TCustomWebDataModule.Response 6520
WebModu.TCustomWebDataModule.Session 6521
WebModu.TCustomWebDataModule.SiteModuleHelper 6521
WebModu.TCustomWebPageModule 6522
WebModu.TCustomWebPageModule.Create 6525
WebModu.TCustomWebPageModule.CreateNew 6525
WebModu.TCustomWebPageModule.DefaultAction 6527
WebModu.TCustomWebPageModule.Destroy 6526
WebModu.TCustomWebPageModule.GetSiteModuleHelper 6526
WebModu.TCustomWebPageModule.GetSitePageModuleHelper 6527
WebModu.TCustomWebPageModule.Notification 6527
WebModu.TCustomWebPageModule.OnAfterDispatchPage 6529
WebModu.TCustomWebPageModule.OnAfterRedirectToPage 6529
WebModu.TCustomWebPageModule.OnBeforeDispatchPage 6530
WebModu.TCustomWebPageModule.OnBeforeRedirectToPage 6530
WebModu.TCustomWebPageModule.PageProducer 6528
WebModu.TCustomWebPageModule.SitePageModuleHelper 6528
WebModu.TGetParamsEvent 6580
WebModu.TGetParamsEvent type 6580
WebModu.TSiteModuleHelper 6531
WebModu.TSiteModuleHelper.Create 6533
WebModu.TSiteModuleHelper.Destroy 6533
WebModu.TSiteModuleHelper.EndIterator 6534
WebModu.TSiteModuleHelper.FindNamedVariable 6534
WebModu.TSiteModuleHelper.FindVariable 6535
WebModu.TSiteModuleHelper.FindVariableContainer 6535
WebModu.TSiteModuleHelper.GetCurrentObject 6535
WebModu.TSiteModuleHelper.GetVariable 6535
WebModu.TSiteModuleHelper.GetVariableCount 6535
WebModu.TSiteModuleHelper.Module 6537
WebModu.TSiteModuleHelper.NextIteration 6536
WebModu.TSiteModuleHelper.Notification 6536
WebModu.TSiteModuleHelper.NotifyActivate 6536
WebModu.TSiteModuleHelper.NotifyDeactivate 6536
WebModu.TSiteModuleHelper.OnActivate 6539
WebModu.TSiteModuleHelper.OnDeactivate 6539
WebModu.TSiteModuleHelper.Request 6537
WebModu.TSiteModuleHelper.Response 6538
WebModu.TSiteModuleHelper.Session 6538
WebModu.TSiteModuleHelper.StartIterator 6537
WebModu.TSiteModuleHelper.WebModuleContext 6538
WebModu.TSitePageModuleHelper 6539
WebModu.TSitePageModuleHelper.Create 6542
WebModu.TSitePageModuleHelper.DefaultAction 6548
WebModu.TSitePageModuleHelper.Destroy 6542
WebModu.TSitePageModuleHelper.DispatchPage 6543
WebModu.TSitePageModuleHelper.GetDefaultAction 6543
WebModu.TSitePageModuleHelper.GetPageProducer 6544
WebModu.TSitePageModuleHelper.GetProducerComponent 6544
WebModu.TSitePageModuleHelper.GetRedirectParams 6544
WebModu.TSitePageModuleHelper.GetTemplateFileType 6544
WebModu.TSitePageModuleHelper.GetTemplateStream 6545
WebModu.TSitePageModuleHelper.GetXMLData 6545
WebModu.TSitePageModuleHelper.GetXSLData 6545
WebModu.TSitePageModuleHelper.HasHTMLBrowserView 6545
WebModu.TSitePageModuleHelper.HasScriptView 6545
WebModu.TSitePageModuleHelper.HasXMLBrowserView 6546
WebModu.TSitePageModuleHelper.HasXSLBrowserView 6546
WebModu.TSitePageModuleHelper.HTMLFromTemplate 6546

WebModu.TSitePageModuleHelper.ImplDispatchPage 6546
WebModu.TSitePageModuleHelper.ImplIncludePage 6547
WebModu.TSitePageModuleHelper.ImplRedirectToPage 6547
WebModu.TSitePageModuleHelper.ImplSetWebContentOptions 6547
WebModu.TSitePageModuleHelper.IncludePage 6547
WebModu.TSitePageModuleHelper.OnAfterDispatchPage 6549
WebModu.TSitePageModuleHelper.OnAfterRedirectToPage 6549
WebModu.TSitePageModuleHelper.OnBeforeDispatchPage 6550
WebModu.TSitePageModuleHelper.OnBeforeRedirectToPage 6550
WebModu.TSitePageModuleHelper.RedirectToPage 6547
WebModu.TSitePageModuleHelper.SetDefaultPageFileName 6548
WebModu.TSitePageModuleHelper.SetWebContentOptions 6548
WebModu.TWebAppDataModule 6550
WebModu.TWebAppDataModule.AppServices 6552
WebModu.TWebAppDataModule.OnActivate 6553
WebModu.TWebAppDataModule.OnDeactivate 6553
WebModu.TWebAppPageModule 6554
WebModu.TWebAppPageModule.AppServices 6556
WebModu.TWebAppPageModule.DefaultAction 6557
WebModu.TWebAppPageModule.OnActivate 6558
WebModu.TWebAppPageModule.OnAfterDispatchPage 6558
WebModu.TWebAppPageModule.OnBeforeDispatchPage 6559
WebModu.TWebAppPageModule.OnDeactivate 6559
WebModu.TWebAppPageModule.PageProducer 6557
WebModu.TWebAppPageModuleHelper 6560
WebModu.TWebAppPageModuleHelper.GetPageProducer 6562
WebModu.TWebDataModule 6562
WebModu.TWebDataModule.OnActivate 6564
WebModu.TWebDataModule.OnDeactivate 6564
WebModu.TWebNotifyContainer 6565
WebModu.TWebNotifyContainer.Add 6568
WebModu.TWebNotifyContainer.Create 6566
WebModu.TWebNotifyContainer.Destroy 6566
WebModu.TWebNotifyContainer.FList 6567
WebModu.TWebNotifyContainer.GetCount 6568
WebModu.TWebNotifyContainer.NotifyWebActivate 6568
WebModu.TWebNotifyContainer.NotifyWebDeactivate 6568
WebModu.TWebPageModule 6569
WebModu.TWebPageModule.DefaultAction 6571
WebModu.TWebPageModule.OnActivate 6572
WebModu.TWebPageModule.OnAfterDispatchPage 6573
WebModu.TWebPageModule.OnBeforeDispatchPage 6573
WebModu.TWebPageModule.OnDeactivate 6574
WebModu.TWebPageModule.PageProducer 6572
WebModu.TWebPageModuleHelper 6574
WebModu.TWebPageModuleHelper.GetPageProducer 6577
WebNode 6581
WebNode namespace 6581
Interfaces 6582
WebNode.IWebNode 6582
WebNode.IWebNode.BeforeExecute 6583
WebNode.IWebNode.Execute 6584
WebNode.IWebNode.GetMimeBoundary 6585
WebNode.IWebNode.MimeBoundary 6585
WebNode.IWebNode.SetMimeBoundary 6585
WebReq 6587
WebReq namespace 6587
Classes 6588
Functions 6608
Structs, Records, Enums 6609
Variables 6610
WebReq.TRequestNotification 6609
WebReq.TRequestNotification enumeration 6609
WebReq.TWebModuleFactoryList 6588
WebReq.TWebModuleFactoryList.AddFactory 6590
WebReq.TWebModuleFactoryList.AppModuleFactory 6591
WebReq.TWebModuleFactoryList.Create 6589
WebReq.TWebModuleFactoryList.Destroy 6590
WebReq.TWebModuleFactoryList.GetItem 6591
WebReq.TWebModuleFactoryList.ItemCount 6591
WebReq.TWebModuleFactoryList.Items 6592
WebReq.TWebModuleList 6592
WebReq.TWebModuleList.AddModule 6594
WebReq.TWebModuleList.AddModuleClass 6595

WebReq.TWebModuleList.AddModuleName 6595
WebReq.TWebModuleList.AutoCreateModules 6596
WebReq.TWebModuleList.AutoDestroyModules 6596
WebReq.TWebModuleList.Create 6593
WebReq.TWebModuleList.Destroy 6594
WebReq.TWebModuleList.Factories 6599
WebReq.TWebModuleList.FindModuleClass 6597
WebReq.TWebModuleList.FindModuleName 6597
WebReq.TWebModuleList.GetFactory 6598
WebReq.TWebModuleList.GetFactoryCount 6598
WebReq.TWebModuleList.GetItem 6598
WebReq.TWebModuleList.GetItemCount 6599
WebReq.TWebModuleList.GetOnModuleAdded 6599
WebReq.TWebModuleList.SetOnModuleAdded 6599
WebReq.TWebRequestHandler 6600
WebReq.TWebRequestHandler.ActivateWebModules 6602
WebReq.TWebRequestHandler.ActiveCount 6604
WebReq.TWebRequestHandler.AddWebModuleFactory 6602
WebReq.TWebRequestHandler.CacheConnections 6605
WebReq.TWebRequestHandler.Create 6601
WebReq.TWebRequestHandler.DeactivateWebModules 6603
WebReq.TWebRequestHandler.Destroy 6601
WebReq.TWebRequestHandler.HandleException 6603
WebReq.TWebRequestHandler.HandleRequest 6603
WebReq.TWebRequestHandler.InactiveCount 6605
WebReq.TWebRequestHandler.MaxConnections 6606
WebReq.TWebRequestHandler.WebModuleClass 6606
WebReq.TWebRequestHandler.WebModuleFactory 6607
WebReq.TWebRequestHandler.WebModuleFactoryCount 6607
WebReq.WebRequestHandler 6608
WebReq.WebRequestHandler function 6608
WebReq.WebRequestHandlerProc 6610
WebReq.WebRequestHandlerProc variable 6610
WebScript 6611
WebScript namespace 6611
Classes 6613
Constants 6689
Structs, Records, Enums 6686
Types 6688
WebScript.CreateComponentStubFactory 6688
WebScript.CreateComponentStubFactory type 6688
WebScript.ERecursiveIncludeFile 6613
WebScript.EScriptEvalError 6614
WebScript.GlobalScriptVariableNames 6689
WebScript.GlobalScriptVariableNames constant 6689
WebScript.PLineNumbers 6688
WebScript.PLineNumbers type 6688
WebScript.TAbstractScriptEngine 6614
WebScript.TAbstractScriptEngine.AddObjectFactory 6616
WebScript.TAbstractScriptEngine.Create 6615
WebScript.TAbstractScriptEngine.CreateComponentStub 6617
WebScript.TAbstractScriptEngine.Destroy 6615
WebScript.TAbstractScriptEngine.ListLanguageNames 6617
WebScript.TAbstractScriptEngine.Name 6617
WebScript.TAbstractScriptEngine.ScriptObjectFactories 6618
WebScript.TAbstractScriptEngine.ScriptProducerClass 6618
WebScript.TAbstractScriptEngine.SupportsLanguage 6617
WebScript.TAbstractScriptObjectFactories 6618
WebScript.TAbstractScriptObjectFactories.Count 6621
WebScript.TAbstractScriptObjectFactories.Factories 6621
WebScript.TAbstractScriptObjectFactories.GetCount 6620
WebScript.TAbstractScriptObjectFactories.GetFactory 6620
WebScript.TAbstractScriptObjectFactories.GetObjectList 6620
WebScript.TAbstractScriptObjectFactories.ObjectList 6621
WebScript.TAbstractScriptObjectFactory 6621
WebScript.TAbstractScriptObjectFactory.AddGlobalObjects 6622
WebScript.TAbstractScriptObjectFactory.CreateComponentObject 6622
WebScript.TAbstractScriptObjectFactory.CreateGlobalObject 6623
WebScript.TAbstractScriptObjectFactory.CreateProducerObject 6623
WebScript.TAbstractScriptObjectFactory.GetScriptEngine 6623
WebScript.TCurrentWebPageInfo 6623
WebScript.TCurrentWebPageInfo.Create 6625
WebScript.TCurrentWebPageInfo.GetHasViewAccess 6626
WebScript.TCurrentWebPageInfo.GetLoginRequired 6626
WebScript.TCurrentWebPageInfo.GetPageDescription 6626

WebScript.TCurrentWebPageInfo.GetPageHREF 6627
WebScript.TCurrentWebPageInfo.GetPageIsPublished 6627
WebScript.TCurrentWebPageInfo.GetPageName 6627
WebScript.TCurrentWebPageInfo.GetPageTitle 6628
WebScript.TCurrentWebPageInfo.WebPageInfo 6628
WebScript.TGlobalScriptVariables 6686
WebScript.TGlobalScriptVariables enumeration 6686
WebScript.THTMLItem 6628
WebScript.THTMLItem.AssignString 6630
WebScript.THTMLItem.Create 6630
WebScript.THTMLItem.HTML 6630
WebScript.THTMLItem.Ident 6631
WebScript.THTMLItem.ItemLength 6631
WebScript.THTMLItem.TagToken 6631
WebScript.THTMLItems 6631
WebScript.THTMLItems.Create 6632
WebScript.THTMLItems.Destroy 6632
WebScript.THTMLItems.GetHTMLItem 6633
WebScript.THTMLItems.Items 6633
WebScript.THTMLItems.MakeIdentifier 6633
WebScript.TIncludeType 6687
WebScript.TIncludeType enumeration 6687
WebScript.TLineNumberMap 6634
WebScript.TLineNumberMap.Count 6636
WebScript.TLineNumberMap.DestLineToSourceLine 6636
WebScript.TLineNumberMap.Items 6637
WebScript.TLineNumberMap.New 6636
WebScript.TLineNumbers 6687
WebScript.TLineNumbers record 6687
WebScript.TScriptCodeItem 6637
WebScript.TScriptCodeItem.AssignString 6638
WebScript.TScriptCodeItem.GetScript 6639
WebScript.TScriptEnginesList 6639
WebScript.TScriptEnginesList.AddEngine 6641
WebScript.TScriptEnginesList.AddObjectFactory 6641
WebScript.TScriptEnginesList.Count 6642
WebScript.TScriptEnginesList.Create 6640
WebScript.TScriptEnginesList.Destroy 6640
WebScript.TScriptEnginesList.Find 6641
WebScript.TScriptEnginesList.FindScriptProducerClass 6642
WebScript.TScriptEnginesList.Items 6643
WebScript.TScriptEnginesList.ListLanguageNames 6642
WebScript.TScriptEnginesList.RemoveEngine 6642
WebScript.TScriptError 6643
WebScript.TScriptError.Create 6644
WebScript.TScriptError.FCharPos 6644
WebScript.TScriptError.FDescription 6644
WebScript.TScriptError.FFileName 6645
WebScript.TScriptError.FLine 6645
WebScript.TScriptError.FSourceLine 6645
WebScript.TScriptError.GetCharPos 6645
WebScript.TScriptError.GetDescription 6646
WebScript.TScriptError.GetFileName 6646
WebScript.TScriptError.GetLine 6646
WebScript.TScriptError.GetSourceLine 6646
WebScript.TScriptError.SetDescription 6647
WebScript.TScriptErrors 6647
WebScript.TScriptErrors.Add 6649
WebScript.TScriptErrors.Create 6647
WebScript.TScriptErrors.Destroy 6648
WebScript.TScriptErrors.GetError 6649
WebScript.TScriptErrors.GetErrorCount 6649
WebScript.TScriptFile 6649
WebScript.TScriptFile.ApplicationName 6652
WebScript.TScriptFile.Create 6650
WebScript.TScriptFile.Destroy 6650
WebScript.TScriptFile.FLineNumberMap 6651
WebScript.TScriptFile.LongName 6652
WebScript.TScriptFile.ParseStream 6651
WebScript.TScriptFile.ParseString 6651
WebScript.TScriptFile.Script 6652
WebScript.TScriptFile.ShortName 6653
WebScript.TScriptFile.Source 6653
WebScript.TScriptGlobalObjects 6653
WebScript.TScriptGlobalObjects.AddNamedIntf 6656
WebScript.TScriptGlobalObjects.Create 6654
WebScript.TScriptGlobalObjects.Destroy 6655
WebScript.TScriptGlobalObjects.FindNamedItemIntf 6656

WebScript.TScriptGlobalObjects.GetNamedItemCount 6656
WebScript.TScriptGlobalObjects.GetNamedItemName 6656
WebScript.TScriptGlobalObjects.GetScriptEngine 6656
WebScript.TScriptGlobalObjects.NamedItemCount 6657
WebScript.TScriptGlobalObjects.NamedItemName 6657
WebScript.TScriptGlobalObjects.Producer 6657
WebScript.TScriptGlobalObjects.ScriptEngine 6658
WebScript.TScriptIncludeFile 6658
WebScript.TScriptIncludeFile.Create 6659
WebScript.TScriptIncludeFile.LoadFile 6659
WebScript.TScriptIncludeFiles 6659
WebScript.TScriptIncludeFiles.Create 6660
WebScript.TScriptIncludeFiles.GetScriptIncludeFile 6660
WebScript.TScriptIncludeFiles.Items 6661
WebScript.TScriptIncludeItem 6662
WebScript.TScriptIncludeItem.Create 6663
WebScript.TScriptIncludeItem.GetScript 6663
WebScript.TScriptItem 6664
WebScript.TScriptItem.Create 6665
WebScript.TScriptItem.GetScript 6665
WebScript.TScriptItem.Script 6666
WebScript.TScriptItems 6666
WebScript.TScriptItems.Create 6667
WebScript.TScriptItems.GetScriptItem 6667
WebScript.TScriptItems.Items 6667
WebScript.TScriptObjectFactories 6668
WebScript.TScriptObjectFactories.Count 6672
WebScript.TScriptObjectFactories.Create 6670
WebScript.TScriptObjectFactories.CreateGlobalObject 6671
WebScript.TScriptObjectFactories.Destroy 6670
WebScript.TScriptObjectFactories.Factories 6672
WebScript.TScriptObjectFactories.GetCount 6671
WebScript.TScriptObjectFactories.GetFactory 6671
WebScript.TScriptObjectFactories.GetObjectList 6672
WebScript.TScriptProducer 6673
WebScript.TScriptProducer.Component 6681
WebScript.TScriptProducer.Content 6681
WebScript.TScriptProducer.Create 6674
WebScript.TScriptProducer.Designing 6681
WebScript.TScriptProducer.Destroy 6674
WebScript.TScriptProducer.Errors 6681
WebScript.TScriptProducer.Evaluate 6676
WebScript.TScriptProducer.EvaluateExpression 6676
WebScript.TScriptProducer.FErrors 6675
WebScript.TScriptProducer.FScriptEngine 6675
WebScript.TScriptProducer.FScriptFile 6676
WebScript.TScriptProducer.GetContent 6677
WebScript.TScriptProducer.GetErrors 6677
WebScript.TScriptProducer.GetHTMLBlock 6677
WebScript.TScriptProducer.GetHTMLBlockCount 6677
WebScript.TScriptProducer.GetWebModuleContext 6678
WebScript.TScriptProducer.HandleParserError 6678
WebScript.TScriptProducer.HandleScriptError 6678
WebScript.TScriptProducer.ImplHandleScriptError 6678
WebScript.TScriptProducer.ParseStream 6678
WebScript.TScriptProducer.ParseString 6679
WebScript.TScriptProducer.ReplaceTags 6679
WebScript.TScriptProducer.Script 6682
WebScript.TScriptProducer.SetContent 6679
WebScript.TScriptProducer.Write 6680
WebScript.TScriptProducer.WriteItem 6680
WebScript.TVariableLookup 6682
WebScript.TVariableLookup.Create 6683
WebScript.TVariableLookup.FindGlobalVariable 6684
WebScript.TVariableLookup.FindGlobalVariableContainer 6684
WebScript.TVariableLookup.Ident 6684
WebScript.TVariableLookup.LookupObject 6684
WebScript.TVariableLookup.Obj 6685
WebServExp 6691
WebServExp namespace 6691
Classes 6693
Functions 6704
Interfaces 6701
Structs, Records, Enums 6706
Types 6708
WebServExp.ArgumentType 6706
WebServExp.ArgumentType enumeration 6706
WebServExp.GetAliasBaseTypeInfo 6704

WebServExp.GetAliasBaseTypeInfo function 6704
WebServExp.GetBindingType 6704
WebServExp.GetBindingType function 6704
WebServExp.IsBaseClassTypeInfo 6705
WebServExp.IsBaseClassTypeInfo function 6705
WebServExp.IWebServExp 6701
WebServExp.IWebServExp.FindOrAddSchema 6702
WebServExp.IWebServExp.GenerateXMLSchema 6702
WebServExp.IWebServExpAccess 6703
WebServExp.IWebServExpAccess.GetWebServExp 6703
WebServExp.MessageType 6706
WebServExp.MessageType enumeration 6706
WebServExp.TAfterPublishingWSDLEvent 6708
WebServExp.TAfterPublishingWSDLEvent type 6708
WebServExp.TBeforePublishingTypesEvent 6708
WebServExp.TBeforePublishingTypesEvent type 6708
WebServExp.TPublishingTypeEvent 6709
WebServExp.TPublishingTypeEvent type 6709
WebServExp.TSchemaType 6707
WebServExp.TSchemaType record 6707
WebServExp.TSchemaTypeArray 6709
WebServExp.TSchemaTypeArray type 6709
WebServExp.TWebServExp 6693
WebServExp.TWebServExp.ArrayAsComplexContent 6697
WebServExp.TWebServExp.BindingType 6698
WebServExp.TWebServExp.Create 6695
WebServExp.TWebServExp.Destroy 6695
WebServExp.TWebServExp.GenerateXMLSchema 6696
WebServExp.TWebServExp.GetWebServExp 6697
WebServExp.TWebServExp.GetWSDLForInterface 6697
WebServExp.TWebServExp.ImportLocations 6698
WebServExp.TWebServExp.ImportNames 6698
WebServExp.TWebServExp.OnAfterPublishingWSDL 6699
WebServExp.TWebServExp.OnBeforePublishingTypes 6699
WebServExp.TWebServExp.OnPublishingType 6699
WebServExp.TWebServExp.ServiceAddress 6698
WebServExp.TWebServExp.TargetNameSpace 6698
WebServExp.TWebServExp.WSDLElements 6699
WebSess 6711
WebSess namespace 6711
Classes 6712
Structs, Records, Enums 6739
Types 6740
WebSess.TCustomSessionsService 6712
WebSess.TCustomSessionsService.CheckStatusChange 6714
WebSess.TCustomSessionsService.Clear 6715
WebSess.TCustomSessionsService.Connect 6715
WebSess.TCustomSessionsService.ConnectActive 6716
WebSess.TCustomSessionsService.Create 6713
WebSess.TCustomSessionsService.DefaultTimeout 6720
WebSess.TCustomSessionsService.DoEndSession 6716
WebSess.TCustomSessionsService.DoStartSession 6716
WebSess.TCustomSessionsService.GetSessionIDs 6717
WebSess.TCustomSessionsService.LoadFromFile 6717
WebSess.TCustomSessionsService.LoadFromStream 6718
WebSess.TCustomSessionsService.MaxSessions 6720
WebSess.TCustomSessionsService.NewSession 6718
WebSess.TCustomSessionsService.NotifyStatusChange 6719
WebSess.TCustomSessionsService.OnEndSession 6721
WebSess.TCustomSessionsService.OnStartSession 6721
WebSess.TCustomSessionsService.SaveToFile 6719
WebSess.TCustomSessionsService.SaveToStream 6719
WebSess.TEndSessionEvent 6740
WebSess.TEndSessionEvent type 6740
WebSess.TEndSessionReason 6739
WebSess.TEndSessionReason enumeration 6739
WebSess.TSessionsService 6722
WebSess.TSessionsService.DefaultTimeout 6724
WebSess.TSessionsService.MaxSessions 6724
WebSess.TSessionsService.OnEndSession 6725
WebSess.TSessionsService.OnStartSession 6725
WebSess.TStartSessionEvent 6740
WebSess.TStartSessionEvent type 6740
WebSess.TWebSessionIDsImpl 6726
WebSess.TWebSessionIDsImpl.Create 6727
WebSess.TWebSessionIDsImpl.Destroy 6728
WebSess.TWebSessionIDsImpl.GetID 6728
WebSess.TWebSessionIDsImpl.GetIDCount 6729

WebSess.TWebSessionImpl 6729
WebSess.TWebSessionImpl.Create 6731
WebSess.TWebSessionImpl.Destroy 6731
WebSess.TWebSessionImpl.ExpirationTime 6736
WebSess.TWebSessionImpl.Expired 6736
WebSess.TWebSessionImpl.GetExpirationTime 6732
WebSess.TWebSessionImpl.GetExpired 6733
WebSess.TWebSessionImpl.GetID 6733
WebSess.TWebSessionImpl.GetIsActive 6733
WebSess.TWebSessionImpl.GetItemCount 6733
WebSess.TWebSessionImpl.GetItems 6734
WebSess.TWebSessionImpl.GetStartTime 6734
WebSess.TWebSessionImpl.GetTerminated 6734
WebSess.TWebSessionImpl.GetTimeoutMinutes 6734
WebSess.TWebSessionImpl.GetTouchTime 6734
WebSess.TWebSessionImpl.GetValue 6735
WebSess.TWebSessionImpl.ID 6736
WebSess.TWebSessionImpl.SetTimeoutMinutes 6735
WebSess.TWebSessionImpl.SetValue 6735
WebSess.TWebSessionImpl.StartTime 6737
WebSess.TWebSessionImpl.Terminate 6735
WebSess.TWebSessionImpl.Terminated 6737
WebSess.TWebSessionImpl.TimeoutMinutes 6737
WebSess.TWebSessionImpl.TouchTime 6737
WebSess.TWebSessionImpl.Values 6738
WebUsers 6741
WebUsers namespace 6741
Classes 6742
Structs, Records, Enums 6764
Types 6765
WebUsers.EUserIDNotFoundException 6742
WebUsers.EValidateUserException 6742
WebUsers.EWebUsersListException 6743
WebUsers.TCheckAccessRightsEvent 6765
WebUsers.TCheckAccessRightsEvent type 6765
WebUsers.TCheckAccessRightsHandledEvent 6765
WebUsers.TCheckAccessRightsHandledEvent type 6765
WebUsers.TCustomWebUserList 6744
WebUsers.TCustomWebUserList.CheckAccessRights 6746
WebUsers.TCustomWebUserList.Create 6745
WebUsers.TCustomWebUserList.Destroy 6745
WebUsers.TCustomWebUserList.LoadFromFile 6746
WebUsers.TCustomWebUserList.OnAfterCheckAccessRights 6748
WebUsers.TCustomWebUserList.OnAfterValidateUser 6749
WebUsers.TCustomWebUserList.OnBeforeCheckAccessRights 6749
WebUsers.TCustomWebUserList.OnBeforeValidateUser 6749
WebUsers.TCustomWebUserList.OnUserIDNotFound 6750
WebUsers.TCustomWebUserList.OnValidateUserError 6750
WebUsers.TCustomWebUserList.SaveToFile 6747
WebUsers.TCustomWebUserList.UserItems 6747
WebUsers.TCustomWebUserList.Users 6748
WebUsers.TCustomWebUserList.ValidateUser 6747
WebUsers.TValidateUserError 6764
WebUsers.TValidateUserError enumeration 6764
WebUsers.TValidateUserErrorEvent 6766
WebUsers.TValidateUserErrorEvent type 6766
WebUsers.TValidateUserEvent 6766
WebUsers.TValidateUserEvent type 6766
WebUsers.TValidateUserHandledEvent 6766
WebUsers.TValidateUserHandledEvent type 6766
WebUsers.TWebUserItem 6750
WebUsers.TWebUserItem.AccessRights 6753
WebUsers.TWebUserItem.AccessRightsStrings 6754
WebUsers.TWebUserItem.Assign 6752
WebUsers.TWebUserItem.CheckRights 6753
WebUsers.TWebUserItem.Create 6751
WebUsers.TWebUserItem.Destroy 6752
WebUsers.TWebUserItem.Password 6754
WebUsers.TWebUserItem.UserID 6754
WebUsers.TWebUserItem.UserName 6754
WebUsers.TWebUserItems 6755
WebUsers.TWebUserItems.Create 6756
WebUsers.TWebUserItems.FindUserID 6756
WebUsers.TWebUserItems.FindUserName 6756
WebUsers.TWebUserItems.GetAttr 6757
WebUsers.TWebUserItems.GetAttrCount 6757
WebUsers.TWebUserItems.GetItemAttr 6758

WebUsers.TWebUserItems.GetOwner 6758
WebUsers.TWebUserItems.Items 6759
WebUsers.TWebUserList 6759
WebUsers.TWebUserList.OnAfterCheckAccessRights 6761
WebUsers.TWebUserList.OnAfterValidateUser 6761
WebUsers.TWebUserList.OnBeforeCheckAccessRights 6762
WebUsers.TWebUserList.OnBeforeValidateUser 6762
WebUsers.TWebUserList.OnUserIDNotFound 6762
WebUsers.TWebUserList.OnValidateUserError 6763
WebUsers.TWebUserList.UserItems 6761
WeImport enumeration member 7137
weMessage enumeration member 7137
weOperation enumeration member 7137
wePart enumeration member 7137
wePortType enumeration member 7137
weService enumeration member 7137
weServiceImpl enumeration member 7137
weServiceIntf enumeration member 7137
weTypes enumeration member 7137
WideString class 3476
!= 3491
!= operator 3491
[] 3492
[] operator 3492
~WideString 3478
+ 3492
+= 3493
< 3493
< operator 3493
<= 3494
<= operator 3493
= 3494
== 3495
== operator 3494
> 3495
> operator 3495
>= 3496
>= operator 3496
about WideString class 3476
c_bstr 3481
cat_printf 3482
cat_sprintf 3482
cat_vprintf 3482
Copy 3482
CurrToStr 3483
CurrToStrF 3483
data 3483
Delete 3484
FloatToStrF 3484
Format 3484
FormatFloat 3484
Insert 3485
IntToHex 3485
IsDelimiter 3485
IsEmpty 3485
IsPathDelimiter 3486
LastChar 3486
LastDelimiter 3486
Length 3486
LowerCase 3487
Pos 3487
printf 3487
SetLength 3487
sprintf 3488
StringOfChar 3488
SubString 3488
swap 3489
ToDouble 3489
ToInt 3489
ToIntDef 3489
Trim 3490
TrimLeft 3490
TrimRight 3490
UpperCase 3490
vprintf 3491
WideString 3479
WideString constructor 3478
WideString enumerations 3480

WideString Members 3476
WideString methods 3481
WideString operators 3491
WideStrings 6767
WideStrings namespace 6767
Classes 6768
Interfaces 6819
Structs, Records, Enums 6821
Types 6822
WideStrings.IWideStringsAdapter 6819
WideStrings.IWideStringsAdapter.ReferenceStrings 6820
WideStrings.IWideStringsAdapter.ReleaseStrings 6820
WideStrings.PWideStringItem 6822
WideStrings.PWideStringItem type 6822
WideStrings.PWideStringItemList 6822
WideStrings.PWideStringItemList type 6822
WideStrings.TWideStringItem 6821
WideStrings.TWideStringItem record 6821
WideStrings.TWideStringItemList 6823
WideStrings.TWideStringItemList type 6823
WideStrings.TWideStringList 6768
WideStrings.TWideStringList.Add 6772
WideStrings.TWideStringList.AddObject 6773
WideStrings.TWideStringList.CaseSensitive 6782
WideStrings.TWideStringList.Changed 6773
WideStrings.TWideStringList.Changing 6774
WideStrings.TWideStringList.Clear 6774
WideStrings.TWideStringList.CompareStrings 6774
WideStrings.TWideStringList.Create 6771
WideStrings.TWideStringList.CustomSort 6775
WideStrings.TWideStringList.Delete 6775
WideStrings.TWideStringList.Destroy 6771
WideStrings.TWideStringList.Duplicates 6782
WideStrings.TWideStringList.Exchange 6776
WideStrings.TWideStringList.Find 6776
WideStrings.TWideStringList.Get 6777
WideStrings.TWideStringList.GetCapacity 6777
WideStrings.TWideStringList.GetCount 6777
WideStrings.TWideStringList.GetObject 6778
WideStrings.TWideStringList.IndexOf 6778
WideStrings.TWideStringList.Insert 6779
WideStrings.TWideStringList.InsertItem 6779
WideStrings.TWideStringList.InsertObject 6779
WideStrings.TWideStringList.OnChange 6784
WideStrings.TWideStringList.OnChanging 6785
WideStrings.TWideStringList.OwnsObjects 6783
WideStrings.TWideStringList.Put 6780
WideStrings.TWideStringList.PutObject 6780
WideStrings.TWideStringList.SetCapacity 6780
WideStrings.TWideStringList.SetUpdateState 6781
WideStrings.TWideStringList.Sort 6781
WideStrings.TWideStringList.Sorted 6783
WideStrings.TWideStringListSortCompare 6823
WideStrings.TWideStringListSortCompare type 6823
WideStrings.TWideStrings 6785
WideStrings.TWideStrings.Add 6789
WideStrings.TWideStrings.AddObject 6790
WideStrings.TWideStrings.AddStrings 6790
WideStrings.TWideStrings.Append 6791
WideStrings.TWideStrings.Assign 6791
WideStrings.TWideStrings.AssignTo 6792
WideStrings.TWideStrings.BeginUpdate 6792
WideStrings.TWideStrings.Capacity 6807
WideStrings.TWideStrings.Clear 6793
WideStrings.TWideStrings.CommaText 6807
WideStrings.TWideStrings.CompareStrings 6793
WideStrings.TWideStrings.Count 6809
WideStrings.TWideStrings.DefineProperties 6794
WideStrings.TWideStrings.Delete 6794
WideStrings.TWideStrings.DelimitedText 6809
WideStrings.TWideStrings.Delimiter 6810
WideStrings.TWideStrings.Destroy 6788
WideStrings.TWideStrings.EndUpdate 6795
WideStrings.TWideStrings.Equals 6795
WideStrings.TWideStrings.Error 6796
WideStrings.TWideStrings.Exchange 6796
WideStrings.TWideStrings.ExtractName 6797
WideStrings.TWideStrings.Get 6797

WideStrings.TWideStrings.GetCapacity 6797
WideStrings.TWideStrings.GetCount 6798
WideStrings.TWideStrings.GetEnumerator 6798
WideStrings.TWideStrings.GetObject 6798
WideStrings.TWideStrings.GetText 6799
WideStrings.TWideStrings.GetTextStr 6799
WideStrings.TWideStrings.IndexOf 6800
WideStrings.TWideStrings.IndexOfName 6800
WideStrings.TWideStrings.IndexOfObject 6800
WideStrings.TWideStrings.Insert 6801
WideStrings.TWideStrings.InsertObject 6801
WideStrings.TWideStrings.LineBreak 6810
WideStrings.TWideStrings.LoadFromFile 6802
WideStrings.TWideStrings.LoadFromStream 6802
WideStrings.TWideStrings.Move 6803
WideStrings.TWideStrings.Names 6810
WideStrings.TWideStrings.NameValueSeparator 6811
WideStrings.TWideStrings.Objects 6811
WideStrings.TWideStrings.Put 6803
WideStrings.TWideStrings.PutObject 6804
WideStrings.TWideStrings.QuoteChar 6812
WideStrings.TWideStrings.SaveToFile 6804
WideStrings.TWideStrings.SaveToStream 6804
WideStrings.TWideStrings.SetCapacity 6805
WideStrings.TWideStrings.SetText 6805
WideStrings.TWideStrings.SetTextStr 6806
WideStrings.TWideStrings.SetUpdateState 6806
WideStrings.TWideStrings.StrictDelimiter 6812
WideStrings.TWideStrings.Strings 6813
WideStrings.TWideStrings.StringsAdapter 6814
WideStrings.TWideStrings.Text 6814
WideStrings.TWideStrings.UpdateCount 6815
WideStrings.TWideStrings.ValueFromIndex 6815
WideStrings.TWideStrings.Values 6816
WideStrings.TWideStringsEnumerator 6816
WideStrings.TWideStringsEnumerator.Create 6817
WideStrings.TWideStringsEnumerator.Current 6818
WideStrings.TWideStringsEnumerator.GetCurrent 6817
WideStrings.TWideStringsEnumerator.MoveNext 6817
WideStrUtils 6825
WideStrUtils namespace 6825
Constants 6844
Functions 6827
Structs, Records, Enums 6842
Types 6843
WideStrUtils.AnsiToUtf8Ex 6828
WideStrUtils.AnsiToUtf8Ex function 6828
WideStrUtils.CharSet 6843
WideStrUtils.CharSet type 6843
WideStrUtils.ConvertStreamFromAnsiToUTF8 6828
WideStrUtils.ConvertStreamFromAnsiToUTF8 function 6828
WideStrUtils.ConvertStreamFromUTF8ToAnsi 6828
WideStrUtils.ConvertStreamFromUTF8ToAnsi function 6828
WideStrUtils.DetectUTF8Encoding 6829
WideStrUtils.DetectUTF8Encoding function 6829
WideStrUtils.HasExtendCharacter 6829
WideStrUtils.HasExtendCharacter function 6829
WideStrUtils.HasUTF8BOM 6829, 6830
WideStrUtils.HasUTF8BOM function 6829, 6830
WideStrUtils.inOpArray 6830
WideStrUtils.inOpArray function 6830
WideStrUtils.inOpSet 6830
WideStrUtils.inOpSet function 6830
WideStrUtils.IsUTF8LeadByte 6831
WideStrUtils.IsUTF8LeadByte function 6831
WideStrUtils.IsUTF8String 6831
WideStrUtils.IsUTF8String function 6831
WideStrUtils.IsUTF8TrailByte 6831
WideStrUtils.IsUTF8TrailByte function 6831
WideStrUtils.LoadResWideString 6832
WideStrUtils.LoadResWideString function 6832
WideStrUtils.LoadWideStr 6832
WideStrUtils.LoadWideStr function 6832
WideStrUtils.sUTF8BOMString 6844
WideStrUtils.sUTF8BOMString constant 6844
WideStrUtils.TEncodeType 6842
WideStrUtils.TEncodeType enumeration 6842
WideStrUtils.UTF8CharLength 6832

WideStrUtils.UTF8CharLength function 6832
WideStrUtils.UTF8CharSize 6833
WideStrUtils.UTF8CharSize function 6833
WideStrUtils.UTF8LowerCase 6833
WideStrUtils.UTF8LowerCase function 6833
WideStrUtils.Utf8ToAnsiEx 6833
WideStrUtils.Utf8ToAnsiEx function 6833
WideStrUtils.UTF8UpperCase 6834
WideStrUtils.UTF8UpperCase function 6834
WideStrUtils.WideAdjustLineBreaks 6834
WideStrUtils.WideAdjustLineBreaks function 6834
WideStrUtils.WideDequotedStr 6834
WideStrUtils.WideDequotedStr function 6834
WideStrUtils.WideExtractQuotedStr 6835
WideStrUtils.WideExtractQuotedStr function 6835
WideStrUtils.WideLastChar 6835
WideStrUtils.WideLastChar function 6835
WideStrUtils.WideQuotedStr 6835
WideStrUtils.WideQuotedStr function 6835
WideStrUtils.WideReplaceStr 6836
WideStrUtils.WideReplaceStr function 6836
WideStrUtils.WideReplaceText 6836
WideStrUtils.WideReplaceText function 6836
WideStrUtils.WideStringReplace 6836
WideStrUtils.WideStringReplace function 6836
WideStrUtils.WStrAlloc 6837
WideStrUtils.WStrAlloc function 6837
WideStrUtils.WStrBufSize 6837
WideStrUtils.WStrBufSize function 6837
WideStrUtils.WStrCat 6837
WideStrUtils.WStrCat function 6837
WideStrUtils.WStrComp 6838
WideStrUtils.WStrComp function 6838
WideStrUtils.WStrCopy 6838
WideStrUtils.WStrCopy function 6838
WideStrUtils.WStrDispose 6838
WideStrUtils.WStrDispose function 6838
WideStrUtils.WStrEnd 6839
WideStrUtils.WStrEnd function 6839
WideStrUtils.WStrLCopy 6839
WideStrUtils.WStrLCopy function 6839
WideStrUtils.WStrLen 6839
WideStrUtils.WStrLen function 6839
WideStrUtils.WStrMove 6840
WideStrUtils.WStrMove function 6840
WideStrUtils.WStrNew 6840
WideStrUtils.WStrNew function 6840
WideStrUtils.WStrPCopy 6840
WideStrUtils.WStrPCopy function 6840
WideStrUtils.WStrPLCopy 6841
WideStrUtils.WStrPLCopy function 6841
WideStrUtils.WStrPos 6841
WideStrUtils.WStrPos function 6841
WideStrUtils.WStrScan 6841
WideStrUtils.WStrScan function 6841
WinHelpViewer 6845
WinHelpViewer namespace 6845
    Interfaces 6846
    Variables 6849
WinHelpViewer.IWinHelpTester 6846
WinHelpViewer.IWinHelpTester.CanShowALink 6847
WinHelpViewer.IWinHelpTester.CanShowContext 6847
WinHelpViewer.IWinHelpTester.CanShowTopic 6847
WinHelpViewer.IWinHelpTester.GetDefaultHelpFile 6847
WinHelpViewer.IWinHelpTester.GetHelpPath 6848
WinHelpViewer.IWinHelpTester.GetHelpStrings 6848
WinHelpViewer.ViewerName 6849
WinHelpViewer.ViewerName variable 6849
WinHelpViewer.WinHelpTester 6849
WinHelpViewer.WinHelpTester variable 6849
wpLoginRequired enumeration member 6275
wpPublished enumeration member 6275
wrAbandoned enumeration member 3094
wrError enumeration member 3094
wrIOCompletion enumeration member 3094
wrSignaled enumeration member 3094
wrTimeout enumeration member 3094
WSDLBind 6851

WSDLBind namespace 6851
Classes 6854
Constants 7124
Functions 7122
Interfaces 7029
WSDLBind.GetDefinition 7122
WSDLBind.GetDefinition function 7122
WSDLBind.IBinding 7029
WSDLBind.IBinding.BindingOperations 7033
WSDLBind.IBinding.GetBindingOperations 7031
WSDLBind.IBinding.GetName 7032
WSDLBind.IBinding.GetType_ 7032
WSDLBind.IBinding.Name 7033
WSDLBind.IBinding.SetName 7032
WSDLBind.IBinding.SetType_ 7032
WSDLBind.IBinding.Type_ 7033
WSDLBind.IBindingInput 7033
WSDLBind.IBindingOperation 7034
WSDLBind.IBindingOperation.Faults 7037
WSDLBind.IBindingOperation.GetFaults 7036
WSDLBind.IBindingOperation.GetInput 7036
WSDLBind.IBindingOperation.GetName 7036
WSDLBind.IBindingOperation.GetOutput 7036
WSDLBind.IBindingOperation.Input 7037
WSDLBind.IBindingOperation.Name 7038
WSDLBind.IBindingOperation.Output 7038
WSDLBind.IBindingOperation.SetName 7037
WSDLBind.IBindingOperations 7038
WSDLBind.IBindingOperations.Add 7040
WSDLBind.IBindingOperations.BindingOperation 7041
WSDLBind.IBindingOperations.GetBindingOperation 7040
WSDLBind.IBindingOperations.Insert 7041
WSDLBind.IBindingOutput 7041
WSDLBind.IBindings 7042
WSDLBind.IBindings.Add 7044
WSDLBind.IBindings.Binding 7045
WSDLBind.IBindings.GetBinding 7044
WSDLBind.IBindings.Insert 7044
WSDLBind.IDefinition 7045
WSDLBind.IDefinition.Bindings 7050
WSDLBind.IDefinition.GetBindings 7047
WSDLBind.IDefinition.GetImports 7047
WSDLBind.IDefinition.GetMessages 7048
WSDLBind.IDefinition.GetName 7048
WSDLBind.IDefinition.GetPortTypes 7048
WSDLBind.IDefinition.GetServices 7048
WSDLBind.IDefinition.GetTargetNamespace 7048
WSDLBind.IDefinition.GetTypes 7049
WSDLBind.IDefinition.Imports 7050
WSDLBind.IDefinition.Messages 7050
WSDLBind.IDefinition.Name 7050
WSDLBind.IDefinition.PortTypes 7050
WSDLBind.IDefinition.Services 7051
WSDLBind.IDefinition.SetName 7049
WSDLBind.IDefinition.SetTargetNamespace 7049
WSDLBind.IDefinition.TargetNamespace 7051
WSDLBind.IDefinition.Types 7051
WSDLBind.IDocumentation 7051
WSDLBind.IDocumentationCollection 7052
WSDLBind.IDocumentationCollection.Add 7054
WSDLBind.IDocumentationCollection.Documentation 7055
WSDLBind.IDocumentationCollection.GetDocumentation 7054
WSDLBind.IDocumentationCollection.Insert 7054
WSDLBind.IDocumented 7055
WSDLBind.IDocumented.Documentation 7057
WSDLBind.IDocumented.GetDocumentationCollection 7057
WSDLBind.IFault 7057
WSDLBind.IFaults 7058
WSDLBind.IFaults.Add 7060
WSDLBind.IFaults.Fault 7061
WSDLBind.IFaults.GetFault 7060
WSDLBind.IFaults.Insert 7060
WSDLBind.IImport 7061
WSDLBind.IImport.GetLocation 7063
WSDLBind.IImport.GetNamespace 7063
WSDLBind.IImport.Location 7064
WSDLBind.IImport.Namespace 7064
WSDLBind.IImport.SetLocation 7063

WSDLBind.IImport.SetNamespace 7063
WSDLBind.IImports 7064
WSDLBind.IImports.Add 7066
WSDLBind.IImports.GetImport 7067
WSDLBind.IImports.Import 7067
WSDLBind.IImports.Insert 7067
WSDLBind.IMessage 7067
WSDLBind.IMessage.GetName 7069
WSDLBind.IMessage.GetParts 7070
WSDLBind.IMessage.Name 7070
WSDLBind.IMessage.Parts 7070
WSDLBind.IMessage.SetName 7070
WSDLBind.IMessages 7071
WSDLBind.IMessages.Add 7073
WSDLBind.IMessages.GetMessage 7073
WSDLBind.IMessages.Insert 7073
WSDLBind.IMessages.Message 7074
WSDLBind.IOperation 7074
WSDLBind.IOperation.Faults 7077
WSDLBind.IOperation.GetFaults 7076
WSDLBind.IOperation.GetInput 7076
WSDLBind.IOperation.GetName 7076
WSDLBind.IOperation.GetOutput 7077
WSDLBind.IOperation.Input 7077
WSDLBind.IOperation.Name 7078
WSDLBind.IOperation.Output 7078
WSDLBind.IOperation.SetName 7077
WSDLBind.IOperations 7078
WSDLBind.IOperations.Add 7080, 7081
WSDLBind.IOperations.GetOperation 7081
WSDLBind.IOperations.Insert 7081
WSDLBind.IOperations.Operation 7081
WSDLBind.IParam 7082
WSDLBind.IParam.GetMessage 7084
WSDLBind.IParam.GetName 7084
WSDLBind.IParam.Message 7085
WSDLBind.IParam.Name 7085
WSDLBind.IParam.SetMessage 7084
WSDLBind.IParam.SetName 7084
WSDLBind.IPart 7085
WSDLBind.IPart.Element 7089
WSDLBind.IPart.GetElement 7087
WSDLBind.IPart.GetName 7088
WSDLBind.IPart.GetType_ 7088
WSDLBind.IPart.Name 7089
WSDLBind.IPart.SetElement 7088
WSDLBind.IPart.SetName 7088
WSDLBind.IPart.SetType_ 7088
WSDLBind.IPart.Type_ 7089
WSDLBind.IParts 7090
WSDLBind.IParts.Add 7092
WSDLBind.IParts.GetPart 7092
WSDLBind.IParts.Insert 7092
WSDLBind.IParts.Part 7093
WSDLBind.IPort 7093
WSDLBind.IPort.Binding 7096
WSDLBind.IPort.GetBinding 7095
WSDLBind.IPort.GetName 7095
WSDLBind.IPort.Name 7096
WSDLBind.IPort.SetBinding 7095
WSDLBind.IPort.SetName 7095
WSDLBind.IPorts 7096
WSDLBind.IPorts.Add 7098
WSDLBind.IPorts.GetPort 7099
WSDLBind.IPorts.Insert 7099
WSDLBind.IPorts.Port 7099
WSDLBind.IPortType 7099
WSDLBind.IPortType.GetName 7101
WSDLBind.IPortType.GetOperations 7102
WSDLBind.IPortType.Name 7102
WSDLBind.IPortType.Operations 7102
WSDLBind.IPortType.SetName 7102
WSDLBind.IPortTypes 7103
WSDLBind.IPortTypes.Add 7105
WSDLBind.IPortTypes.GetPortType 7105
WSDLBind.IPortTypes.Insert 7105
WSDLBind.IPortTypes.PortType 7106
WSDLBind.IService 7106

WSDLBind.IService.GetName 7108
WSDLBind.IService.GetPorts 7108
WSDLBind.IService.Name 7109
WSDLBind.IService.Ports 7109
WSDLBind.IService.SetName 7108
WSDLBind.IServices 7109
WSDLBind.IServices.Add 7111
WSDLBind.IServices.GetService 7111
WSDLBind.IServices.Insert 7112
WSDLBind.IServices.Service 7112
WSDLBind.ITypes 7112
WSDLBind.ITypes.GetSchemaDefs 7114
WSDLBind.ITypes.SchemaDefs 7114
WSDLBind.IWSDLDocument 7115
WSDLBind.IWSDLDocument.Definition 7118
WSDLBind.IWSDLDocument.GetDefinition 7117
WSDLBind.IWSDLDocument.SetDefinition 7117
WSDLBind.IXMLSchemaDefs 7119
WSDLBind.IXMLSchemaDefs.Add 7121
WSDLBind.IXMLSchemaDefs.GetXMLSchemaDef 7121
WSDLBind.IXMLSchemaDefs.SchemaDef 7121
WSDLBind.LoadWSDLDoc 7122
WSDLBind.LoadWSDLDoc function 7122
WSDLBind.NewWSDLDoc 7122
WSDLBind.NewWSDLDoc function 7122
WSDLBind.SBinding 7124
WSDLBind.SBinding constant 7124
WSDLBind.SDefinitions 7124
WSDLBind.SDefinitions constant 7124
WSDLBind.SFault 7125
WSDLBind.SFault constant 7125
WSDLBind.SImport 7125
WSDLBind.SImport constant 7125
WSDLBind.SInput 7125
WSDLBind.SInput constant 7125
WSDLBind.SLocation 7126
WSDLBind.SLocation constant 7126
WSDLBind.SMessage 7126
WSDLBind.SMessage constant 7126
WSDLBind.SNamespace 7126
WSDLBind.SNamespace constant 7126
WSDLBind.SOperation 7127
WSDLBind.SOperation constant 7127
WSDLBind.SOutput 7127
WSDLBind.SOutput constant 7127
WSDLBind.SPart 7127
WSDLBind.SPart constant 7127
WSDLBind.SPort 7128
WSDLBind.SPort constant 7128
WSDLBind.SPortType 7128
WSDLBind.SPortType constant 7128
WSDLBind.SService 7128
WSDLBind.SService constant 7128
WSDLBind.SType 7129
WSDLBind.SType constant 7129
WSDLBind.STypes 7129
WSDLBind.STypes constant 7129
WSDLBind.TBinding 6854
WSDLBind.TBinding.AfterConstruction 6860
WSDLBind.TBinding.GetBindingOperations 6860
WSDLBind.TBinding.GetName 6860
WSDLBind.TBinding.GetType_ 6861
WSDLBind.TBinding.SetName 6861
WSDLBind.TBinding.SetType_ 6861
WSDLBind.TBindingOperation 6861
WSDLBind.TBindingOperation.AfterConstruction 6867
WSDLBind.TBindingOperation.GetFaults 6867
WSDLBind.TBindingOperation.GetInput 6868
WSDLBind.TBindingOperation.GetName 6868
WSDLBind.TBindingOperation.GetOutput 6868
WSDLBind.TBindingOperation.SetName 6868
WSDLBind.TBindingOperations 6869
WSDLBind.TBindingOperations.Add 6874
WSDLBind.TBindingOperations.GetBindingOperation 6875
WSDLBind.TBindingOperations.Insert 6875
WSDLBind.TBindings 6875
WSDLBind.TBindings.Add 6881
WSDLBind.TBindings.GetBinding 6881

WSDLBind.TBindings.Insert 6881
WSDLBind.TDefinition 6882
WSDLBind.TDefinition.AfterConstruction 6887
WSDLBind.TDefinition.GetBindings 6888
WSDLBind.TDefinition.GetImports 6888
WSDLBind.TDefinition.GetMessages 6888
WSDLBind.TDefinition.GetName 6889
WSDLBind.TDefinition.GetPortTypes 6889
WSDLBind.TDefinition.GetServices 6889
WSDLBind.TDefinition.GetTargetNamespace 6889
WSDLBind.TDefinition.GetTypes 6889
WSDLBind.TDefinition.SetName 6890
WSDLBind.TDefinition.SetTargetNamespace 6890
WSDLBind.TDocumentation 6890
WSDLBind.TDocumentationCollection 6891
WSDLBind.TDocumentationCollection.Add 6896
WSDLBind.TDocumentationCollection.GetDocumentation 6897
WSDLBind.TDocumentationCollection.Insert 6897
WSDLBind.TDocumented 6897
WSDLBind.TDocumented.AfterConstruction 6902
WSDLBind.TDocumented.GetDocumentationCollection 6903
WSDLBind.TFault 6903
WSDLBind.TFaults 6904
WSDLBind.TFaults.Add 6909
WSDLBind.TFaults.GetFault 6910
WSDLBind.TFaults.Insert 6910
WSDLBind.TImport 6910
WSDLBind.TImport.GetLocation 6915
WSDLBind.TImport.GetNamespace 6916
WSDLBind.TImport.SetLocation 6916
WSDLBind.TImport.SetNamespace 6916
WSDLBind.TImports 6916
WSDLBind.TImports.Add 6922
WSDLBind.TImports.GetImport 6922
WSDLBind.TImports.Insert 6923
WSDLBind.TMessage 6923
WSDLBind.TMessage.AfterConstruction 6928
WSDLBind.TMessage.GetName 6929
WSDLBind.TMessage.GetParts 6929
WSDLBind.TMessage.SetName 6929
WSDLBind.TMessages 6930
WSDLBind.TMessages.Add 6935
WSDLBind.TMessages.GetMessage 6936
WSDLBind.TMessages.Insert 6936
WSDLBind.TOperation 6936
WSDLBind.TOperation.AfterConstruction 6942
WSDLBind.TOperation.GetFaults 6942
WSDLBind.TOperation.GetInput 6942
WSDLBind.TOperation.GetName 6942
WSDLBind.TOperation.GetOutput 6943
WSDLBind.TOperation.SetName 6943
WSDLBind.TOperations 6943
WSDLBind.TOperations.Add 6949
WSDLBind.TOperations.GetOperation 6950
WSDLBind.TOperations.Insert 6950
WSDLBind.TParam 6950
WSDLBind.TParam.GetMessage 6956
WSDLBind.TParam.GetName 6956
WSDLBind.TParam.SetMessage 6956
WSDLBind.TParam.SetName 6956
WSDLBind.TPart 6957
WSDLBind.TPart.GetElement 6962
WSDLBind.TPart.GetName 6962
WSDLBind.TPart.GetType_ 6963
WSDLBind.TPart.SetElement 6963
WSDLBind.TPart.SetName 6963
WSDLBind.TPart.SetType_ 6963
WSDLBind.TParts 6964
WSDLBind.TParts.Add 6969
WSDLBind.TParts.GetPart 6970
WSDLBind.TParts.Insert 6970
WSDLBind.TPort 6970
WSDLBind.TPort.GetBinding 6976
WSDLBind.TPort.GetName 6976
WSDLBind.TPort.SetBinding 6976
WSDLBind.TPort.SetName 6976
WSDLBind.TPorts 6977
WSDLBind.TPorts.Add 6982

WSDLBind.TPorts.GetPort 6983
WSDLBind.TPorts.Insert 6983
WSDLBind.TPortType 6983
WSDLBind.TPortType.AfterConstruction 6988
WSDLBind.TPortType.GetName 6989
WSDLBind.TPortType.GetOperations 6989
WSDLBind.TPortType.SetName 6989
WSDLBind.TPortTypes 6990
WSDLBind.TPortTypes.Add 6995
WSDLBind.TPortTypes.GetPortType 6996
WSDLBind.TPortTypes.Insert 6996
WSDLBind.TService 6996
WSDLBind.TService.AfterConstruction 7001
WSDLBind.TService.GetName 7002
WSDLBind.TService.GetPorts 7002
WSDLBind.TService.SetName 7002
WSDLBind.TServices 7003
WSDLBind.TServices.Add 7008
WSDLBind.TServices.GetService 7009
WSDLBind.TServices.Insert 7009
WSDLBind.TTypes 7009
WSDLBind.TTypes.AfterConstruction 7014
WSDLBind.TTypes.GetSchemaDefs 7015
WSDLBind.TWSDLDocument 7015
WSDLBind.TWSDLDocument.AfterConstruction 7020
WSDLBind.TWSDLDocument.Definition 7022
WSDLBind.TWSDLDocument.GetDefinition 7021
WSDLBind.TWSDLDocument.LoadData 7021
WSDLBind.TWSDLDocument.SetDefinition 7021
WSDLBind.TXMLSchemaDefs 7022
WSDLBind.TXMLSchemaDefs.Add 7028
WSDLBind.TXMLSchemaDefs.GetXMLSchemaDef 7028
WSDLIntf 7131
WSDLIntf namespace 7131
Classes 7133
Constants 7140
Structs, Records, Enums 7137
Types 7139
WSDLIntf.Directives 7141
WSDLIntf.Directives constant 7141
WSDLIntf.DynArrayTypes 7141
WSDLIntf.DynArrayTypes constant 7141
WSDLIntf.Httpns 7142
WSDLIntf.Httpns constant 7142
WSDLIntf.Operators 7142
WSDLIntf.Operators constant 7142
WSDLIntf.OtherReservedWords 7142
WSDLIntf.OtherReservedWords constant 7142
WSDLIntf.ReservedWords 7143
WSDLIntf.ReservedWords constant 7143
WSDLIntf.SAddress 7143
WSDLIntf.SAddress constant 7143
WSDLIntf.SAnySimpleType 7144
WSDLIntf.SAnySimpleType constant 7144
WSDLIntf.SAnyType 7144
WSDLIntf.SAnyType constant 7144
WSDLIntf.SArray 7144
WSDLIntf.SArray constant 7144
WSDLIntf.SArrayOf 7145
WSDLIntf.SArrayOf constant 7145
WSDLIntf.SArrayType 7145
WSDLIntf.SArrayType constant 7145
WSDLIntf.SBinding 7145
WSDLIntf.SBinding constant 7145
WSDLIntf.SBody 7146
WSDLIntf.SBody constant 7146
WSDLIntf.ScalarArrayTypes 7146
WSDLIntf.ScalarArrayTypes constant 7146
WSDLIntf.SComplexType 7146
WSDLIntf.SComplexType constant 7146
WSDLIntf.SDefinitions 7147
WSDLIntf.SDefinitions constant 7147
WSDLIntf.SDynArray 7147
WSDLIntf.SDynArray constant 7147
WSDLIntf.SElement 7147
WSDLIntf.SElement constant 7147
WSDLIntf.SEncodingStyle 7148
WSDLIntf.SEncodingStyle constant 7148

WSDLIntf.SFault 7148
WSDLIntf.SFault constant 7148
WSDLIntf.SHeader 7148
WSDLIntf.SHeader constant 7148
WSDLIntf.SImport 7149
WSDLIntf.SImport constant 7149
WSDLIntf.SInOut 7149
WSDLIntf.SInOut constant 7149
WSDLIntf.SInput 7149
WSDLIntf.SInput constant 7149
WSDLIntf.SLocation 7150
WSDLIntf.SLocation constant 7150
WSDLIntf.SMessage 7150
WSDLIntf.SMessage constant 7150
WSDLIntf.SName 7150
WSDLIntf.SName constant 7150
WSDLIntf.SNameSpace 7151
WSDLIntf.SNameSpace constant 7151
WSDLIntf.SNsPrefix 7151
WSDLIntf.SNsPrefix constant 7151
WSDLIntf.Soapns 7151
WSDLIntf.Soapns constant 7151
WSDLIntf.SOperation 7152
WSDLIntf.SOperation constant 7152
WSDLIntf.SOutput 7152
WSDLIntf.SOutput constant 7152
WSDLIntf.SPart 7152
WSDLIntf.SPart constant 7152
WSDLIntf.SParts 7153
WSDLIntf.SParts constant 7153
WSDLIntf.SPort 7153
WSDLIntf.SPort constant 7153
WSDLIntf.SPortType 7153
WSDLIntf.SPortType constant 7153
WSDLIntf.SRequest 7154
WSDLIntf.SRequest constant 7154
WSDLIntf.SResponse 7154
WSDLIntf.SResponse constant 7154
WSDLIntf.SReturn 7154
WSDLIntf.SReturn constant 7154
WSDLIntf.SSchema 7155
WSDLIntf.SSchema constant 7155
WSDLIntf.SService 7155
WSDLIntf.SService constant 7155
WSDLIntf.SSoapAction 7155
WSDLIntf.SSoapAction constant 7155
WSDLIntf.SStyle 7156
WSDLIntf.SStyle constant 7156
WSDLIntf.STns 7156
WSDLIntf.STns constant 7156
WSDLIntf.STransport 7156
WSDLIntf.STransport constant 7156
WSDLIntf.SType 7157
WSDLIntf.SType constant 7157
WSDLIntf.STypes 7157
WSDLIntf.STypes constant 7157
WSDLIntf.SUnknown 7157
WSDLIntf.SUnknown constant 7157
WSDLIntf.SUse 7158
WSDLIntf.SUse constant 7158
WSDLIntf.SVerb 7158
WSDLIntf.SVerb constant 7158
WSDLIntf.TDOMStrings 7133
WSDLIntf.TDOMStrings.Add 7134
WSDLIntf.TDOMStrings.Clear 7135
WSDLIntf.TDOMStrings.Count 7135
WSDLIntf.TDOMStrings.Create 7134
WSDLIntf.TDOMStrings.Destroy 7134
WSDLIntf.TDOMStrings.IndexOf 7135
WSDLIntf.TDOMStrings.IndexOfIgnoreCase 7135
WSDLIntf.TDOMStrings.Insert 7135
WSDLIntf.TDOMStrings.Strings 7136
WSDLIntf.tns 7158
WSDLIntf.tns constant 7158
WSDLIntf.TWebServiceBindingType 7137
WSDLIntf.TWebServiceBindingType enumeration 7137
WSDLIntf.TWSDLElement 7137
WSDLIntf.TWSDLElement enumeration 7137

WSDLIntf.TWSDLElements 7139
WSDLIntf.TWSDLElements type 7139
WSDLIntf.TWString 7138
WSDLIntf.TWString record 7138
WSDLIntf.Wsdlns 7159
WSDLIntf.Wsdlns constant 7159
WSDLIntf.xsdns 7159
WSDLIntf.xsdns constant 7159
WSDLItems 7161
WSDLItems namespace 7161
Classes 7163
Functions 7211
Interfaces 7199
Structs, Records, Enums 7213
Types 7214
WSDLItems.HasDefinition 7211
WSDLItems.HasDefinition function 7211
WSDLItems.IHeaderInfo 7199
WSDLItems.IHeaderInfo.EncodingStyle 7203
WSDLItems.IHeaderInfo.GetEncodingStyle 7200
WSDLItems.IHeaderInfo.GetName 7200
WSDLItems.IHeaderInfo.GetNamespace 7201
WSDLItems.IHeaderInfo.GetPart 7201
WSDLItems.IHeaderInfo.GetRequired 7201
WSDLItems.IHeaderInfo.GetUse 7201
WSDLItems.IHeaderInfo.Name 7203
WSDLItems.IHeaderInfo.Namespace 7203
WSDLItems.IHeaderInfo.Part 7204
WSDLItems.IHeaderInfo.Required 7204
WSDLItems.IHeaderInfo.SetEncodingStyle 7201
WSDLItems.IHeaderInfo.SetName 7202
WSDLItems.IHeaderInfo.SetNamespace 7202
WSDLItems.IHeaderInfo.SetPart 7202
WSDLItems.IHeaderInfo.SetRequired 7202
WSDLItems.IHeaderInfo.SetUse 7203
WSDLItems.IHeaderInfo.Use 7204
WSDLItems.IHeaderInfoArray 7214
WSDLItems.IHeaderInfoArray type 7214
WSDLItems.IQualifiedName 7204
WSDLItems.IQualifiedName.GetName 7205
WSDLItems.IQualifiedName.GetNamespace 7206
WSDLItems.IQualifiedName.Name 7207
WSDLItems.IQualifiedName.Namespace 7207
WSDLItems.IQualifiedName.SetName 7206
WSDLItems.IQualifiedName.SetNamespace 7206
WSDLItems.IQualifiedNameArray 7214
WSDLItems.IQualifiedNameArray type 7214
WSDLItems.IterateImportOptions 7213
WSDLItems.IterateImportOptions enumeration 7213
WSDLItems.IWSDLItems 7208
WSDLItems.IWSDLItems.GetWSDLDoc 7209
WSDLItems.IWSDLItems.GetWSDLItems 7209
WSDLItems.IWSDLItems.Load 7209
WSDLItems.NewQualifiedName 7211
WSDLItems.NewQualifiedName function 7211
WSDLItems.TWSDLItems 7163
WSDLItems.TWSDLItems.CompareName 7171
WSDLItems.TWSDLItems.Create 7169
WSDLItems.TWSDLItems.GetBindingForServicePort 7172
WSDLItems.TWSDLItems.GetBindingOfType 7173
WSDLItems.TWSDLItems.GetBindingOperationNode 7173
WSDLItems.TWSDLItems.GetBindings 7174
WSDLItems.TWSDLItems.GetBindingType 7174
WSDLItems.TWSDLItems.GetHttpBindingAttribute 7175
WSDLItems.TWSDLItems.GetImports 7175
WSDLItems.TWSDLItems.GetLocationForImport 7176
WSDLItems.TWSDLItems.GetMessageNode 7176
WSDLItems.TWSDLItems.GetMessages 7177
WSDLItems.TWSDLItems.GetName 7177
WSDLItems.TWSDLItems.GetOperationNode 7177
WSDLItems.TWSDLItems.GetOperations 7178
WSDLItems.TWSDLItems.GetOperationsForBinding 7178
WSDLItems.TWSDLItems.GetPartNode 7179
WSDLItems.TWSDLItems.GetParts 7179
WSDLItems.TWSDLItems.GetPartsForOperation 7180
WSDLItems.TWSDLItems.GetPortsForService 7180
WSDLItems.TWSDLItems.GetPortTypeNode 7181
WSDLItems.TWSDLItems.GetPortTypes 7181

WSDLItems.TWSDLItems.GetSchemaNode 7182
WSDLItems.TWSDLItems.GetSchemas 7182
WSDLItems.TWSDLItems.GetServiceAndPortOfBinding 7183
WSDLItems.TWSDLItems.GetServiceNode 7183
WSDLItems.TWSDLItems.GetServices 7184
WSDLItems.TWSDLItems.GetSoapAction 7184
WSDLItems.TWSDLItems.GetSoapActionList 7185
WSDLItems.TWSDLItems.GetSoapAddressForServicePort 7185
WSDLItems.TWSDLItems.GetSoapBindingAttribute 7186
WSDLItems.TWSDLItems.GetSoapBindingInputNode 7186
WSDLItems.TWSDLItems.GetSoapBindingIONode 7187
WSDLItems.TWSDLItems.GetSoapBindingOutputNode 7188
WSDLItems.TWSDLItems.GetSoapBodyAttribute 7188
WSDLItems.TWSDLItems.GetSoapBodyInputAttribute 7189
WSDLItems.TWSDLItems.GetSoapBodyNamespace 7190
WSDLItems.TWSDLItems.GetSoapBodyOutputAttribute 7190
WSDLItems.TWSDLItems.GetSoapHeaders 7191
WSDLItems.TWSDLItems.GetSoapInputHeaders 7192
WSDLItems.TWSDLItems.GetSoapOperationAttribute 7192
WSDLItems.TWSDLItems.GetSoapOperationStyle 7193
WSDLItems.TWSDLItems.GetSoapOutputHeaders 7193
WSDLItems.TWSDLItems.GetTargetNamespace 7194
WSDLItems.TWSDLItems.GetTargetNamespacePrefix 7195
WSDLItems.TWSDLItems.GetWSDLDoc 7195
WSDLItems.TWSDLItems.GetWSDLItems 7195
WSDLItems.TWSDLItems.HasTypesNode 7196
WSDLItems.TWSDLItems.IsPortTypeHTTPBound 7196
WSDLItems.TWSDLItems.IterateImports 7196
WSDLItems.TWSDLItems.Load 7197
WSDLItems.TWSDLItems.StreamLoader 7197
WSDLItems.TWSDLItems.TargetNamespace 7198
WSDLItems.TWSDLIterateProc 7214
WSDLItems.TWSDLIterateProc type 7214
WSDLNode 7217
WSDLNode namespace 7217
Classes 7218
Functions 7229
WSDLNode.ActivateWSDL 7229
WSDLNode.ActivateWSDL function 7229
WSDLNode.EWSDLLoadException 7218
WSDLNode.TWSDLClientNode 7218
WSDLNode.TWSDLClientNode.BeforeExecute 7219
WSDLNode.TWSDLClientNode.Execute 7220, 7221
WSDLNode.TWSDLClientNode.GetMimeBoundary 7221
WSDLNode.TWSDLClientNode.Notification 7221
WSDLNode.TWSDLClientNode.SetMimeBoundary 7222
WSDLNode.TWSDLClientNode.TransportNode 7222
WSDLNode.TWSDLClientNode.WSDLView 7223
WSDLNode.TWSDLView 7223
WSDLNode.TWSDLView.Activate 7225
WSDLNode.TWSDLView.IntfInfo 7224
WSDLNode.TWSDLView.Operation 7225
WSDLNode.TWSDLView.Password 7226
WSDLNode.TWSDLView.Port 7226
WSDLNode.TWSDLView.PortType 7226
WSDLNode.TWSDLView.Proxy 7227
WSDLNode.TWSDLView.Service 7227
WSDLNode.TWSDLView.SetDesignState 7225
WSDLNode.TWSDLView.UserName 7227
WSDLNode.TWSDLView.WSDL 7228
WSDLPub 7231
WSDLPub namespace 7231
Classes 7232
Functions 7249
Interfaces 7247
Structs, Records, Enums 7250
Types 7251
WSDLPub.IWSDLPublish 7247
WSDLPub.IWSDLPublish.GetPortTypeList 7248
WSDLPub.IWSDLPublish.GetTypeSystemsList 7248
WSDLPub.IWSDLPublish.GetWSDLForPortType 7248
WSDLPub.IWSDLPublish.GetXSDForTypeSystem 7248
WSDLPub.SOAPPublishOptions 7250
WSDLPub.SOAPPublishOptions enumeration 7250
WSDLPub.TBeforePublishingWSDLEvent 7251
WSDLPub.TBeforePublishingWSDLEvent type 7251
WSDLPub.TSOAPPublishOptions 7251
WSDLPub.TSOAPPublishOptions type 7251

WSDLPub.TWSDLHTMLPublish 7232
WSDLPub.TWSDLHTMLPublish.AddInterfaceList 7235
WSDLPub.TWSDLHTMLPublish.AddPortList 7235
WSDLPub.TWSDLHTMLPublish.AdminEnabled 7239
WSDLPub.TWSDLHTMLPublish.Create 7233
WSDLPub.TWSDLHTMLPublish.Destroy 7234
WSDLPub.TWSDLHTMLPublish.DispatchEnabled 7235
WSDLPub.TWSDLHTMLPublish.DispatchMask 7235
WSDLPub.TWSDLHTMLPublish.DispatchMethodType 7236
WSDLPub.TWSDLHTMLPublish.DispatchRequest 7236
WSDLPub.TWSDLHTMLPublish.GetHostScriptBaseURL 7237
WSDLPub.TWSDLHTMLPublish.GetTargetNamespace 7237
WSDLPub.TWSDLHTMLPublish.HandleRequest 7237
WSDLPub.TWSDLHTMLPublish.OnAfterPublishingWSDL 7241
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingTypes 7241
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingWSDL 7242
WSDLPub.TWSDLHTMLPublish.OnPublishingType 7242
WSDLPub.TWSDLHTMLPublish.PublishOptions 7240
WSDLPub.TWSDLHTMLPublish.ServiceInfo 7238
WSDLPub.TWSDLHTMLPublish.TargetNamespace 7240
WSDLPub.TWSDLHTMLPublish.UpdatePortList 7238
WSDLPub.TWSDLHTMLPublish.WebDispatch 7240
WSDLPub.TWSDLHTMLPublish.WSILInfo 7238
WSDLPub.TWSDLPublish 7242
WSDLPub.TWSDLPublish.GetPortTypeEntries 7243
WSDLPub.TWSDLPublish.GetPortTypeList 7244
WSDLPub.TWSDLPublish.GetTypeSystemsList 7244
WSDLPub.TWSDLPublish.GetWSDLForPortType 7244
WSDLPub.TWSDLPublish.GetXSDForTypeSystem 7244
WSDLPub.TWSDLPublish.OnAfterPublishingWSDL 7245
WSDLPub.TWSDLPublish.OnBeforePublishingTypes 7245
WSDLPub.TWSDLPublish.OnBeforePublishingWSDL 7246
WSDLPub.TWSDLPublish.OnPublishingType 7246
WSDLPub.TWSDLPublish.TargetNamespace 7245
WSDLPub.WSDLPubFactory 7249
WSDLPub.WSDLPubFactory function 7249

X

xdsFile enumeration member 7447
xdsNone enumeration member 7447
xdsStream enumeration member 7447
xdsXMLData enumeration member 7447
xdsXMLProperty enumeration member 7447
xetEBCDICLike enumeration member 7665
xetUCS_4BE enumeration member 7665
xetUCS_4LE enumeration member 7665
xetUCS_4Like enumeration member 7665
xetUCS_4Order2134 enumeration member 7665
xetUCS_4Order3412 enumeration member 7665
xetUnknown enumeration member 7665
xetUTF_16BE enumeration member 7665
xetUTF_16BELike enumeration member 7665
xetUTF_16LE enumeration member 7665
xetUTF_16LELike enumeration member 7665
xetUTF_8 enumeration member 7665
xetUTF_8Like enumeration member 7665
xilCUSTOM enumeration member 1631
xilDIRTYREAD enumeration member 1631
xilREADCOMMITTED enumeration member 1631
xilREPEATABLEREAD enumeration member 1631
XMLBrokr 7253
XMLBrokr namespace 7253
  Classes 7255
  Constants 7295
  Functions 7291
  Interfaces 7289
  Structs, Records, Enums 7292
  Types 7293
XMLBrokr.FormatXML 7291
XMLBrokr.FormatXML function 7291
XMLBrokr.INotifyConnectionChange 7289
XMLBrokr.INotifyConnectionChange.ConnectionChange 7290
XMLBrokr.sPostDelta 7295
XMLBrokr.sPostDelta constant 7295
XMLBrokr.sProducer 7295

XMLBrokr.sProducer constant 7295
XMLBrokr.sRedirect 7295
XMLBrokr.sRedirect constant 7295
XMLBrokr.TGetErrorResponseEvent 7293
XMLBrokr.TGetErrorResponseEvent type 7293
XMLBrokr.TGetResponseEvent 7293
XMLBrokr.TGetResponseEvent type 7293
XMLBrokr.TRequestRecordsEvent 7294
XMLBrokr.TRequestRecordsEvent type 7294
XMLBrokr.TRequestUpdateEvent 7294
XMLBrokr.TRequestUpdateEvent type 7294
XMLBrokr.TXMLBroker 7255
XMLBrokr.TXMLBroker.AddNotify 7259
XMLBrokr.TXMLBroker.AfterDispatch 7281
XMLBrokr.TXMLBroker.ApplyXMLUpdates 7260
XMLBrokr.TXMLBroker.AppServer 7274
XMLBrokr.TXMLBroker.AS_ApplyUpdates 7260
XMLBrokr.TXMLBroker.AS_FetchParams 7261
XMLBrokr.TXMLBroker.AS_GetRecords 7261
XMLBrokr.TXMLBroker.BeforeDispatch 7282
XMLBrokr.TXMLBroker.CheckInactive 7261
XMLBrokr.TXMLBroker.Connected 7274
XMLBrokr.TXMLBroker.Create 7257
XMLBrokr.TXMLBroker.Destroy 7258
XMLBrokr.TXMLBroker.DispatchEnabled 7261
XMLBrokr.TXMLBroker.DispatchMask 7261
XMLBrokr.TXMLBroker.DispatchMethodType 7262
XMLBrokr.TXMLBroker.DispatchRequest 7262
XMLBrokr.TXMLBroker.FetchParams 7262
XMLBrokr.TXMLBroker.GetAppServer 7262
XMLBrokr.TXMLBroker.GetConnected 7263
XMLBrokr.TXMLBroker.GetDelta 7263
XMLBrokr.TXMLBroker.GetErrorCount 7263
XMLBrokr.TXMLBroker.GetErrorResponse 7263
XMLBrokr.TXMLBroker.GetErrors 7264
XMLBrokr.TXMLBroker.GetHasAppServer 7264
XMLBrokr.TXMLBroker.GetNotify 7264
XMLBrokr.TXMLBroker.GetNotifyCount 7265
XMLBrokr.TXMLBroker.GetProducerName 7265
XMLBrokr.TXMLBroker.GetRedirect 7265
XMLBrokr.TXMLBroker.GetResponse 7265
XMLBrokr.TXMLBroker.GetXMLRecords 7265
XMLBrokr.TXMLBroker.GetXMLStream 7266
XMLBrokr.TXMLBroker.HasAppServer 7275
XMLBrokr.TXMLBroker.HTMLSubmitFormName 7266
XMLBrokr.TXMLBroker.MasterRowSetVarName 7267
XMLBrokr.TXMLBroker.MaxErrors 7275
XMLBrokr.TXMLBroker.MaxRecords 7276
XMLBrokr.TXMLBroker.Notification 7267
XMLBrokr.TXMLBroker.Notify 7276
XMLBrokr.TXMLBroker.NotifyCount 7277
XMLBrokr.TXMLBroker.OnGetErrorResponse 7282
XMLBrokr.TXMLBroker.OnGetResponse 7283
XMLBrokr.TXMLBroker.OnRequestRecords 7284
XMLBrokr.TXMLBroker.OnRequestUpdate 7284
XMLBrokr.TXMLBroker.Params 7277
XMLBrokr.TXMLBroker.ProviderName 7278
XMLBrokr.TXMLBroker.ReconcileProducer 7279
XMLBrokr.TXMLBroker.RemoteServer 7280
XMLBrokr.TXMLBroker.RemoveNotify 7268
XMLBrokr.TXMLBroker.RequestRecords 7268
XMLBrokr.TXMLBroker.RequestUpdate 7269
XMLBrokr.TXMLBroker.RowSetVarName 7270
XMLBrokr.TXMLBroker.SendConnectEvent 7271
XMLBrokr.TXMLBroker.SetAppServer 7271
XMLBrokr.TXMLBroker.SetConnected 7271
XMLBrokr.TXMLBroker.SetParams 7271
XMLBrokr.TXMLBroker.SetProvider 7272
XMLBrokr.TXMLBroker.SetProviderName 7272
XMLBrokr.TXMLBroker.SetReconcileProducer 7272
XMLBrokr.TXMLBroker.SetRemoteServer 7272
XMLBrokr.TXMLBroker.SetWebDispatch 7273
XMLBrokr.TXMLBroker.SubmitFormVarName 7273
XMLBrokr.TXMLBroker.WebDispatch 7280
XMLBrokr.TXMLOption 7292
XMLBrokr.TXMLOption enumeration 7292
XMLBrokr.TXMLOptions 7294
XMLBrokr.TXMLOptions type 7294

XMLBrokr.TXMLParams 7285
XMLBrokr.TXMLParams.AssignStrings 7287
XMLBrokr.TXMLParams.AssignTo 7287
XMLDoc 7297
XMLDoc namespace 7297
Classes 7299
Functions 7439
Interfaces 7429
Structs, Records, Enums 7445
Types 7449
Variables 7453
XMLDoc.CheckEncoding 7439
XMLDoc.CheckEncoding function 7439
XMLDoc.CreateDOMNode 7439
XMLDoc.CreateDOMNode function 7439
XMLDoc.DetectCharEncoding 7440
XMLDoc.DetectCharEncoding function 7440
XMLDoc.FormatXMLData 7441
XMLDoc.FormatXMLData function 7441
XMLDoc.IXMLDocumentAccess 7429
XMLDoc.IXMLDocumentAccess.DocumentObject 7430
XMLDoc.IXMLDocumentAccess.DOMPersist 7431
XMLDoc.IXMLDocumentAccess.GetDocumentObject 7430
XMLDoc.IXMLDocumentAccess.GetDOMPersist 7430
XMLDoc.IXMLNodeAccess 7431
XMLDoc.IXMLNodeAccess.AddChild 7433
XMLDoc.IXMLNodeAccess.CheckTextNode 7434
XMLDoc.IXMLNodeAccess.ChildNodeClasses 7438
XMLDoc.IXMLNodeAccess.ClearDocumentRef 7434
XMLDoc.IXMLNodeAccess.CreateAttributeNode 7434
XMLDoc.IXMLNodeAccess.CreateChildNode 7434
XMLDoc.IXMLNodeAccess.CreateCollection 7434
XMLDoc.IXMLNodeAccess.DOMElement 7435
XMLDoc.IXMLNodeAccess.FindHostedNode 7435
XMLDoc.IXMLNodeAccess.GetChildNodeClasses 7435
XMLDoc.IXMLNodeAccess.GetHostNode 7435
XMLDoc.IXMLNodeAccess.GetNodeObject 7436
XMLDoc.IXMLNodeAccess.HasChildNode 7436
XMLDoc.IXMLNodeAccess.HostNode 7438
XMLDoc.IXMLNodeAccess.InternalAddChild 7436
XMLDoc.IXMLNodeAccess.NestingLevel 7437
XMLDoc.IXMLNodeAccess.RegisterChildNode 7437
XMLDoc.IXMLNodeAccess.RegisterChildNodes 7437
XMLDoc.IXMLNodeAccess.SetCollection 7437
XMLDoc.IXMLNodeAccess.SetParentNode 7438
XMLDoc.LoadXMLData 7441
XMLDoc.LoadXMLData function 7441
XMLDoc.LoadXMLDocument 7442
XMLDoc.LoadXMLDocument function 7442
XMLDoc.NewXMLDocument 7442
XMLDoc.NewXMLDocument function 7442
XMLDoc.TNodeChange 7445
XMLDoc.TNodeChange enumeration 7445
XMLDoc.TNodeChangeEvent 7449
XMLDoc.TNodeChangeEvent type 7449
XMLDoc.TNodeClassArray 7449
XMLDoc.TNodeClassArray type 7449
XMLDoc.TNodeClassInfo 7446
XMLDoc.TNodeClassInfo record 7446
XMLDoc.TNodeListNotification 7450
XMLDoc.TNodeListNotification type 7450
XMLDoc.TNodeListOperation 7446
XMLDoc.TNodeListOperation enumeration 7446
XMLDoc.TXMLDocument 7299
XMLDoc.TXMLDocument._AddRef 7306
XMLDoc.TXMLDocument._Release 7306
XMLDoc.TXMLDocument.Active 7333
XMLDoc.TXMLDocument.AddChild 7307
XMLDoc.TXMLDocument.AfterClose 7344
XMLDoc.TXMLDocument.AfterConstruction 7307
XMLDoc.TXMLDocument.AfterNodeChange 7345
XMLDoc.TXMLDocument.AfterOpen 7345
XMLDoc.TXMLDocument.AssignParseOptions 7308
XMLDoc.TXMLDocument.AsyncLoadState 7334
XMLDoc.TXMLDocument.BeforeClose 7346
XMLDoc.TXMLDocument.BeforeNodeChange 7346
XMLDoc.TXMLDocument.BeforeOpen 7347
XMLDoc.TXMLDocument.CheckActive 7308

XMLDoc.TXMLDocument.CheckAutoSave 7308
XMLDoc.TXMLDocument.CheckDOM 7309
XMLDoc.TXMLDocument.ChildNodes 7335
XMLDoc.TXMLDocument.Create 7303
XMLDoc.TXMLDocument.CreateElement 7309
XMLDoc.TXMLDocument.CreateNode 7309
XMLDoc.TXMLDocument.DefineProperties 7311
XMLDoc.TXMLDocument.Destroy 7304
XMLDoc.TXMLDocument.DoAfterClose 7311
XMLDoc.TXMLDocument.DoAfterOpen 7311
XMLDoc.TXMLDocument.DoBeforeClose 7312
XMLDoc.TXMLDocument.DoBeforeOpen 7312
XMLDoc.TXMLDocument.DocBindingInfo 7335
XMLDoc.TXMLDocument.DocSource 7335
XMLDoc.TXMLDocument.DocumentElement 7335
XMLDoc.TXMLDocument.DOMDocument 7336
XMLDoc.TXMLDocument.DOMImplementation 7337
XMLDoc.TXMLDocument.DOMParseOptions 7337
XMLDoc.TXMLDocument.DOMPersist 7337
XMLDoc.TXMLDocument.DOMVendor 7337
XMLDoc.TXMLDocument.DoNodeChange 7312
XMLDoc.TXMLDocument.Encoding 7338
XMLDoc.TXMLDocument.FileName 7339
XMLDoc.TXMLDocument.GeneratePrefix 7312
XMLDoc.TXMLDocument.GetActive 7313
XMLDoc.TXMLDocument.GetAsyncLoadState 7313
XMLDoc.TXMLDocument.GetChildNodeClass 7313
XMLDoc.TXMLDocument.GetChildNodes 7313
XMLDoc.TXMLDocument.GetDocBinding 7314
XMLDoc.TXMLDocument.GetDocumentElement 7314
XMLDoc.TXMLDocument.GetDocumentNode 7315
XMLDoc.TXMLDocument.GetDocumentObject 7315
XMLDoc.TXMLDocument.GetDOMDocument 7315
XMLDoc.TXMLDocument.GetDOMPersist 7315
XMLDoc.TXMLDocument.GetEncoding 7316
XMLDoc.TXMLDocument.GetFileName 7316
XMLDoc.TXMLDocument.GetModified 7316
XMLDoc.TXMLDocument.GetNodeIndentStr 7317
XMLDoc.TXMLDocument.GetOptions 7317
XMLDoc.TXMLDocument.GetParseOptions 7317
XMLDoc.TXMLDocument.GetPrologNode 7317
XMLDoc.TXMLDocument.GetPrologValue 7318
XMLDoc.TXMLDocument.GetSchemaRef 7318
XMLDoc.TXMLDocument.GetStandAlone 7318
XMLDoc.TXMLDocument.GetVersion 7318
XMLDoc.TXMLDocument.GetXML 7319
XMLDoc.TXMLDocument.InternalSetPrologValue 7319
XMLDoc.TXMLDocument.IsEmptyDoc 7319
XMLDoc.TXMLDocument.LoadData 7320
XMLDoc.TXMLDocument.Loaded 7320
XMLDoc.TXMLDocument.LoadFromFile 7321
XMLDoc.TXMLDocument.LoadFromStream 7321
XMLDoc.TXMLDocument.LoadFromXML 7322
XMLDoc.TXMLDocument.Modified 7339
XMLDoc.TXMLDocument.NewInstance 7322
XMLDoc.TXMLDocument.Node 7340
XMLDoc.TXMLDocument.NodeIndentStr 7340
XMLDoc.TXMLDocument.NSPrefixBase 7341
XMLDoc.TXMLDocument.OnAsyncLoad 7347
XMLDoc.TXMLDocument.Options 7341
XMLDoc.TXMLDocument.ParseOptions 7342
XMLDoc.TXMLDocument.PrefixID 7342
XMLDoc.TXMLDocument.ReadDOMVendor 7323
XMLDoc.TXMLDocument.Refresh 7323
XMLDoc.TXMLDocument.RegisterDocBinding 7324
XMLDoc.TXMLDocument.ReleaseDoc 7324
XMLDoc.TXMLDocument.Resync 7325
XMLDoc.TXMLDocument.SaveToFile 7325
XMLDoc.TXMLDocument.SaveToStream 7325
XMLDoc.TXMLDocument.SaveToUTF8String 7326
XMLDoc.TXMLDocument.SaveToXML 7326
XMLDoc.TXMLDocument.SaveToXMLStrings 7327
XMLDoc.TXMLDocument.SchemaRef 7343
XMLDoc.TXMLDocument.SetActive 7327
XMLDoc.TXMLDocument.SetDocumentElement 7327
XMLDoc.TXMLDocument.SetDOMDocument 7328
XMLDoc.TXMLDocument.SetEncoding 7328
XMLDoc.TXMLDocument.SetFileName 7328

XMLDoc.TXMLDocument.SetModified 7329
XMLDoc.TXMLDocument.SetNodeIndentStr 7329
XMLDoc.TXMLDocument.SetOnAsyncLoad 7329
XMLDoc.TXMLDocument.SetOptions 7330
XMLDoc.TXMLDocument.SetParseOptions 7330
XMLDoc.TXMLDocument.SetPrologValue 7330
XMLDoc.TXMLDocument.SetStandAlone 7330
XMLDoc.TXMLDocument.SetVersion 7331
XMLDoc.TXMLDocument.SetXML 7331
XMLDoc.TXMLDocument.SetXMLStrings 7331
XMLDoc.TXMLDocument.StandAlone 7343
XMLDoc.TXMLDocument.Version 7343
XMLDoc.TXMLDocument.WriteDOMVendor 7332
XMLDoc.TXMLDocument.XML 7344
XMLDoc.TXMLDocument.XMLStringsChanging 7332
XMLDoc.TXMLDocumentSource 7447
XMLDoc.TXMLDocumentSource enumeration 7447
XMLDoc.TXMLNode 7348
XMLDoc.TXMLNode._AddRef 7355
XMLDoc.TXMLNode._Release 7356
XMLDoc.TXMLNode.AddChild 7356
XMLDoc.TXMLNode.AddHostedNode 7357
XMLDoc.TXMLNode.AttributeListNotify 7358
XMLDoc.TXMLNode.AttributeNodes 7390
XMLDoc.TXMLNode.CheckNotHosted 7358
XMLDoc.TXMLNode.CheckReadOnly 7359
XMLDoc.TXMLNode.CheckTextNode 7359
XMLDoc.TXMLNode.ChildListNotify 7360
XMLDoc.TXMLNode.ChildNodeClasses 7390
XMLDoc.TXMLNode.ChildNodes 7391
XMLDoc.TXMLNode.ClearDocumentRef 7360
XMLDoc.TXMLNode.CloneNode 7361
XMLDoc.TXMLNode.Collection 7391
XMLDoc.TXMLNode.Create 7352
XMLDoc.TXMLNode.CreateAttributeList 7361
XMLDoc.TXMLNode.CreateAttributeNode 7362
XMLDoc.TXMLNode.CreateChildList 7362
XMLDoc.TXMLNode.CreateChildNode 7362
XMLDoc.TXMLNode.CreateCollection 7363
XMLDoc.TXMLNode.CreateHosted 7353
XMLDoc.TXMLNode.DeclareNamespace 7364
XMLDoc.TXMLNode.Destroy 7353
XMLDoc.TXMLNode.DOMElement 7364
XMLDoc.TXMLNode.DOMNode 7392
XMLDoc.TXMLNode.DoNodeChange 7365
XMLDoc.TXMLNode.FindHostedNode 7365
XMLDoc.TXMLNode.FindNamespaceDecl 7366
XMLDoc.TXMLNode.FindNamespaceURI 7366
XMLDoc.TXMLNode.GetAttribute 7367
XMLDoc.TXMLNode.GetAttributeNodes 7367
XMLDoc.TXMLNode.GetAttributeNS 7368
XMLDoc.TXMLNode.GetChildNodeClasses 7368
XMLDoc.TXMLNode.GetChildNodes 7369
XMLDoc.TXMLNode.GetChildValue 7369
XMLDoc.TXMLNode.GetCollection 7370
XMLDoc.TXMLNode.GetDOMNode 7370
XMLDoc.TXMLNode.GetHasChildNodes 7370
XMLDoc.TXMLNode.GetHostedNodes 7371
XMLDoc.TXMLNode.GetHostNode 7371
XMLDoc.TXMLNode.GetIsTextElement 7371
XMLDoc.TXMLNode.GetLocalName 7372
XMLDoc.TXMLNode.GetNamespaceURI 7373
XMLDoc.TXMLNode.GetNodeName 7373
XMLDoc.TXMLNode.GetNodeObject 7374
XMLDoc.TXMLNode.GetNodeType 7374
XMLDoc.TXMLNode.GetNodeValue 7375
XMLDoc.TXMLNode.GetOwnerDocument 7375
XMLDoc.TXMLNode.GetParentNode 7376
XMLDoc.TXMLNode.GetPrefix 7376
XMLDoc.TXMLNode.GetPrefixedName 7377
XMLDoc.TXMLNode.GetReadOnly 7377
XMLDoc.TXMLNode.GetText 7378
XMLDoc.TXMLNode.GetXML 7378
XMLDoc.TXMLNode.HasAttribute 7379
XMLDoc.TXMLNode.HasChildNode 7379
XMLDoc.TXMLNode.HostedNodes 7393
XMLDoc.TXMLNode.HostNode 7393
XMLDoc.TXMLNode.InternalAddChild 7380

XMLDoc.TXMLNode.NestingLevel 7380
XMLDoc.TXMLNode.NextSibling 7381
XMLDoc.TXMLNode.Normalize 7381
XMLDoc.TXMLNode.OnHostAttrNotify 7395
XMLDoc.TXMLNode.OnHostChildNotify 7396
XMLDoc.TXMLNode.OwnerDocument 7394
XMLDoc.TXMLNode.ParentNode 7395
XMLDoc.TXMLNode.PreviousSibling 7382
XMLDoc.TXMLNode.RegisterChildNode 7382
XMLDoc.TXMLNode.RegisterChildNodes 7383
XMLDoc.TXMLNode.RemoveHostedNode 7383
XMLDoc.TXMLNode.Resync 7384
XMLDoc.TXMLNode.SetAttribute 7384
XMLDoc.TXMLNode.SetAttributeNodes 7384
XMLDoc.TXMLNode.SetAttributeNS 7385
XMLDoc.TXMLNode.SetChildNodes 7385
XMLDoc.TXMLNode.SetChildValue 7386
XMLDoc.TXMLNode.SetCollection 7386
XMLDoc.TXMLNode.SetNodeValue 7387
XMLDoc.TXMLNode.SetParentNode 7387
XMLDoc.TXMLNode.SetReadOnly 7388
XMLDoc.TXMLNode.SetText 7388
XMLDoc.TXMLNode.TransformNode 7389
XMLDoc.TXMLNodeArray 7451
XMLDoc.TXMLNodeArray type 7451
XMLDoc.TXMLNodeClass 7451
XMLDoc.TXMLNodeClass type 7451
XMLDoc.TXMLNodeCollection 7396
XMLDoc.TXMLNodeCollection.AddItem 7403
XMLDoc.TXMLNodeCollection.AfterConstruction 7403
XMLDoc.TXMLNodeCollection.ChildListNotify 7404
XMLDoc.TXMLNodeCollection.Clear 7404
XMLDoc.TXMLNodeCollection.Count 7409
XMLDoc.TXMLNodeCollection.CreateItemList 7405
XMLDoc.TXMLNodeCollection.Delete 7405
XMLDoc.TXMLNodeCollection.GetCount 7406
XMLDoc.TXMLNodeCollection.GetList 7406
XMLDoc.TXMLNodeCollection.GetNode 7406
XMLDoc.TXMLNodeCollection.InsertInCollection 7407
XMLDoc.TXMLNodeCollection.IsCollectionItem 7407
XMLDoc.TXMLNodeCollection.ItemInterface 7410
XMLDoc.TXMLNodeCollection.ItemNS 7410
XMLDoc.TXMLNodeCollection.ItemTag 7411
XMLDoc.TXMLNodeCollection.List 7411
XMLDoc.TXMLNodeCollection.Remove 7408
XMLDoc.TXMLNodeCollection.SetChildNodes 7408
XMLDoc.TXMLNodeCollection.UpdateCollectionList 7409
XMLDoc.TXMLNodeCollectionClass 7452
XMLDoc.TXMLNodeCollectionClass type 7452
XMLDoc.TXMLNodeList 7411
XMLDoc.TXMLNodeList.Add 7416
XMLDoc.TXMLNodeList.BeginUpdate 7416
XMLDoc.TXMLNodeList.Clear 7417
XMLDoc.TXMLNodeList.Count 7426
XMLDoc.TXMLNodeList.Create 7414
XMLDoc.TXMLNodeList.DefaultNamespaceURI 7426
XMLDoc.TXMLNodeList.Delete 7417
XMLDoc.TXMLNodeList.Destroy 7415
XMLDoc.TXMLNodeList.DoNotify 7418
XMLDoc.TXMLNodeList.EndUpdate 7419
XMLDoc.TXMLNodeList.FindNode 7419
XMLDoc.TXMLNodeList.FindSibling 7420
XMLDoc.TXMLNodeList.First 7420
XMLDoc.TXMLNodeList.Get 7421
XMLDoc.TXMLNodeList.GetCount 7421
XMLDoc.TXMLNodeList.GetNode 7422
XMLDoc.TXMLNodeList.GetUpdateCount 7422
XMLDoc.TXMLNodeList.IndexOf 7423
XMLDoc.TXMLNodeList.Insert 7423
XMLDoc.TXMLNodeList.InternalInsert 7424
XMLDoc.TXMLNodeList.Last 7424
XMLDoc.TXMLNodeList.List 7426
XMLDoc.TXMLNodeList.NotificationProc 7428
XMLDoc.TXMLNodeList.Owner 7427
XMLDoc.TXMLNodeList.Remove 7424
XMLDoc.TXMLNodeList.ReplaceNode 7425
XMLDoc.TXMLNodeList.UpdateCount 7427
XMLDoc.TXMLPrologItem 7448

XMLDoc.TXMLPrologItem enumeration 7448
XMLDoc.UseXSDBooleanStrings 7453
XMLDoc.UseXSDBooleanStrings variable 7453
XMLDoc.XMLDocError 7443
XMLDoc.XMLDocError function 7443
XMLDoc.XMLStringToUnicodeString 7443
XMLDoc.XMLStringToUnicodeString function 7443
XMLDoc.XMLStringToWideString 7443
XMLDoc.XMLStringToWideString function 7443
xmldom 7455
xmldom namespace 7455
Classes 7458
Constants 7569
Functions 7557
Interfaces 7473
Types 7565
Variables 7568
xmldom.ATTRIBUTE_NODE 7569
xmldom.ATTRIBUTE_NODE constant 7569
xmldom.CDATA_SECTION_NODE 7570
xmldom.CDATA_SECTION_NODE constant 7570
xmldom.COMMENT_NODE 7570
xmldom.COMMENT_NODE constant 7570
xmldom.DefaultDOMVendor 7568
xmldom.DefaultDOMVendor variable 7568
xmldom.DOCUMENT_FRAGMENT_NODE 7570
xmldom.DOCUMENT_FRAGMENT_NODE constant 7570
xmldom.DOCUMENT_NODE 7571
xmldom.DOCUMENT_NODE constant 7571
xmldom.DOCUMENT_TYPE_NODE 7571
xmldom.DOCUMENT_TYPE_NODE constant 7571
xmldom.DOMException 7458
xmldom.DOMException.code 7460
xmldom.DOMNodeType 7565
xmldom.DOMNodeType type 7565
xmldom.DOMString 7565
xmldom.DOMString type 7565
xmldom.DOMSTRING_SIZE_ERR 7571
xmldom.DOMSTRING_SIZE_ERR constant 7571
xmldom.DOMTimeStamp 7566
xmldom.DOMTimeStamp type 7566
xmldom.DOMVendorNotSupported 7557
xmldom.DOMVendorNotSupported function 7557
xmldom.DOMVendors 7568
xmldom.DOMVendors variable 7568
xmldom.DOMWrapperVersion 7572
xmldom.DOMWrapperVersion constant 7572
xmldom.EDOMParseError 7460
xmldom.EDOMParseError.Create 7462
xmldom.EDOMParseError.ErrorCode 7463
xmldom.EDOMParseError.FilePos 7463
xmldom.EDOMParseError.Line 7464
xmldom.EDOMParseError.LinePos 7464
xmldom.EDOMParseError.ParseError 7464
xmldom.EDOMParseError.Reason 7465
xmldom.EDOMParseError.SrcText 7465
xmldom.EDOMParseError.URL 7465
xmldom.ELEMENT_NODE 7572
xmldom.ELEMENT_NODE constant 7572
xmldom.ENTITY_NODE 7572
xmldom.ENTITY_NODE constant 7572
xmldom.ENTITY_REFERENCE_NODE 7573
xmldom.ENTITY_REFERENCE_NODE constant 7573
xmldom.ExtractLocalName 7557
xmldom.ExtractLocalName function 7557
xmldom.ExtractPrefix 7558
xmldom.ExtractPrefix function 7558
xmldom.GetDOM 7559
xmldom.GetDOM function 7559
xmldom.GetDOMNodeEx 7559
xmldom.GetDOMNodeEx function 7559
xmldom.GetDOMVendor 7560
xmldom.GetDOMVendor function 7560
xmldom.HIERARCHY_REQUEST_ERR 7573
xmldom.HIERARCHY_REQUEST_ERR constant 7573
xmldom.IDOMAttr 7473
xmldom.IDOMAttr.get_name 7475
xmldom.IDOMAttr.get_ownerElement 7475

xmldom.IDOMAttr.get_specified 7475
xmldom.IDOMAttr.get_value 7476
xmldom.IDOMAttr.name 7476
xmldom.IDOMAttr.ownerElement 7476
xmldom.IDOMAttr.set_value 7476
xmldom.IDOMAttr.specified 7477
xmldom.IDOMAttr.value 7477
xmldom.IDOMCDATASection 7477
xmldom.IDOMCharacterData 7478
xmldom.IDOMCharacterData.appendData 7479
xmldom.IDOMCharacterData.data 7481
xmldom.IDOMCharacterData.deleteData 7480
xmldom.IDOMCharacterData.get_data 7480
xmldom.IDOMCharacterData.get_length 7480
xmldom.IDOMCharacterData.insertData 7480
xmldom.IDOMCharacterData.length 7481
xmldom.IDOMCharacterData.replaceData 7480
xmldom.IDOMCharacterData.set_data 7481
xmldom.IDOMCharacterData.substringData 7481
xmldom.IDOMComment 7482
xmldom.IDOMDocument 7482
xmldom.IDOMDocument.createAttribute 7484
xmldom.IDOMDocument.createAttributeNS 7485
xmldom.IDOMDocument.createCDATASection 7485
xmldom.IDOMDocument.createComment 7485
xmldom.IDOMDocument.createDocumentFragment 7485
xmldom.IDOMDocument.createElement 7485
xmldom.IDOMDocument.createElementNS 7486
xmldom.IDOMDocument.createEntityReference 7486
xmldom.IDOMDocument.createProcessingInstruction 7486
xmldom.IDOMDocument.createTextNode 7486
xmldom.IDOMDocument.doctype 7489
xmldom.IDOMDocument.documentElement 7489
xmldom.IDOMDocument.domImplementation 7489
xmldom.IDOMDocument.get_doctype 7487
xmldom.IDOMDocument.get_documentElement 7487
xmldom.IDOMDocument.get_domImplementation 7487
xmldom.IDOMDocument.getElementById 7487
xmldom.IDOMDocument.getElementsByTagName 7488
xmldom.IDOMDocument.getElementsByTagNameNS 7488
xmldom.IDOMDocument.importNode 7488
xmldom.IDOMDocument.set_documentElement 7488
xmldom.IDOMDocumentFragment 7489
xmldom.IDOMDocumentType 7490
xmldom.IDOMDocumentType.entities 7493
xmldom.IDOMDocumentType.get_entities 7492
xmldom.IDOMDocumentType.get_internalSubset 7492
xmldom.IDOMDocumentType.get_name 7492
xmldom.IDOMDocumentType.get_notations 7492
xmldom.IDOMDocumentType.get_publicId 7492
xmldom.IDOMDocumentType.get_systemId 7493
xmldom.IDOMDocumentType.internalSubset 7493
xmldom.IDOMDocumentType.name 7494
xmldom.IDOMDocumentType.notations 7494
xmldom.IDOMDocumentType.publicId 7494
xmldom.IDOMDocumentType.systemId 7494
xmldom.IDOMElement 7494
xmldom.IDOMElement.get_tagName 7497
xmldom.IDOMElement.getAttribute 7497
xmldom.IDOMElement.getAttributeNode 7497
xmldom.IDOMElement.getAttributeNodeNS 7497
xmldom.IDOMElement.getAttributeNS 7497
xmldom.IDOMElement.getElementsByTagName 7498
xmldom.IDOMElement.getElementsByTagNameNS 7498
xmldom.IDOMElement.hasAttribute 7498
xmldom.IDOMElement.hasAttributeNS 7498
xmldom.IDOMElement.removeAttribute 7499
xmldom.IDOMElement.removeAttributeNode 7499
xmldom.IDOMElement.removeAttributeNS 7499
xmldom.IDOMElement.setAttribute 7499
xmldom.IDOMElement.setAttributeNode 7499
xmldom.IDOMElement.setAttributeNodeNS 7500
xmldom.IDOMElement.setAttributeNS 7500
xmldom.IDOMElement.tagName 7500
xmldom.IDOMEntity 7501
xmldom.IDOMEntity.get_notationName 7502
xmldom.IDOMEntity.get_publicId 7502
xmldom.IDOMEntity.get_systemId 7503

xmldom.IDOMEntity.notationName 7503
xmldom.IDOMEntity.publicId 7503
xmldom.IDOMEntity.systemId 7503
xmldom.IDOMEntityReference 7504
xmldom.IDOMImplementation 7504
xmldom.IDOMImplementation.createDocument 7505
xmldom.IDOMImplementation.createDocumentType 7505
xmldom.IDOMImplementation.hasFeature 7505
xmldom.IDOMNamedNodeMap 7506
xmldom.IDOMNamedNodeMap.get_item 7507
xmldom.IDOMNamedNodeMap.get_length 7507
xmldom.IDOMNamedNodeMap.getNamedItem 7507
xmldom.IDOMNamedNodeMap.getNamedItemNS 7507
xmldom.IDOMNamedNodeMap.item 7509
xmldom.IDOMNamedNodeMap.length 7509
xmldom.IDOMNamedNodeMap.removeNamedItem 7508
xmldom.IDOMNamedNodeMap.removeNamedItemNS 7508
xmldom.IDOMNamedNodeMap.setNamedItem 7508
xmldom.IDOMNamedNodeMap.setNamedItemNS 7508
xmldom.IDOMNode 7509
xmldom.IDOMNode.appendChild 7511
xmldom.IDOMNode.attributes 7520
xmldom.IDOMNode.childNodes 7520
xmldom.IDOMNode.cloneNode 7512
xmldom.IDOMNode.firstChild 7520
xmldom.IDOMNode.get_attributes 7512
xmldom.IDOMNode.get_childNodes 7512
xmldom.IDOMNode.get_firstChild 7512
xmldom.IDOMNode.get_lastChild 7513
xmldom.IDOMNode.get_localName 7513
xmldom.IDOMNode.get_namespaceURI 7513
xmldom.IDOMNode.get_nextSibling 7514
xmldom.IDOMNode.get_nodeName 7514
xmldom.IDOMNode.get_nodeType 7514
xmldom.IDOMNode.get_nodeValue 7515
xmldom.IDOMNode.get_ownerDocument 7515
xmldom.IDOMNode.get_parentNode 7515
xmldom.IDOMNode.get_prefix 7515
xmldom.IDOMNode.get_previousSibling 7516
xmldom.IDOMNode.hasChildNodes 7516
xmldom.IDOMNode.insertBefore 7516
xmldom.IDOMNode.lastChild 7520
xmldom.IDOMNode.localName 7521
xmldom.IDOMNode.namespaceURI 7521
xmldom.IDOMNode.nextSibling 7521
xmldom.IDOMNode.nodeName 7521
xmldom.IDOMNode.nodeType 7521
xmldom.IDOMNode.nodeValue 7522
xmldom.IDOMNode.normalize 7517
xmldom.IDOMNode.ownerDocument 7522
xmldom.IDOMNode.parentNode 7522
xmldom.IDOMNode.prefix 7522
xmldom.IDOMNode.previousSibling 7523
xmldom.IDOMNode.removeChild 7518
xmldom.IDOMNode.replaceChild 7518
xmldom.IDOMNode.set_nodeValue 7519
xmldom.IDOMNode.supports 7519
xmldom.IDOMNodeEx 7523
xmldom.IDOMNodeEx.get_text 7524
xmldom.IDOMNodeEx.get_xml 7525
xmldom.IDOMNodeEx.set_text 7525
xmldom.IDOMNodeEx.text 7526
xmldom.IDOMNodeEx.transformNode 7525, 7526
xmldom.IDOMNodeEx.xml 7526
xmldom.IDOMNodeList 7526
xmldom.IDOMNodeList.get_item 7527
xmldom.IDOMNodeList.get_length 7527
xmldom.IDOMNodeList.item 7528
xmldom.IDOMNodeList.length 7528
xmldom.IDOMNodeSelect 7528
xmldom.IDOMNodeSelect.selectNode 7529
xmldom.IDOMNodeSelect.selectNodes 7529
xmldom.IDOMNotation 7530
xmldom.IDOMNotation.get_publicId 7531
xmldom.IDOMNotation.get_systemId 7531
xmldom.IDOMNotation.publicId 7532
xmldom.IDOMNotation.systemId 7532
xmldom.IDOMParseError 7532

xmldom.IDOMParseError.errorCode 7535
xmldom.IDOMParseError.filePos 7535
xmldom.IDOMParseError.get_errorCode 7533
xmldom.IDOMParseError.get_filepos 7534
xmldom.IDOMParseError.get_line 7534
xmldom.IDOMParseError.get_linepos 7534
xmldom.IDOMParseError.get_reason 7534
xmldom.IDOMParseError.get_srcText 7534
xmldom.IDOMParseError.get_url 7535
xmldom.IDOMParseError.line 7536
xmldom.IDOMParseError.linePos 7536
xmldom.IDOMParseError.reason 7536
xmldom.IDOMParseError.srcText 7536
xmldom.IDOMParseError.url 7536
xmldom.IDOMParseOptions 7537
xmldom.IDOMParseOptions.async 7540
xmldom.IDOMParseOptions.get_async 7538
xmldom.IDOMParseOptions.get_preserveWhiteSpace 7538
xmldom.IDOMParseOptions.get_resolveExternals 7538
xmldom.IDOMParseOptions.get_validate 7538
xmldom.IDOMParseOptions.preserveWhiteSpace 7540
xmldom.IDOMParseOptions.resolveExternals 7540
xmldom.IDOMParseOptions.set_async 7539
xmldom.IDOMParseOptions.set_preserveWhiteSpace 7539
xmldom.IDOMParseOptions.set_resolveExternals 7539
xmldom.IDOMParseOptions.set_validate 7539
xmldom.IDOMParseOptions.validate 7540
xmldom.IDOMPersist 7541
xmldom.IDOMPersist.asyncLoadState 7542
xmldom.IDOMPersist.get_xml 7542
xmldom.IDOMPersist.load 7542
xmldom.IDOMPersist.loadFromStream 7542, 7543
xmldom.IDOMPersist.loadxml 7543
xmldom.IDOMPersist.save 7543
xmldom.IDOMPersist.saveToStream 7544
xmldom.IDOMPersist.set_OnAsyncLoad 7544
xmldom.IDOMPersist.xml 7544
xmldom.IDOMProcessingInstruction 7545
xmldom.IDOMProcessingInstruction.data 7547
xmldom.IDOMProcessingInstruction.get_data 7546
xmldom.IDOMProcessingInstruction.get_target 7546
xmldom.IDOMProcessingInstruction.set_data 7547
xmldom.IDOMProcessingInstruction.target 7547
xmldom.IDOMText 7548
xmldom.IDOMText.splitText 7549
xmldom.IDOMXMLProlog 7550
xmldom.IDOMXMLProlog.Encoding 7552
xmldom.IDOMXMLProlog.get_Encoding 7551
xmldom.IDOMXMLProlog.get_Standalone 7551
xmldom.IDOMXMLProlog.get_Version 7551
xmldom.IDOMXMLProlog.set_Encoding 7551
xmldom.IDOMXMLProlog.set_Standalone 7551
xmldom.IDOMXMLProlog.set_Version 7552
xmldom.IDOMXMLProlog.Standalone 7552
xmldom.IDOMXMLProlog.Version 7552
xmldom.IDOMXSLProcessor 7553
xmldom.IDOMXSLProcessor.Get_input 7554
xmldom.IDOMXSLProcessor.Get_output 7554
xmldom.IDOMXSLProcessor.Get_stylesheet 7554
xmldom.IDOMXSLProcessor.input 7556
xmldom.IDOMXSLProcessor.output 7556
xmldom.IDOMXSLProcessor.reset 7555
xmldom.IDOMXSLProcessor.Set_input 7555
xmldom.IDOMXSLProcessor.Set_output 7555
xmldom.IDOMXSLProcessor.setParameter 7555
xmldom.IDOMXSLProcessor.stylesheet 7556
xmldom.IDOMXSLProcessor.transform 7555
xmldom.INDEX_SIZE_ERR 7573
xmldom.INDEX_SIZE_ERR constant 7573
xmldom.INUSE_ATTRIBUTE_ERR 7574
xmldom.INUSE_ATTRIBUTE_ERR constant 7574
xmldom.INVALID_ACCESS_ERR 7574
xmldom.INVALID_ACCESS_ERR constant 7574
xmldom.INVALID_CHARACTER_ERR 7574
xmldom.INVALID_CHARACTER_ERR constant 7574
xmldom.INVALID_MODIFICATION_ERR 7575
xmldom.INVALID_MODIFICATION_ERR constant 7575
xmldom.INVALID_STATE_ERR 7575

xmldom.INVALID_STATE_ERR constant 7575
xmldom.IsPrefixed 7560
xmldom.IsPrefixed function 7560
xmldom.MakeNodeName 7561
xmldom.MakeNodeName function 7561
xmldom.NAMESPACE_ERR 7575
xmldom.NAMESPACE_ERR constant 7575
xmldom.NO_DATA_ALLOWED_ERR 7576
xmldom.NO_DATA_ALLOWED_ERR constant 7576
xmldom.NO_MODIFICATION_ALLOWED_ERR 7576
xmldom.NO_MODIFICATION_ALLOWED_ERR constant 7576
xmldom.NodeMatches 7561
xmldom.NodeMatches function 7561
xmldom.NOT_FOUND_ERR 7576
xmldom.NOT_FOUND_ERR constant 7576
xmldom.NOT_SUPPORTED_ERR 7577
xmldom.NOT_SUPPORTED_ERR constant 7577
xmldom.NOTATION_NODE 7577
xmldom.NOTATION_NODE constant 7577
xmldom.NSDelim 7577
xmldom.NSDelim constant 7577
xmldom.PROCESSING_INSTRUCTION_NODE 7578
xmldom.PROCESSING_INSTRUCTION_NODE constant 7578
xmldom.RegisterDOMVendor 7562
xmldom.RegisterDOMVendor function 7562
xmldom.SameNamespace 7562
xmldom.SameNamespace function 7562
xmldom.SEncoding 7578
xmldom.SEncoding constant 7578
xmldom.SHttp 7578
xmldom.SHttp constant 7578
xmldom.SStandalone 7579
xmldom.SStandalone constant 7579
xmldom.SVersion 7579
xmldom.SVersion constant 7579
xmldom.SXML 7579
xmldom.SXML constant 7579
xmldom.SXMLNamespaceURI 7580
xmldom.SXMLNamespaceURI constant 7580
xmldom.SXMLNS 7580
xmldom.SXMLNS constant 7580
xmldom.SXMLPrefixNamespaceURI 7580
xmldom.SXMLPrefixNamespaceURI constant 7580
xmldom.SYNTAX_ERR 7581
xmldom.SYNTAX_ERR constant 7581
xmldom.TAsyncEventHandler 7566
xmldom.TAsyncEventHandler type 7566
xmldom.TDOMVendor 7466
xmldom.TDOMVendor.Description 7467
xmldom.TDOMVendor.DOMImplementation 7467
xmldom.TDOMVendorArray 7567
xmldom.TDOMVendorArray type 7567
xmldom.TDOMVendorList 7468
xmldom.TDOMVendorList.Add 7469
xmldom.TDOMVendorList.Count 7469
xmldom.TDOMVendorList.Find 7470
xmldom.TDOMVendorList.GetVendors 7470
xmldom.TDOMVendorList.Remove 7471
xmldom.TDOMVendorList.Vendors 7471
xmldom.TEXT_NODE 7581
xmldom.TEXT_NODE constant 7581
xmldom.UnRegisterDOMVendor 7563
xmldom.UnRegisterDOMVendor function 7563
xmldom.WRONG_DOCUMENT_ERR 7581
xmldom.WRONG_DOCUMENT_ERR constant 7581
XMLIntf 7583
XMLIntf namespace 7583
Classes 7585
Constants 7668
Interfaces 7586
Structs, Records, Enums 7662
Types 7667
XMLIntf.EXMLDocError 7585
XMLIntf.IXMLDocument 7586
XMLIntf.IXMLDocument.Active 7607
XMLIntf.IXMLDocument.AddChild 7589
XMLIntf.IXMLDocument.AsyncLoadState 7608
XMLIntf.IXMLDocument.ChildNodes 7609

XMLIntf.IXMLDocument.CreateElement 7590
XMLIntf.IXMLDocument.CreateNode 7591
XMLIntf.IXMLDocument.DocumentElement 7609
XMLIntf.IXMLDocument.DOMDocument 7609
XMLIntf.IXMLDocument.Encoding 7610
XMLIntf.IXMLDocument.FileName 7610
XMLIntf.IXMLDocument.GeneratePrefix 7592
XMLIntf.IXMLDocument.GetActive 7592
XMLIntf.IXMLDocument.GetAsyncLoadState 7593
XMLIntf.IXMLDocument.GetChildNodes 7593
XMLIntf.IXMLDocument.GetDocBinding 7593
XMLIntf.IXMLDocument.GetDocumentElement 7594
XMLIntf.IXMLDocument.GetDocumentNode 7594
XMLIntf.IXMLDocument.GetDOMDocument 7594
XMLIntf.IXMLDocument.GetEncoding 7595
XMLIntf.IXMLDocument.GetFileName 7595
XMLIntf.IXMLDocument.GetModified 7595
XMLIntf.IXMLDocument.GetNodeIndentStr 7596
XMLIntf.IXMLDocument.GetOptions 7596
XMLIntf.IXMLDocument.GetParseOptions 7596
XMLIntf.IXMLDocument.GetSchemaRef 7596
XMLIntf.IXMLDocument.GetStandAlone 7597
XMLIntf.IXMLDocument.GetVersion 7597
XMLIntf.IXMLDocument.GetXML 7597
XMLIntf.IXMLDocument.IsEmptyDoc 7598
XMLIntf.IXMLDocument.LoadFromFile 7598
XMLIntf.IXMLDocument.LoadFromStream 7599
XMLIntf.IXMLDocument.LoadFromXML 7599
XMLIntf.IXMLDocument.Modified 7611
XMLIntf.IXMLDocument.Node 7611
XMLIntf.IXMLDocument.NodeIndentStr 7612
XMLIntf.IXMLDocument.Options 7612
XMLIntf.IXMLDocument.ParseOptions 7613
XMLIntf.IXMLDocument.Refresh 7600
XMLIntf.IXMLDocument.RegisterDocBinding 7600
XMLIntf.IXMLDocument.Resync 7601
XMLIntf.IXMLDocument.SaveToFile 7601
XMLIntf.IXMLDocument.SaveToStream 7602
XMLIntf.IXMLDocument.SaveToXML 7602
XMLIntf.IXMLDocument.SchemaRef 7613
XMLIntf.IXMLDocument.SetActive 7603
XMLIntf.IXMLDocument.SetDocumentElement 7603
XMLIntf.IXMLDocument.SetDOMDocument 7604
XMLIntf.IXMLDocument.SetEncoding 7604
XMLIntf.IXMLDocument.SetFileName 7604
XMLIntf.IXMLDocument.SetNodeIndentStr 7605
XMLIntf.IXMLDocument.SetOnAsyncLoad 7605
XMLIntf.IXMLDocument.SetOptions 7605
XMLIntf.IXMLDocument.SetParseOptions 7606
XMLIntf.IXMLDocument.SetStandAlone 7606
XMLIntf.IXMLDocument.SetVersion 7606
XMLIntf.IXMLDocument.SetXML 7607
XMLIntf.IXMLDocument.StandAlone 7614
XMLIntf.IXMLDocument.Version 7614
XMLIntf.IXMLDocument.XML 7614
XMLIntf.IXMLNode 7615
XMLIntf.IXMLNode.AddChild 7618
XMLIntf.IXMLNode.AttributeNodes 7634
XMLIntf.IXMLNode.Attributes 7634
XMLIntf.IXMLNode.ChildNodes 7635
XMLIntf.IXMLNode.ChildValues 7636
XMLIntf.IXMLNode.CloneNode 7619
XMLIntf.IXMLNode.Collection 7636
XMLIntf.IXMLNode.DeclareNamespace 7620
XMLIntf.IXMLNode.DOMNode 7637
XMLIntf.IXMLNode.FindNamespaceDecl 7620
XMLIntf.IXMLNode.FindNamespaceURI 7620
XMLIntf.IXMLNode.GetAttribute 7621
XMLIntf.IXMLNode.GetAttributeNodes 7621
XMLIntf.IXMLNode.GetAttributeNS 7622
XMLIntf.IXMLNode.GetChildNodes 7622
XMLIntf.IXMLNode.GetChildValue 7622
XMLIntf.IXMLNode.GetCollection 7623
XMLIntf.IXMLNode.GetDOMNode 7623
XMLIntf.IXMLNode.GetHasChildNodes 7624
XMLIntf.IXMLNode.GetIsTextElement 7624
XMLIntf.IXMLNode.GetLocalName 7624
XMLIntf.IXMLNode.GetNamespaceURI 7625

XMLIntf.IXMLNode.GetNodeName 7625
XMLIntf.IXMLNode.GetNodeType 7625
XMLIntf.IXMLNode.GetNodeValue 7626
XMLIntf.IXMLNode.GetOwnerDocument 7626
XMLIntf.IXMLNode.GetParentNode 7626
XMLIntf.IXMLNode.GetPrefix 7626
XMLIntf.IXMLNode.GetReadOnly 7627
XMLIntf.IXMLNode.GetText 7627
XMLIntf.IXMLNode.GetXML 7627
XMLIntf.IXMLNode.HasAttribute 7628
XMLIntf.IXMLNode.HasChildNodes 7637
XMLIntf.IXMLNode.IsTextElement 7638
XMLIntf.IXMLNode.LocalName 7638
XMLIntf.IXMLNode.NamespaceURI 7639
XMLIntf.IXMLNode.NextSibling 7628
XMLIntf.IXMLNode.NodeName 7639
XMLIntf.IXMLNode.NodeType 7640
XMLIntf.IXMLNode.NodeValue 7641
XMLIntf.IXMLNode.Normalize 7629
XMLIntf.IXMLNode.OwnerDocument 7641
XMLIntf.IXMLNode.ParentNode 7641
XMLIntf.IXMLNode.Prefix 7642
XMLIntf.IXMLNode.PreviousSibling 7629
XMLIntf.IXMLNode.ReadOnly 7642
XMLIntf.IXMLNode.Resync 7630
XMLIntf.IXMLNode.SetAttribute 7630
XMLIntf.IXMLNode.SetAttributeNS 7630
XMLIntf.IXMLNode.SetChildValue 7631
XMLIntf.IXMLNode.SetNodeValue 7631
XMLIntf.IXMLNode.SetReadOnly 7632
XMLIntf.IXMLNode.SetText 7632
XMLIntf.IXMLNode.Text 7643
XMLIntf.IXMLNode.TransformNode 7633
XMLIntf.IXMLNode.XML 7643
XMLIntf.IXMLNodeCollection 7644
XMLIntf.IXMLNodeCollection.Clear 7646
XMLIntf.IXMLNodeCollection.Count 7648
XMLIntf.IXMLNodeCollection.Delete 7646
XMLIntf.IXMLNodeCollection.GetCount 7647
XMLIntf.IXMLNodeCollection.GetNode 7647
XMLIntf.IXMLNodeCollection.Nodes 7649
XMLIntf.IXMLNodeCollection.Remove 7648
XMLIntf.IXMLNodeList 7649
XMLIntf.IXMLNodeList.Add 7651
XMLIntf.IXMLNodeList.BeginUpdate 7651
XMLIntf.IXMLNodeList.Clear 7652
XMLIntf.IXMLNodeList.Count 7660
XMLIntf.IXMLNodeList.Delete 7653
XMLIntf.IXMLNodeList.EndUpdate 7653
XMLIntf.IXMLNodeList.FindNode 7654
XMLIntf.IXMLNodeList.FindSibling 7655
XMLIntf.IXMLNodeList.First 7655
XMLIntf.IXMLNodeList.Get 7656
XMLIntf.IXMLNodeList.GetCount 7656
XMLIntf.IXMLNodeList.GetNode 7656
XMLIntf.IXMLNodeList.GetUpdateCount 7657
XMLIntf.IXMLNodeList.IndexOf 7657
XMLIntf.IXMLNodeList.Insert 7658
XMLIntf.IXMLNodeList.Last 7658
XMLIntf.IXMLNodeList.Nodes 7660
XMLIntf.IXMLNodeList.Remove 7659
XMLIntf.IXMLNodeList.ReplaceNode 7659
XMLIntf.IXMLNodeList.UpdateCount 7661
XMLIntf.NodeTypeNames 7668
XMLIntf.NodeTypeNames constant 7668
XMLIntf.TNodeType 7662
XMLIntf.TNodeType enumeration 7662
XMLIntf.TParseOption 7664
XMLIntf.TParseOption enumeration 7664
XMLIntf.TParseOptions 7667
XMLIntf.TParseOptions type 7667
XMLIntf.TXMLDocOption 7664
XMLIntf.TXMLDocOption enumeration 7664
XMLIntf.TXMLDocOptions 7667
XMLIntf.TXMLDocOptions type 7667
XMLIntf.TXMLEncodingType 7665
XMLIntf.TXMLEncodingType enumeration 7665
Xmlxform 7669

Xmlxform namespace 7669
Classes 7670
Types 7705
Xmlxform.TranslateException 7670
Xmlxform.TRowEvent 7705
Xmlxform.TRowEvent type 7705
Xmlxform.TTranslateEvent 7705
Xmlxform.TTranslateEvent type 7705
Xmlxform.TXMLTransform 7671
Xmlxform.TXMLTransform.AfterEachRow 7681
Xmlxform.TXMLTransform.AfterEachRowSet 7681
Xmlxform.TXMLTransform.BeforeEachRow 7682
Xmlxform.TXMLTransform.BeforeEachRowSet 7682
Xmlxform.TXMLTransform.Data 7675
Xmlxform.TXMLTransform.DoTransform 7673
Xmlxform.TXMLTransform.EmptyDestinationDocument 7676
Xmlxform.TXMLTransform.GetData 7673
Xmlxform.TXMLTransform.GetResultString 7673
Xmlxform.TXMLTransform.OnTranslate 7683
Xmlxform.TXMLTransform.ResultDocument 7677
Xmlxform.TXMLTransform.ResultString 7677
Xmlxform.TXMLTransform.SourceXml 7678
Xmlxform.TXMLTransform.SourceXmlDocument 7678
Xmlxform.TXMLTransform.SourceXmlFile 7679
Xmlxform.TXMLTransform.Transform 7673
Xmlxform.TXMLTransform.TransformationDocument 7679
Xmlxform.TXMLTransform.TransformationFile 7680
Xmlxform.TXMLTransform.TransformXML 7674
Xmlxform.TXMLTransform.Translate 7674
Xmlxform.TXMLTransformClient 7683
Xmlxform.TXMLTransformClient.ApplyUpdates 7686
Xmlxform.TXMLTransformClient.Create 7685
Xmlxform.TXMLTransformClient.Destroy 7685
Xmlxform.TXMLTransformClient.GetDataAsXml 7686
Xmlxform.TXMLTransformClient.ProviderName 7689
Xmlxform.TXMLTransformClient.RemoteServer 7689
Xmlxform.TXMLTransformClient.SetParams 7687
Xmlxform.TXMLTransformClient.SetupAppServer 7688
Xmlxform.TXMLTransformClient.TransformApplyUpdates 7690
Xmlxform.TXMLTransformClient.TransformGetData 7690
Xmlxform.TXMLTransformClient.TransformSetParams 7691
Xmlxform.TXMLTransformProvider 7691
Xmlxform.TXMLTransformProvider.AfterApplyUpdates 7698
Xmlxform.TXMLTransformProvider.AfterGetRecords 7699
Xmlxform.TXMLTransformProvider.AfterRowRequest 7700
Xmlxform.TXMLTransformProvider.BeforeApplyUpdates 7701
Xmlxform.TXMLTransformProvider.BeforeGetRecords 7701
Xmlxform.TXMLTransformProvider.BeforeRowRequest 7703
Xmlxform.TXMLTransformProvider.CacheData 7696
Xmlxform.TXMLTransformProvider.Create 7693
Xmlxform.TXMLTransformProvider.Destroy 7694
Xmlxform.TXMLTransformProvider.InternalApplyUpdates 7694
Xmlxform.TXMLTransformProvider.InternalGetRecords 7695
Xmlxform.TXMLTransformProvider.OnDataRequest 7703
Xmlxform.TXMLTransformProvider.TransformRead 7696
Xmlxform.TXMLTransformProvider.TransformWrite 7697
Xmlxform.TXMLTransformProvider.XMLDataFile 7697
xoQuote enumeration member 7292
XPActnCtrls 7707
XPActnCtrls namespace 7707
Classes 7708
XPActnCtrls.TXPStyleAddRemoveItem 7708
XPActnCtrls.TXPStyleAddRemoveItem.CalcBounds 7709
XPActnCtrls.TXPStyleAddRemoveItem.DrawBackground 7709
XPActnCtrls.TXPStyleAddRemoveItem.DrawGlyph 7710
XPActnCtrls.TXPStyleAddRemoveItem.DrawSeparator 7710
XPActnCtrls.TXPStyleAddRemoveItem.DrawText 7710
XPActnCtrls.TXPStyleAddRemoveItem.DrawUnusedEdges 7710
XPActnCtrls.TXPStyleButton 7711
XPActnCtrls.TXPStyleButton.DrawBackground 7711
XPActnCtrls.TXPStyleButton.DrawFrame 7712
XPActnCtrls.TXPStyleButton.DrawGlyph 7712
XPActnCtrls.TXPStyleButton.DrawText 7712
XPActnCtrls.TXPStyleCustomizePopup 7713
XPActnCtrls.TXPStyleCustomizePopup.Create 7713
XPActnCtrls.TXPStyleCustomizePopup.GetAddRemoveItemClass 7714

XPActnCtrls.TXPStyleCustomizePopup.GetDefaultColorMapClass 7714
XPActnCtrls.TXPStyleCustomizePopup.NCPaint 7714
XPActnCtrls.TXPStyleDropDownBtn 7714
XPActnCtrls.TXPStyleDropDownBtn.DrawBackground 7715
XPActnCtrls.TXPStyleDropDownBtn.DrawFrame 7716
XPActnCtrls.TXPStyleDropDownBtn.DrawGlyph 7716
XPActnCtrls.TXPStyleDropDownBtn.GetPopupClass 7716
XPActnCtrls.TXPStyleExpandBtn 7716
XPActnCtrls.TXPStyleExpandBtn.DrawBackground 7717
XPActnCtrls.TXPStyleExpandBtn.DrawFrame 7717
XPActnCtrls.TXPStyleMenuButton 7718
XPActnCtrls.TXPStyleMenuButton.DrawBackground 7718
XPActnCtrls.TXPStyleMenuButton.DrawText 7719
XPActnCtrls.TXPStyleMenuItem 7719
XPActnCtrls.TXPStyleMenuItem.CalcBounds 7720
XPActnCtrls.TXPStyleMenuItem.CalcLayout 7720
XPActnCtrls.TXPStyleMenuItem.DrawBackground 7721
XPActnCtrls.TXPStyleMenuItem.DrawGlyph 7721
XPActnCtrls.TXPStyleMenuItem.DrawSeparator 7721
XPActnCtrls.TXPStyleMenuItem.DrawShadowedText 7721
XPActnCtrls.TXPStyleMenuItem.DrawSubMenuGlyph 7722
XPActnCtrls.TXPStyleMenuItem.DrawText 7722
XPActnCtrls.TXPStyleMenuItem.DrawUnusedEdges 7722
XPActnCtrls.TXPStylePopupMenu 7722
XPActnCtrls.TXPStylePopupMenu.Create 7723
XPActnCtrls.TXPStylePopupMenu.Destroy 7723
XPActnCtrls.TXPStylePopupMenu.DisplayShadow 7724
XPActnCtrls.TXPStylePopupMenu.GetExpandBtnClass 7724
XPActnCtrls.TXPStylePopupMenu.HideShadow 7725
XPActnCtrls.TXPStylePopupMenu.NCPaint 7725
XPActnCtrls.TXPStylePopupMenu.PositionPopup 7725
XPActnCtrls.TXPStylePopupMenu.VisibleChanging 7725
XPActnCtrls.TXPStyleToolScrollBtn 7726
XPActnCtrls.TXPStyleToolScrollBtn.DrawBackground 7726
xpEncoding enumeration member 7448
XPMan 7729
XPMan namespace 7729
Classes 7730
XPMan.TXPManifest 7730
xpStandalone enumeration member 7448
XPStyleActnCtrls 7731
XPStyleActnCtrls namespace 7731
Classes 7732
Variables 7735
XPStyleActnCtrls.TXPStyleActionBars 7732
XPStyleActnCtrls.TXPStyleActionBars.GetAddRemoveItemClass 7733
XPStyleActnCtrls.TXPStyleActionBars.GetColorMapClass 7733
XPStyleActnCtrls.TXPStyleActionBars.GetControlClass 7733
XPStyleActnCtrls.TXPStyleActionBars.GetPopupClass 7734
XPStyleActnCtrls.TXPStyleActionBars.GetScrollBtnClass 7734
XPStyleActnCtrls.TXPStyleActionBars.GetStyleName 7734
XPStyleActnCtrls.XPStyle 7735
XPStyleActnCtrls.XPStyle variable 7735
xpVersion enumeration member 7448
XSBuiltIns 7737
XSBuiltIns namespace 7737
Classes 7739
Constants 7810
Functions 7805
Types 7809
XSBuiltIns.DateTimeToXMLTime 7805
XSBuiltIns.DateTimeToXMLTime function 7805
XSBuiltIns.DateTimeToXSDateTime 7805
XSBuiltIns.DateTimeToXSDateTime function 7805
XSBuiltIns.EXSDateTimeException 7739
XSBuiltIns.EXSDecimalException 7740
XSBuiltIns.EXSHexBinaryException 7740
XSBuiltIns.GetDataFromFile 7806
XSBuiltIns.GetDataFromFile function 7806
XSBuiltIns.IncrementAmount 7810
XSBuiltIns.IncrementAmount constant 7810
XSBuiltIns.InitXSTypes 7806
XSBuiltIns.InitXSTypes function 7806
XSBuiltIns.MaxDay 7811
XSBuiltIns.MaxDay constant 7811
XSBuiltIns.MaxMonth 7811

XSBuiltIns.MaxMonth constant 7811
XSBuiltIns.MinDay 7811
XSBuiltIns.MinDay constant 7811
XSBuiltIns.MinMonth 7812
XSBuiltIns.MinMonth constant 7812
XSBuiltIns.SDecimal 7812
XSBuiltIns.SDecimal constant 7812
XSBuiltIns.SHexMarker 7812
XSBuiltIns.SHexMarker constant 7812
XSBuiltIns.SLocalTimeMarker 7813
XSBuiltIns.SLocalTimeMarker constant 7813
XSBuiltIns.SNAN 7813
XSBuiltIns.SNAN constant 7813
XSBuiltIns.SNegative 7813
XSBuiltIns.SNegative constant 7813
XSBuiltIns.SoapDecimalSeparator 7814
XSBuiltIns.SoapDecimalSeparator constant 7814
XSBuiltIns.SoapFloatToStr 7807
XSBuiltIns.SoapFloatToStr function 7807
XSBuiltIns.SoapStrToFloat 7807
XSBuiltIns.SoapStrToFloat function 7807
XSBuiltIns.SoapTimePrefix 7814
XSBuiltIns.SoapTimePrefix constant 7814
XSBuiltIns.SPlus 7814
XSBuiltIns.SPlus constant 7814
XSBuiltIns.SSciNotationMarker 7815
XSBuiltIns.SSciNotationMarker constant 7815
XSBuiltIns.TXMLData 7741
XSBuiltIns.TXMLData.Create 7742
XSBuiltIns.TXMLData.Destroy 7742
XSBuiltIns.TXMLData.LoadFromXML 7743
XSBuiltIns.TXMLData.ObjectToSOAP 7743
XSBuiltIns.TXMLData.SOAPToObject 7744
XSBuiltIns.TXMLData.XMLNode 7745
XSBuiltIns.TXSBoolean 7745
XSBuiltIns.TXSBoolean.AsBoolean 7747
XSBuiltIns.TXSBoolean.NativeToXS 7746
XSBuiltIns.TXSBoolean.XSToNative 7746
XSBuiltIns.TXSCustomDateTime 7747
XSBuiltIns.TXSCustomDateTime.AsDateTime 7759
XSBuiltIns.TXSCustomDateTime.AsUTCDateTime 7760
XSBuiltIns.TXSCustomDateTime.Create 7749
XSBuiltIns.TXSCustomDateTime.Day 7760
XSBuiltIns.TXSCustomDateTime.Destroy 7749
XSBuiltIns.TXSCustomDateTime.FDateTime 7750
XSBuiltIns.TXSCustomDateTime.FractionalSeconds 7761
XSBuiltIns.TXSCustomDateTime.GetAsDateTime 7751
XSBuiltIns.TXSCustomDateTime.GetAsUTCDateTime 7751
XSBuiltIns.TXSCustomDateTime.GetDay 7751
XSBuiltIns.TXSCustomDateTime.GetFractionalSeconds 7752
XSBuiltIns.TXSCustomDateTime.GetHour 7752
XSBuiltIns.TXSCustomDateTime.GetHourOffset 7752
XSBuiltIns.TXSCustomDateTime.GetMillisecond 7752
XSBuiltIns.TXSCustomDateTime.GetMinute 7753
XSBuiltIns.TXSCustomDateTime.GetMinuteOffset 7753
XSBuiltIns.TXSCustomDateTime.GetMonth 7753
XSBuiltIns.TXSCustomDateTime.GetSecond 7754
XSBuiltIns.TXSCustomDateTime.GetUseZeroMilliseconds 7754
XSBuiltIns.TXSCustomDateTime.GetYear 7754
XSBuiltIns.TXSCustomDateTime.Hour 7761
XSBuiltIns.TXSCustomDateTime.HourOffset 7762
XSBuiltIns.TXSCustomDateTime.Millisecond 7762
XSBuiltIns.TXSCustomDateTime.Minute 7763
XSBuiltIns.TXSCustomDateTime.MinuteOffset 7763
XSBuiltIns.TXSCustomDateTime.Month 7764
XSBuiltIns.TXSCustomDateTime.Second 7764
XSBuiltIns.TXSCustomDateTime.SetAsDateTime 7755
XSBuiltIns.TXSCustomDateTime.SetAsUTCDateTime 7755
XSBuiltIns.TXSCustomDateTime.SetDay 7755
XSBuiltIns.TXSCustomDateTime.SetFractionalSeconds 7755
XSBuiltIns.TXSCustomDateTime.SetHour 7756
XSBuiltIns.TXSCustomDateTime.SetHourOffset 7756
XSBuiltIns.TXSCustomDateTime.SetMillisecond 7756
XSBuiltIns.TXSCustomDateTime.SetMinute 7757
XSBuiltIns.TXSCustomDateTime.SetMinuteOffset 7757
XSBuiltIns.TXSCustomDateTime.SetMonth 7757
XSBuiltIns.TXSCustomDateTime.SetSecond 7758
XSBuiltIns.TXSCustomDateTime.SetUseZeroMilliseconds 7758

XSBuiltIns.TXSCustomDateTime.SetYear 7758
XSBuiltIns.TXSCustomDateTime.UseZeroMilliseconds 7765
XSBuiltIns.TXSCustomDateTime.Year 7765
XSBuiltIns.TXSDate 7766
XSBuiltIns.TXSDate.AsDate 7771
XSBuiltIns.TXSDate.Clone 7768
XSBuiltIns.TXSDate.Create 7767
XSBuiltIns.TXSDate.Day 7771
XSBuiltIns.TXSDate.GetAsDate 7768
XSBuiltIns.TXSDate.GetDay 7768
XSBuiltIns.TXSDate.GetMonth 7768
XSBuiltIns.TXSDate.GetYear 7769
XSBuiltIns.TXSDate.Month 7771
XSBuiltIns.TXSDate.NativeToXS 7769
XSBuiltIns.TXSDate.SetAsDate 7769
XSBuiltIns.TXSDate.SetDay 7769
XSBuiltIns.TXSDate.SetMonth 7770
XSBuiltIns.TXSDate.SetYear 7770
XSBuiltIns.TXSDate.XSToNative 7770
XSBuiltIns.TXSDate.Year 7772
XSBuiltIns.TXSDateTime 7772
XSBuiltIns.TXSDateTime.Clone 7774
XSBuiltIns.TXSDateTime.CompareDateTimeParam 7775
XSBuiltIns.TXSDateTime.Millisecond 7776
XSBuiltIns.TXSDateTime.NativeToXS 7775
XSBuiltIns.TXSDateTime.XSToNative 7775
XSBuiltIns.TXSDecimal 7776
XSBuiltIns.TXSDecimal.AsBcd 7779
XSBuiltIns.TXSDecimal.DecimalString 7779
XSBuiltIns.TXSDecimal.NativeToXS 7778
XSBuiltIns.TXSDecimal.XSToNative 7778
XSBuiltIns.TXSDuration 7780
XSBuiltIns.TXSDuration.Create 7782
XSBuiltIns.TXSDuration.DecimalSecond 7784
XSBuiltIns.TXSDuration.GetDecimalSecond 7782
XSBuiltIns.TXSDuration.NativeToXS 7783
XSBuiltIns.TXSDuration.SetDecimalSecond 7783
XSBuiltIns.TXSDuration.XSToNative 7783
XSBuiltIns.TXSHexBinary 7784
XSBuiltIns.TXSHexBinary.AsByteArray 7786
XSBuiltIns.TXSHexBinary.HexBinaryString 7787
XSBuiltIns.TXSHexBinary.NativeToXS 7785
XSBuiltIns.TXSHexBinary.XSToNative 7786
XSBuiltIns.TXSInteger 7787
XSBuiltIns.TXSInteger.AsInteger 7789
XSBuiltIns.TXSInteger.NativeToXS 7788
XSBuiltIns.TXSInteger.XSToNative 7788
XSBuiltIns.TXSLong 7789
XSBuiltIns.TXSLong.AsLong 7791
XSBuiltIns.TXSLong.NativeToXS 7790
XSBuiltIns.TXSLong.XSToNative 7790
XSBuiltIns.TXSString 7791
XSBuiltIns.TXSString.NativeToXS 7792
XSBuiltIns.TXSString.XSToNative 7792
XSBuiltIns.TXSTime 7792
XSBuiltIns.TXSTime.AsTime 7800
XSBuiltIns.TXSTime.Clone 7795
XSBuiltIns.TXSTime.FractionalSeconds 7801
XSBuiltIns.TXSTime.FractionalSecondString 7801
XSBuiltIns.TXSTime.GetAsTime 7795
XSBuiltIns.TXSTime.GetFractionalSeconds 7795
XSBuiltIns.TXSTime.GetFractionalSecondString 7796
XSBuiltIns.TXSTime.GetHour 7796
XSBuiltIns.TXSTime.GetHourOffset 7796
XSBuiltIns.TXSTime.GetMillisecond 7796
XSBuiltIns.TXSTime.GetMinute 7796
XSBuiltIns.TXSTime.GetMinuteOffset 7797
XSBuiltIns.TXSTime.GetSecond 7797
XSBuiltIns.TXSTime.Hour 7801
XSBuiltIns.TXSTime.HourOffset 7802
XSBuiltIns.TXSTime.Millisecond 7802
XSBuiltIns.TXSTime.Minute 7802
XSBuiltIns.TXSTime.MinuteOffset 7803
XSBuiltIns.TXSTime.NativeToXS 7797
XSBuiltIns.TXSTime.Second 7803
XSBuiltIns.TXSTime.SetAsTime 7797
XSBuiltIns.TXSTime.SetFractionalSeconds 7798
XSBuiltIns.TXSTime.SetHour 7798

XSBuiltIns.TXSTime.SetHourOffset 7798
XSBuiltIns.TXSTime.SetMillisecond 7798
XSBuiltIns.TXSTime.SetMinute 7799
XSBuiltIns.TXSTime.SetMinuteOffset 7799
XSBuiltIns.TXSTime.SetSecond 7799
XSBuiltIns.TXSTime.UseZeroMilliseconds 7804
XSBuiltIns.TXSTime.XSToNative 7799
XSBuiltIns.TXSTimeInstant 7809
XSBuiltIns.TXSTimeInstant type 7809
XSBuiltIns.XMLDateSeparator 7815
XSBuiltIns.XMLDateSeparator constant 7815
XSBuiltIns.XMLDayPos 7815
XSBuiltIns.XMLDayPos constant 7815
XSBuiltIns.XMLDefaultYearDigits 7816
XSBuiltIns.XMLDefaultYearDigits constant 7816
XSBuiltIns.XMLDurationDay 7816
XSBuiltIns.XMLDurationDay constant 7816
XSBuiltIns.XMLDurationHour 7816
XSBuiltIns.XMLDurationHour constant 7816
XSBuiltIns.XMLDurationMinute 7817
XSBuiltIns.XMLDurationMinute constant 7817
XSBuiltIns.XMLDurationMonth 7817
XSBuiltIns.XMLDurationMonth constant 7817
XSBuiltIns.XMLDurationSecond 7817
XSBuiltIns.XMLDurationSecond constant 7817
XSBuiltIns.XMLDurationStart 7818
XSBuiltIns.XMLDurationStart constant 7818
XSBuiltIns.XMLDurationYear 7818
XSBuiltIns.XMLDurationYear constant 7818
XSBuiltIns.XMLHourOffsetMinusMarker 7818
XSBuiltIns.XMLHourOffsetMinusMarker constant 7818
XSBuiltIns.XMLHourOffsetPlusMarker 7819
XSBuiltIns.XMLHourOffsetPlusMarker constant 7819
XSBuiltIns.XMLMilSecPos 7819
XSBuiltIns.XMLMilSecPos constant 7819
XSBuiltIns.XMLMonthPos 7819
XSBuiltIns.XMLMonthPos constant 7819
XSBuiltIns.XMLTimeSeparator 7820
XSBuiltIns.XMLTimeSeparator constant 7820
XSBuiltIns.XMLTimeToDateTime 7807
XSBuiltIns.XMLTimeToDateTime function 7807
XSBuiltIns.XMLYearPos 7820
XSBuiltIns.XMLYearPos constant 7820
XSLProd 7821
XSLProd namespace 7821
Classes 7822
XSLProd.EXSLPageProducerError 7822
XSLProd.TCustomXSLPageProducer 7822
XSLProd.TCustomXSLPageProducer.Content 7829
XSLProd.TCustomXSLPageProducer.ContentFromStream 7829
XSLProd.TCustomXSLPageProducer.ContentFromString 7830
XSLProd.TCustomXSLPageProducer.ContentFromWideString 7830
XSLProd.TCustomXSLPageProducer.Dispatcher 7835
XSLProd.TCustomXSLPageProducer.DispatcherComponent 7836
XSLProd.TCustomXSLPageProducer.GetAppDispatcher 7831
XSLProd.TCustomXSLPageProducer.GetProducerTemplateFileName 7831
XSLProd.TCustomXSLPageProducer.GetProducerTemplateStream 7831
XSLProd.TCustomXSLPageProducer.GetTemplateFileType 7831
XSLProd.TCustomXSLPageProducer.GetTemplateStream 7832
XSLProd.TCustomXSLPageProducer.GetXMLData 7832
XSLProd.TCustomXSLPageProducer.GetXMLDocument 7832
XSLProd.TCustomXSLPageProducer.GetXSLData 7832
XSLProd.TCustomXSLPageProducer.HasHTMLBrowserView 7832
XSLProd.TCustomXSLPageProducer.HasScriptView 7833
XSLProd.TCustomXSLPageProducer.HasXMLBrowserView 7833
XSLProd.TCustomXSLPageProducer.HasXSLBrowserView 7833
XSLProd.TCustomXSLPageProducer.Notification 7833
XSLProd.TCustomXSLPageProducer.ProduceContent 7834
XSLProd.TCustomXSLPageProducer.ProduceContentFromStream 7834
XSLProd.TCustomXSLPageProducer.ProduceContentFromString

7834
XSLProd.TCustomXSLPageProducer.SetAppDispatcher 7835
XSLProd.TCustomXSLPageProducer.XMLData 7836
XSLProd.TXSLPageProducer 7837
XSLProd.TXSLPageProducer.XMLData 7843

Z

zcDefault enumeration member 7869
zcFastest enumeration member 7869
zcMax enumeration member 7869
zcNone enumeration member 7869
ZLib 7845
ZLib namespace 7845
Classes 7848
Constants 7874
Functions 7859
Structs, Records, Enums 7868
Types 7871
ZLib._z_errmsg 7874
ZLib._z_errmsg constant 7874
ZLib.adler32 7859
ZLib.adler32 function 7859
ZLib.deflate 7860
ZLib.deflate function 7860
ZLib.deflateEnd 7860
ZLib.deflateEnd function 7860
ZLib.deflateInit_ 7860
ZLib.deflateInit_ function 7860
ZLib.DeflateInit2_ 7861
ZLib.DeflateInit2_ function 7861
ZLib.EZCompressionError 7848
ZLib.EZDecompressionError 7848
ZLib.EZLibError 7849
ZLib.inflate 7861
ZLib.inflate function 7861
ZLib.inflateEnd 7861
ZLib.inflateEnd function 7861
ZLib.inflateInit_ 7862
ZLib.inflateInit_ function 7862
ZLib.inflateInit2_ 7862
ZLib.inflateInit2_ function 7862
ZLib.inflateReset 7862
ZLib.inflateReset function 7862
ZLib.MoveI32 7863
ZLib.MoveI32 function 7863
ZLib.SZInvalid 7875
ZLib.SZInvalid constant 7875
ZLib.TCompressionLevel 7868
ZLib.TCompressionLevel enumeration 7868
ZLib.TCompressionStream 7871
ZLib.TCompressionStream type 7871
ZLib.TCustomZLibStream 7872
ZLib.TCustomZLibStream type 7872
ZLib.TCustomZStream 7849
ZLib.TCustomZStream.Create 7850
ZLib.TCustomZStream.DoProgress 7850
ZLib.TCustomZStream.OnProgress 7850
ZLib.TDecompressionStream 7872
ZLib.TDecompressionStream type 7872
ZLib.TZAlloc 7872
ZLib.TZAlloc type 7872
ZLib.TZCompressionLevel 7869
ZLib.TZCompressionLevel enumeration 7869
ZLib.TZCompressionStream 7851
ZLib.TZCompressionStream.CompressionRate 7854
ZLib.TZCompressionStream.Create 7852
ZLib.TZCompressionStream.Destroy 7852
ZLib.TZCompressionStream.OnProgress 7855
ZLib.TZCompressionStream.Read 7853
ZLib.TZCompressionStream.Seek 7854
ZLib.TZCompressionStream.Write 7854
ZLib.TZDecompressionStream 7855
ZLib.TZDecompressionStream.Create 7856
ZLib.TZDecompressionStream.Destroy 7856
ZLib.TZDecompressionStream.OnProgress 7858
ZLib.TZDecompressionStream.Read 7857
ZLib.TZDecompressionStream.Seek 7857
ZLib.TZDecompressionStream.Write 7857

ZLib.TZFree 7873
ZLib.TZFree type 7873
ZLib.TZStreamRec 7869
ZLib.TZStreamRec record 7869
ZLib.Z_ASCII 7875
ZLib.Z_ASCII constant 7875
ZLib.Z_BEST_COMPRESSION 7875
ZLib.Z_BEST_COMPRESSION constant 7875
ZLib.Z_BEST_SPEED 7876
ZLib.Z_BEST_SPEED constant 7876
ZLib.Z_BINARY 7876
ZLib.Z_BINARY constant 7876
ZLib.Z_BUF_ERROR 7876
ZLib.Z_BUF_ERROR constant 7876
ZLib.Z_DATA_ERROR 7877
ZLib.Z_DATA_ERROR constant 7877
ZLib.Z_DEFAULT_COMPRESSION 7877
ZLib.Z_DEFAULT_COMPRESSION constant 7877
ZLib.Z_DEFAULT_STRATEGY 7877
ZLib.Z_DEFAULT_STRATEGY constant 7877
ZLib.Z_DEFLATED 7878
ZLib.Z_DEFLATED constant 7878
ZLib.Z_ERRNO 7878
ZLib.Z_ERRNO constant 7878
ZLib.Z_FILTERED 7878
ZLib.Z_FILTERED constant 7878
ZLib.Z_FINISH 7879
ZLib.Z_FINISH constant 7879
ZLib.Z_FULL_FLUSH 7879
ZLib.Z_FULL_FLUSH constant 7879
ZLib.Z_HUFFMAN_ONLY 7879
ZLib.Z_HUFFMAN_ONLY constant 7879
ZLib.Z_MEM_ERROR 7880
ZLib.Z_MEM_ERROR constant 7880
ZLib.Z_NEED_DICT 7880
ZLib.Z_NEED_DICT constant 7880
ZLib.Z_NO_COMPRESSION 7880
ZLib.Z_NO_COMPRESSION constant 7880
ZLib.Z_NO_FLUSH 7881
ZLib.Z_NO_FLUSH constant 7881
ZLib.Z_OK 7881
ZLib.Z_OK constant 7881
ZLib.Z_PARTIAL_FLUSH 7881
ZLib.Z_PARTIAL_FLUSH constant 7881
ZLib.Z_STREAM_END 7882
ZLib.Z_STREAM_END constant 7882
ZLib.Z_STREAM_ERROR 7882
ZLib.Z_STREAM_ERROR constant 7882
ZLib.Z_SYNC_FLUSH 7882
ZLib.Z_SYNC_FLUSH constant 7882
ZLib.Z_UNKNOWN 7883
ZLib.Z_UNKNOWN constant 7883
ZLib.Z_VERSION_ERROR 7883
ZLib.Z_VERSION_ERROR constant 7883
ZLib.ZCompress 7863
ZLib.ZCompress function 7863
ZLib.ZCompressStr 7864
ZLib.ZCompressStr function 7864
ZLib.ZCompressStream 7864, 7865
ZLib.ZCompressStream function 7864, 7865
ZLib.ZDecompress 7865
ZLib.ZDecompress function 7865
ZLib.ZDecompressStr 7865
ZLib.ZDecompressStr function 7865
ZLib.ZDecompressStream 7866
ZLib.ZDecompressStream function 7866
ZLib.ZLevels 7883
ZLib.ZLevels constant 7883
ZLib.ZLIB_VERSION 7884
ZLib.ZLIB_VERSION constant 7884
ZLib.zlibAllocMem 7866
ZLib.zlibAllocMem function 7866
ZLib.zlibFreeMem 7866
ZLib.zlibFreeMem function 7866
ZLib.ZSendToBrowser 7867
ZLib.ZSendToBrowser function 7867
ZLibConst 7885
ZLibConst namespace 7885

Constants 7886
ZLibConst.sError 7886
ZLibConst.sError constant 7886
ZLibConst.sInvalidStreamOp 7886
ZLibConst.sInvalidStreamOp constant 7886
ZLibConst.sTargetBufferTooSmall 7886
ZLibConst.sTargetBufferTooSmall constant 7886